Dependable Computing

A Multilevel Approach

Behrooz Parhami

University of California, Santa Barbara

parhami@ece.ucsb.edu
http://www.ece.ucsb.edu/~parhami

This is a draft of the forthcoming book
Dependable Computing: A Multilevel Approach,
by Behrooz Parhami, publisher TBD
ISBN TBD; Call number TBD

Dedication

To my academic mentors of long ago:

Professor Robert Allen Short (1927-2003),
who taught me digital systems theory
and encouraged me to publish my first research paper on
stochastic automata and reliable sequential machines,

and

Professor Algirdas Antanas Avižienis (1932-)
who provided me with a comprehensive overview
of the dependable computing discipline
and oversaw my maturation as a researcher.

About the Cover

The cover design shown is a placeholder. It will be replaced by the actual cover image once the design becomes available. The two elements in this image convey the ideas that computer system dependability is a weakest-link phenomenon and that modularization & redundancy can be employed, in a manner not unlike the practice in structural engineering, to prevent failures or to limit their impact.

Preface

"The skill of writing is to create a context in which other people can think."

Edwin Schlossberg

"When a complex system succeeds, that success masks its proximity to failure. ... Thus, the failure of the *Titanic* contributed much more to the design of safe ocean liners than would have her success. That is the paradox of engineering and design."

Henry Petroski, Success through Failure: The Paradox of Design

The Context of Dependable Computing

Accounts of computer system errors, failures, and other mishaps range from humorous to horrifying. On the lighter side, we have data entry or computation errors that lead to an electric company sending a bill for hundreds of thousands of dollars to a small residential customer. Such errors cause a chuckle or two as they are discussed at the dinner table and are usually corrected to everyone's satisfaction, leaving no permanent damage (though in a few early occurrences of this type, some customers suffered from their power being disconnected due to nonpayment of the erroneous bill). At the other extreme, there are dire consequences such as an airliner with hundreds of passengers on board crashing, two high-speed commuter trains colliding, a nuclear reactor taken to the brink of meltdown, or financial markets in a large industrial country collapsing. Causes of such annoying or catastrophic behavior range from design deficiencies, interaction of several rare or unforeseen events, operator errors, external disturbances, or malicious actions.

In nearly all engineering and industrial disciplines, quality control is an integral part of the design and manufacturing processes. There are also standards and guidelines that make at least certain aspects of quality assurance more or less routine. This is far from being the case in computer engineering, particularly with regard to software products. True, we do offer dependable computing courses to our students, but in doing so, we create an undesirable separation between design and dependability concerns. A structural engineer does not learn about bridge-building in one course and about ensuring that bridges do not collapse in another. A toaster or steam-iron manufacturer does not ship its products with a warning label that there is no guarantee that the device will prepare toast or remove wrinkles from clothing and that the manufacturer will not be liable for any harm resulting from their use.

The field of dependable (aka fault-tolerant) computing was born in the late 1960s, because longevity, safety, and robustness requirements of space and military applications could not be met through conventional design. Space applications presented the need for long-life systems, with either no possibility of on-board maintenance or repair (unmanned missions) or with stringent reliability requirements in harsh, and relatively unpredictable environments (manned missions). Military applications required extreme robustness in the face of punishing operational conditions, partial system wipeout, or targeted attacks by a sophisticated adversary. Early researchers of the field were thus predominantly supported by aerospace and defense organizations.

Of course, designing computer systems that were robust, self-reconfiguring, and ultimately self-repairing was only one part of the problem. There was also a need to quantify various aspects of system dependability via detailed and faithful models that would allow the designers of such systems to gain the confidence and trust of the user community. A failure probability of one-in-a-billion, say, was simply too low to allow experimental assessment and validation, as such experiments would have required the construction and testing of many millions of copies of the target system in order to allow general conclusions to be drawn with reasonable confidence. Thus, research-and-development teams in dependable computing pursued analytical and simulation models to help with the evaluation process. Extending and fine-tuning of such models is one of the main activity threads in the field.

As the field matured, application areas broadened beyond aerospace and military systems and they now include a wide array of domains, from automotive computers to large redundant disk arrays. In fact, many of the methods discussed in this book are routinely utilized even in contexts that do not satisfy the traditional definitions of high-risk or safety-critical systems, although the most elaborate techniques continue to be developed for systems whose failure would endanger human lives. Systems in the latter category include:

– Advanced infrastructure and transportation systems, such as high-speed trains
– Process control in hazardous environments, such as nuclear power plants
– Patient monitoring and emergency health-care procedures, as in surgical robots

Such advanced techniques then trickle down and eventually find their way into run-of-the-mill computer systems, such as traditional desktop and laptop computers.

Scope and Features

The field of dependable computing has matured to the point that a dozen or so texts and reference books have been published. Some of these books that cover dependable computing in general (as opposed to special aspects or ad-hoc/unconventional methods) are listed at the end of this preface. Each of these books possesses unique strengths and has contributed to the formation and fruition of the field. The current text, *Dependable Computing: A Multilevel Approach*, exposes the reader to the basic concpets of dependable computing in sufficient detail to enable their use in many hardware/software contexts. Covered methods include monitoring, redundancy, error detection/correction, self-test, self-check, self-repair, adjudication, and fail-soft operation. The book is an outgrowth of lecture notes that the author has developed and refined over many years. Here are the most important features of this text in comparison with the listed books:

a. *Division of material into lecture-size chapters*: In my approach to teaching, a lecture is a more or less self-contained module with links to past lectures and pointers to what will transpire in future. Each lecture must have a theme or title and must proceed from motivation, to details, to conclusion. In designing the text, I have strived to divide the material into chapters, each of which is suitable for one lecture (1-2 hours). A short lecture can cover the first few subsections while a longer lecture can deal with variations, peripheral ideas, or more advanced material near the end of the chapter. To make the structure hierarchical, as opposed to flat or linear, lectures are grouped into seven parts, each composed of four lectures and covering one level in our multilevel model (see the figure at the end of this preface).

b. *Emphasis on both the underlying theory and actual system designs*: The ability to cope with complexity requires both a deep knowledge of the theoretical underpinnings of dependable computing and examples of designs that help us understand the theory. Such designs also provide building blocks for synthesis as well as reference points for cost-performance comparisons. This viewpoint is reflected, for example, in the detailed coverage of error-coding techniques that later lead to a better understanding of various of self-checking design methods and redundant disk-arrays (Part IV). Another example can be found in Chapter 17 where the rigorous discussion of malfunction diagnosis allows a more systematic treatment of reconfiguration and self-repair.

c. *Linking dependable computing to other subfields of computing*: Dependable computing is nourished by, and in turn feeds other subfields of computer systems and technology. Examples of such links abound. Parallel and distributed computing is a case in point, given that such systems contain multiple resources of each kind and thus offer the possibility of sharing spare subsystems for greater efficiency. In fact, one can even find links to topics outside traditional science and engineering disciplines. For example, designers of redundant systems with replication and voting can learn a great deal from the treatment of voting systems by mathematicians and social scientists. Such links are pointed out and pursued throughout the text.

d. *Wide coverage of important topics*: The current text covers virtually all important algorithmic and hardware design topics in dependable computing, thus providing a balanced and complete view of the field. Coverage of testable design, voting algorithms, software redundancy, and fail-safe systems do not all appear in other textbooks.

e. *Unified and consistent notation/terminology throughout the text*: Every effort is made to use consistent notation/terminology throughout the text. For example, R always stands for reliability and s for the number of spare units. While other authors have done this in the basic parts of their texts, there is a tendency to cover more advanced research topics by simply borrowing the notation and terminology from the reference source. Such an approach has the advantage of making the transition between reading the text and the original reference source easier, but it is utterly confusing to the majority of the students who rely on the text and do not consult the original references except, perhaps, to write a research paper.

Summary of Topics

The seven parts of this book, each composed of four chapters, have been written with the following goals:

Part I sets the stage, gives a taste of what is to come, and provides a detailed perspective on the assessment of dependability in computing systems and the modeling tools needed for this purpose.

Part II deals with impairments to dependability at the physical (device) level, how they may lead to system vulnerability and low integrated-circuit yield, and what countermeasures are available for dealing with them.

Part III deals with impairments to dependability at the logical (circuit) level, how the resulting faults can affect system behavior, and how redundancy methods can be used to deal with them.

Part IV covers information-level impairments that lead to data-path and control errors, methods for detecting/correcting such errors, and ways of preventing such errors from propagating and causing problems at even higher levels of abstraction.

Part V deals with everything that can go wrong at the architectural level, that is, at the level of interactions between subsystems, be they parts of a single computer or nodes in a widely distributed system.

Part VI covers service-level impairments that may cause a system not to be able to perform the required tasks, even though it has not totally failed in an absolute sense.

Part VII deals with breaches at the computation-result or outcome level, where the success or failure of a computing system is ultimately judged and the costs of aberrant results or actions must be borne.

Pointers on How to Use the Book

For classroom use, the topics in each chapter of this text can be covered in a lecture spanning 1-2 hours. In my own teaching, I have used the chapters primarily for 1.5-hour lectures, twice a week, in a 10-week quarter, omitting or combining some chapters to fit the material into 16-20 lectures. But the modular structure of the text lends itself to other lecture formats, self-study, or review of the field by practitioners. In the latter two cases, the readers can view each chapter as a study unit (for one week, say) rather than as a lecture. Ideally, all topics in each chapter should be covered before moving to the next chapter. However, if fewer lecture hours are available, then some of the subsections located at the end of chapters can be omitted or introduced only in terms of motivations and key results.

Problems of varying complexity, from straightforward numerical examples or exercises to more demanding studies or mini-projects, have been supplied for each chapter. These problems form an integral part of the book and have not been added as afterthoughts to make the book more attractive for use as a text. A total of xxx problems are included (xx-xx per chapter). Assuming that two lectures are given per week, either weekly or biweekly homework can be assigned, with each assignment having the specific coverage of the respective half-part (two chapters) or full part (four chapters) as its "title".

An instructor's solutions manual is planned. The author's detailed syllabus for the course ECE 257A at UCSB is available at

http://www.ece.ucsb.edu/~parhami/ece_257a.htm

References to classical papers in dependable computing key design ideas, and important state-of-the-art research contributions are listed at the end of each chapter. These references provide good starting points for doing in-depth studies or for preparing term papers/projects. New ideas in the field of dependable computing appear in papers presented at an annual technical gathering, the Dependable Systems and Networks (DSN) conference, sponsored jointly by Institute of Electrical and Electronics Engineers (IEEE) and International Federation for Information Processing (IFIP). DSN, which was formed by merging meetings sponsored separately by IEEE and IFIP, covres all aspects of the field, including techniques for dependable computing and communications, as well as performance and other implications of dependability features.

Other conferences include Pacific Rim International Symposium on Dependable Computing [PRDC], European Dependable Computing Conference [EDCC], Symposium on Reliable Distributed Systems [SRDS], and International Conference on Computer Design [ICCD]. The field's most pertinent archival journals are *IEEE Transactions on Dependable and Secure Computing* [TDSC], *IEEE Transactions on Reliability* [TRel], *IEEE Transactions on Computers* [TCom], and *The Computer Journal* [ComJ].

Acknowledgments

The current text, *Dependable Computing: A Multilevel Approach*, is an outgrowth of lecture notes that I have used for the graduate course "ECE 257A: Fault-Tolerant Computing" at the University of California, Santa Barbara, and, in rudimentary forms, at several other institutions prior to 1988. The text has benefited greatly from keen observations, curiosity, and encouragement of my many students in these courses. A sincere thanks to all of them!

General References

The list that follows contains references of two kinds: (1) books that have greatly influenced the current text and (2) general reference sources for in-depth study or research. Books and other information sources that are relevant to specific chapters are listed in the end-of-chapter reference lists.

[Ande81] Anderson, T. and P. A. Lee, *Fault Tolerance: Principles and Practice*, Prentice Hall, 1981.

[Ande85] Anderson, T. A. (ed.), *Resilient Computing Systems*, Collins, 1985. Also: Wiley, 1986.

[ComJ] *The Computer Journal*, published monthly by the British Computer Society.

[Comp] *IEEE Computer*, magazine published by the IEEE Computer Society. Has published several special issues on dependable computing: Vol. 17, No. 6, August 1984; Vol. 23, No. 5, July 1990.

[CSur] *ACM Computing Surveys*, journal published by the Association for Computing Machinery.

[DCCA] Dependable Computing for Critical Applications, series of conferences later merged into DSN.

[Diab05] Diab, H. B. and A. Y. Zomaya (eds.), *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, Wiley, 2005.

[DSN] *Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks*, Conference formed by merging earlier series of meetings, the oldest of which (FTCS) dated back to 1971. URL: http://www.dsn.org

[Dubr13] Dubrova, E., *Fault-Tolerant Design*, Springer, 185 pp., 2013.

[Dunn02] Dunn, W. R., *Practical Design of Safetry-Critical Computer Systems*, Reliability Press, 2002.

[EDCC] *Proc. European Dependable Computing Conf.*, Conference held 10 times from 1994 to 2014 and turned into an annual event in 2015, when it merged with the European Workshop on Dependable Computing. URL: http://edcc2015.lip6.fr/

[FTCS] IEEE Symp. Fault-Tolerant Computing, series of annual symposia, begun in 1971 and eventually merged into DSN.

[Geff02] Geffroy, J.-C. and G. Motet, *Design of Dependable Computing Systems*, Springer, 2002.

[Gray85] Gray, J., "Why Do Computers Stop and What Can Be Done About It?" Technical Report TR85.7, Tandem Corp., 1985.

[ICCD] *Proc. IEEE Int'l Conf. Computer Design*, sponsored annually by the IEEE Computer Society since 1983.

[IFIP] Web site of the International Federation for Information Processing Working Group WG 10.4 on Dependable Computing. http://www.dependability.org/wg10.4

[Jalo94] Jalote, P., *Fault Tolerance in Distributed Systems*, Prentice Hall, 1994.

[John89] Johnson, B. W., *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.

[Knig12] Knight, J., *Fundamentals of Dependable Computing for Software Engineers*, CRC Press, 2012.

[Kore07] Koren, I. and C. M. Krishna, *Fault-Tolerant Systems*, Morgan Kaufmann, 2nd ed., 2020.

[Lala01] Lala, P. K., *Self-Checking and Fault-Tolerant Digital Design*, Morgan Kaufmann, 2001.

[Levi94] Levi, S.-T. and A. K. Agrawala, *Fault-Tolerant System Design*, McGraw-Hill. 1994.

[Negr89] Negrini, R., M. G. Sami, and R. Stefanelli, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, MIT Press, 1989.

[Nels87] Nelson, V. P. and B. D. Carroll (eds.), *Tutorial: Fault-Tolerant Computing*, IEEE Computer Society Press, 1987.

[Perr99] Perrow, C., *Normal Accidents: Living with High-Risk Technologies*, Princeton U. Press, 1999.

[Perr07] Perrow, C., *The Next Catastrophe: Reducing Our Vulnerabilities to Natural, Industrial, and Terrorist Disasters*, Princeton U. Press, 2007.

[Prad86] Pradhan, D. K. (ed.), *Fault-Tolerant Computing: Theory and Techniques*, 2 Vols., Prentice Hall, 1986.

[Prad96] Pradhan, D. K. (ed.), *Fault-Tolerant Computer System Design*, Prentice Hall, 1996.

[PRDC] *Proc. IEEE Pacific Rim Int'l Symp. Dependable Computing*, sponsored by IEEE Computer Society and held every 1-2 years since 1989.

[Rand13] Randall, B., J.-C. Laprie, H. Kopetx, and B. Littlewood (eds.), *Predictably Dependable Computing Systems*, Springer, 2013.

[Shoo02] Shooman, M. L., *Reliability of Computer Systems and Networks*, Wiley, 2002.

[Siew82] Siewiorek, D. P. and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.

[Siew92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 2nd Edition, 1992. Also: A. K. Peters, 1998.

[Sori09] Sorin, D., *Fault Tolerant Computer Architecture*, Morgan & Claypool, 2009.

[SRDS] *Proc. IEEE Symp. Reliable Distributed Systems*, sponsored annually by IEEE Computer Society.

[TCom] *IEEE Trans. Computers*, journal published by the IEEE Computer Society. Has publlished a number of sepecial issues on dependable computing: Vol. 41, No. 2, February 1992; Vol. 47, No. 4, April 1998; Vol. 51, No. 2, February 2002.

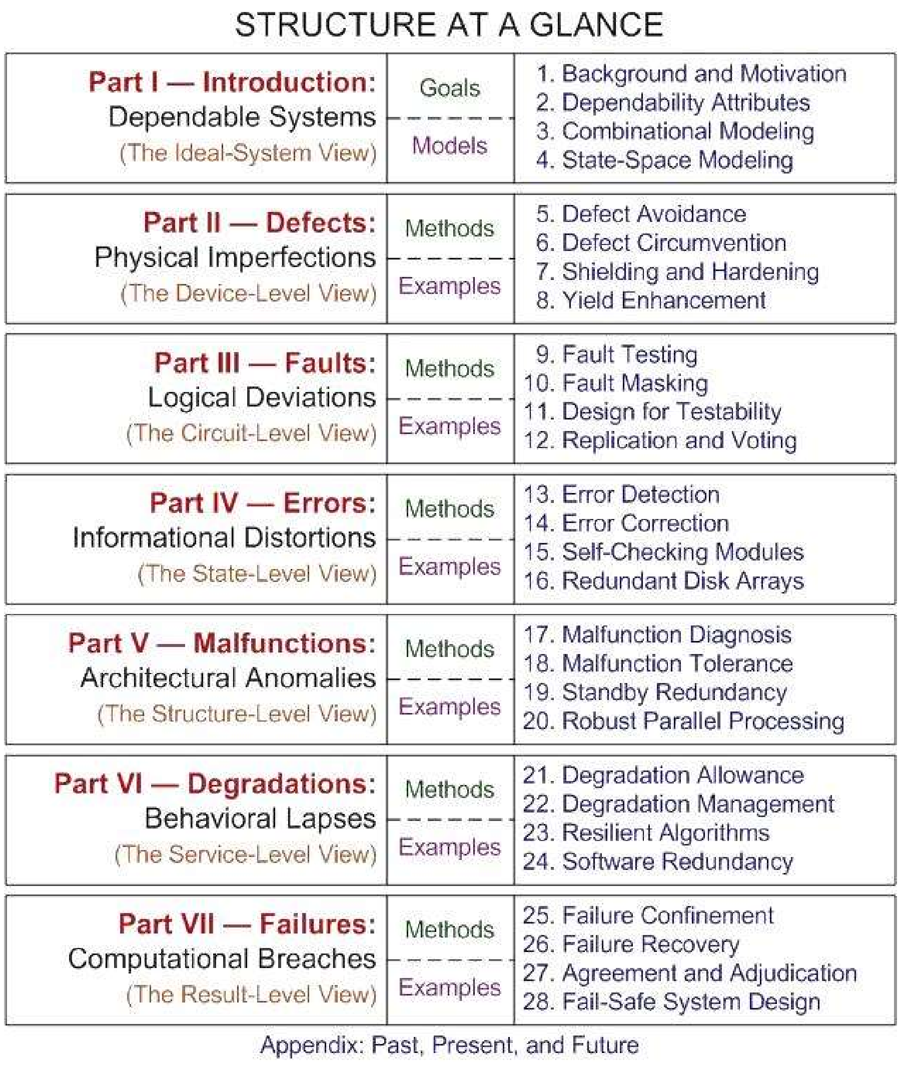
[TDSC] *IEEE Trans. Dependable and Secure Computing*, journals published by the IEEE Computer Society.

[Toy87] Toy, W. N., “Fault-Tolerant Computing,” *Advances in Computers*, Vol. 26, 1987, pp. 201-279.

[TRel] *IEEE Trans. Reliability*, quarterly journal published by IEEE Reliability Society.

Structure at a Glance

The multilevel model on the right of the following table is shown to emphasize its influence on the structure of this book; the model is explained in Chapter 1 (Section 1.4).



STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future

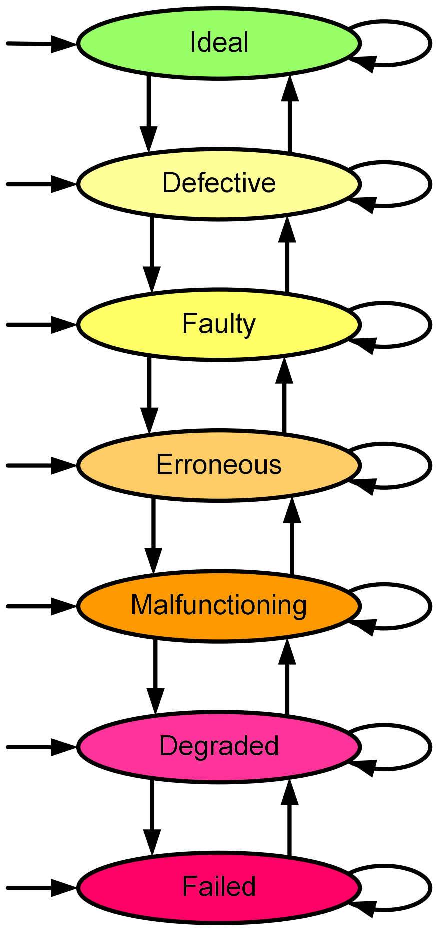
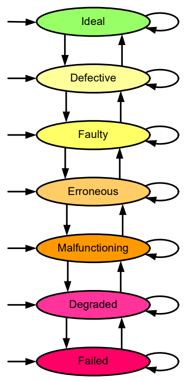


Table of Contents

I Introduction: Dependable Systems



"Once every decade an unloaded gun will fire; once every century a rake will fire."

Russian saying about rifles used on stage

"The severity with which a system fails is directly proportional to the intensity of the designer's belief that it cannot."

Anonymous

Chapters in This Part

Dependability concerns are integral parts of engineering design. Ideally, we would like our computer systems to be perfect, yielding timely and correct results in all cases. However, just as bridges collapse and airplanes crash occasionally (albeit rarely), so too computer hardware and software cannot be made totally immune to unpredictable or catastrophic behavior. Despite great strides in component reliability and programming methodology, the exponentially increasing complexity of integrated circuits and software components makes the design of prefect computer systems nearly impossible. In this part, after reviewing some application areas for which conventionally designed hardware and software do not offer the desired level of dependability, we discuss evaluation criteria and tools for dependable computer systems. Put another way, the four chapters of this part, listed above, answer the key questions of where we want to go and how we know whether we have gotten there.

1 Background and Motivation

"Failing to plan is planning to fail."

Effie Jones

". . . on October 5, 1960, the warning system at NORAD indicated that the United States was under massive attack by Soviet missiles with a certainty of 99.9 percent. It turned out that the BMEWS radar in Thule, Greenland, had spotted the *rising moon*. Nobody had thought about the moon when specifying how the system should act."

A. Borning, Computer System Reliability and Nuclear War

Topics in This Chapter

Learning how to do things right, without first developing an appreciation of why those things need to be done, consequences of not doing them, and underlying objectives, can be difficult. Dependable computing is no exception. In this chapter, we establish the need for dependable computing, introduce the basic terminology of the field, and review several application areas for which certain facets of computer system dependability are important, but where dependability requirements cannot be met with straightforward design. The terminology is presented in the framework of a multilevel model that facilitates the study of dependability, from the lowest levels of devices and circuits to the highest levels of service adequacy and computational outcomes.

1.1 The Need for Dependability

Computer and information systems are important components of the modern society, which has grown increasingly reliant on the availability and proper functioning of such systems. When a computer systems fails:

- A writer or reporter, who can no longer type on her personal computer, may become less productive, perhaps missing a deadline.
- A bank customer, unable to withdraw or deposit funds through a remote ATM, may become irate and perhaps suffer financial losses.
- A telephone company subscriber may miss personal or business calls or even suffer loss of life due to delayed emergency medical care.
- An airline pilot or passenger may experience delays and discomfort, perhaps even perish in a crash or midair collision.
- A space station or manned spacecraft may lose maneuverability or propulsion, wandering off in space and becoming irretrievably lost.

Thus, consequences of computer system failures can range from inconvenience to loss of life. Low-severity consequences, such as dissatisfaction and lost productivity, though not as important on a per-occurrence basis, are much more frequent, thus leading to nonnegligible aggregate effects on the society's economic well-being and quality of life. Serious injury or loss of life, due to the failure of a safety-critical system, is no doubt a cause for concern. As computers are used for demanding and critical applications by an ever expanding population of minimally trained users, the dependability of computer hardware and software becomes even more important.

But what is dependability? Webster's Ninth New Collegiate Dictionary defines "dependable" as capable of being trusted or relied upon; a synonym for "reliable." In the technical sense of the term, dependability is viewed as a qualitative system attribute that can be variously quantified by *reliability*, *availability*, *performability*, *safety*, and so on. Thus, the use of "dependability" to represent the qualitative sense of the term "reliability" allows us to restrict the application of the latter term to a precisely defined probabilistic measure of survival.

In one of the early proposals, dependability is defined succinctly as *"the probability that a system will be able to operate when needed"* [Hosf60]. This simplistic definition, which subsumes both of the well-known notions of reliability and availability, is only valid for systems with a single catastrophic failure mode; i.e., systems that are either completely operational or totally failed. The problem lies in the phrase "be able to operate." What we are actually interested in is "task accomplishment" rather than "system operation."

The following definition [Lapr82] is more suitable in this respect: *"... dependability [is defined] as the ability of a system to accomplish the tasks (or equivalently, to provide the service[s]) which are expected from it."* This definition does have its own weaknesses. For one thing, the common notion of "specified behavior" has been replaced by "expected behavior" so that possible specification slips are accommodated in addition to the usual design and implementation flaws. However, if our expectations are realistic and precise, they might be considered as simply another form of system specification (possibly a higher-level one). However, if we expect too much from a system, then this definition is an invitation to blame our misguided expectations on the system's undependability.

A more useful definition has been provided by Carter [Cart82]: *"... dependability may be defined as the trustworthiness and continuity of computer system service such that reliance can justifiably be placed on this service."* This definition has two positive aspects: It takes the time element into account explicitly ("continuity") and stresses the need for dependability validation ("justifiably"). Laprie's version of this definition [Lapr85] can be considered a step backwards in that it substitutes "quality" for "trustworthiness and continuity." The notions of "quality" and "quality assurance" are well-known in many engineering disciplines and their use in connection with computing is a welcome trend. However, precision need not be sacrificed for compatibility.

In the most recent compilation of dependable computing terminology evolving from the preceding efforts [Aviz04], dependability is defined as a system's "ability to deliver service that can justifiably be trusted" (original definition) and "ability to avoid service failures that are more frequent and more severe than is acceptable" (alternate definition), with trust defined as "accepted dependence." Both of these definitions are rather unhelpful and the first one appears to be circular.

To present a more useful definition of dependable computing, we must examine the various aspects of undependability. From a user's viewpoint, undependability takes the form of *late, incomplete, inaccurate,* or *incorrect* results or actions [Parh78]. The two notions of *trustworthiness* (correctness, accuracy) and *timeliness* can be abstracted from the above; completeness need not be considered separately, because any missing result or action can be deemed infinitely late. We also note that dependability should not be considered an intrinsic property of a computer system. Rather, it should be defined with respect to particular (classes of) computations and/or interactions. A system that is physically quite unreliable may well be made dependable for a particular class of interactions through the use of reasonableness checks and other algorithmic methods.

The preceding discussion leads us to the following [Parh94]: *Dependability of a computer or computer-based system may be defined as justifiable confidence that it will perform specified actions or deliver specified results in a trustworthy and timely manner*. Note that the preceding definition does not preclude the possibility of having different levels of importance for diverse classes of user interactions or varying levels of criticality for situations in which the computer is required to react. Such variations simply correspond to different levels of *confidence* and *dependability*. Our definition retains the positive elements of previous definitions, while presenting a result-level view of the time dimension by replacing the notion of "service continuity" by *timeliness* of actions or results.

Having defined computer system dependability, we next turn to the question of why we need to be concerned about it. This we will do through a sequence of viewpoints, or arguments. In these arguments, we use back-of-the-envelope calculations to illustrate the three classes of systems for which dependability requirements are impossible to meet without special provisions:

Long-life	=	Fail-slow	=	Rugged	=	High-reliability
Safety-critical	=	Fail-safe	=	Sound	=	High-integrity
Nonstop	=	Fail-soft	=	Robust	=	High-availability

a. The reliability argument: Assume that electronic digital systems fail at a rate of about $\lambda = 10^{-9}$ per transistor per hour. This failure rate may be higher or lower for different types of circuits, hardware technologies, and operating environments, but the same argument is applicable if we change λ. Given a constant per-transistor failure rate λ, an n-transistor digital system will have all of its components still working after t hours with probability $R(t) = e^{-n\lambda t}$. We will justify this exponential reliability equation in Chapter 2; for now, let's accept it on faith. For a fixed λ, as assumed here, $R(t)$ is a function of nt. Figure 1.1 shows the variation of $R(t)$ with nt. While it is the case that not every transistor failure translates into system failure, to be absolutely sure about correct system operation, we have to proceed with this highly pessimistic assumption. Now, from Fig. 1.1, we can draw some interesting conclusions.

- The on-board computer for a 10-year unmanned space mission to explore the solar system should be built out of only $O(10^3)$ transistors if the $O(10^5)$-hour mission is to have a 90% success probability.

- The computerized flight control system on board an aircraft cannot contain more than $O(10^4)$ transistors if it is to fail with a likelihood of less than 1 in 10,000 during a 10-hour intercontinental flight.

The need for special design methods becomes quite apparent if we note that modern microprocessors and digital signal processor (DSP) chips contain tens or hundreds of millions of transistors.

b. The safety argument: In a safety-critical system, the "cost" of failure can be quantified by the product of hazard probability and severity of its consequences. Based on the reliability argument above, a 1M-transistor flight control computer fails with probability 10^{-2} during a 10-hour flight. If 0.1% of all computer failures cause the airplane to crash, an airline that operates $O(10^3)$ planes, each of which has $O(10^2)$ such flights per year, will experience $O(1)$ computer-related crashes per year, on the average. If a crash kills $O(10^2)$ people and the cost per lost life to the airline is $O(10^7)$ dollars, then the airline can expect safety mishaps resulting from computer failures to cost it billions of dollars per year. Clearly, this is just one view of safety; other entities, such as transportation safety boards, passengers, and their families may quantify the consequences of a plane crash differently. However, the preceding argument is enough to show that safety requirements alone justify investing in special design and implementation methods to postpone, or reduce the chances of, computer system failures.

c. The availability argument: A central telephone switching facility should not be down for more than a few minutes per year; more outages would be undesirable not only because they lead to customer dissatisfaction, but also owing to the potential loss of revenues. If the diagnosis, and replacement, of subsystems that are known to be malfunctioning takes 20-30 minutes, say, a mean time between failures (MTBF) of $O(10^5)$ hours is required. A system with reliability $R(t) = e^{-n\lambda t}$, as assumed in our earlier reliability argument, has an MTBF of $1/(n\lambda)$. Again, we accept this without proof for now. With $\lambda = 10^{-9}$/transistor/hour, the telephone switching facility cannot contain more than $O(10^4)$ transistors if the stated availability requirement is to be met; this is several orders of magnitude below the complexity of modern telephone switching centers.

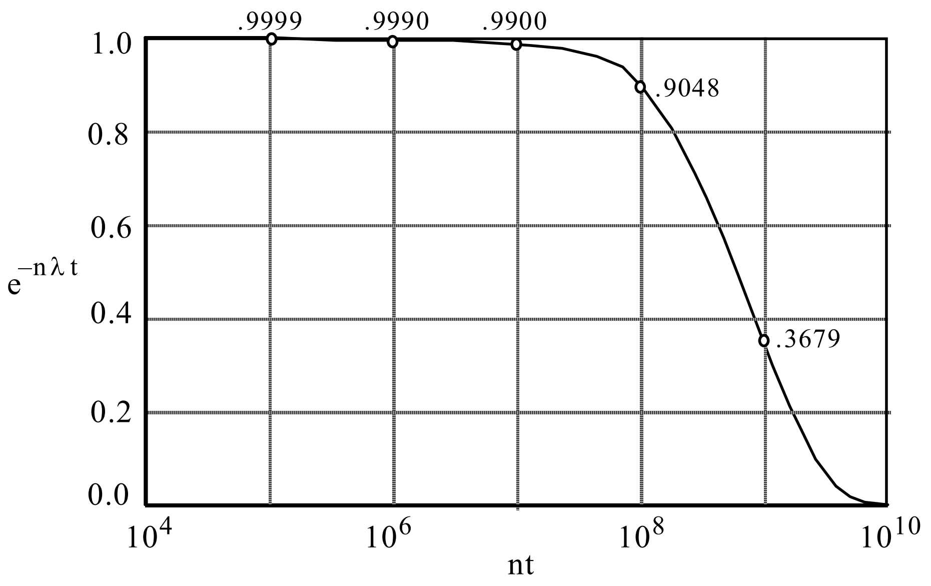


Fig. 1.1 **Reliability of an *n*-transistor system after *t* hours as a function of *nt* for $\lambda = 10^{-9}$/transistor/hour.**

1.2 A Motivating Case Study

One of the important problems facing the designers of distributed computing systems is ensuring data availability and integrity. Consider, for example, a system with five sites (processors) that are interconnected pairwise via dedicated links, as depicted in Fig. 1.2. The detailed specifications of the five sites, S_0-S_4, and 10 links, L_0-L_9, are not important for this case study; what is important, is that sites and links can malfunction, making the data stored at one site inaccessible to one or more other sites. If data availability to all sites is critical, then everything must be stored in redundant form.

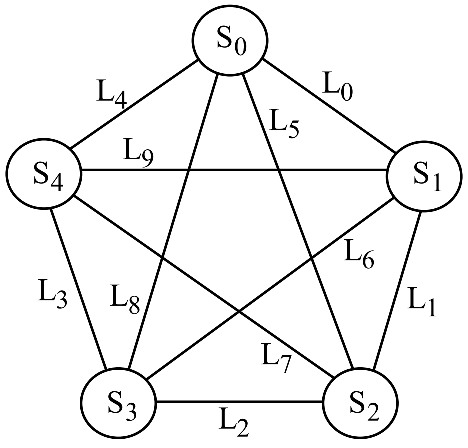


Fig. 1.2 Five-site distributed computer system with a dedicated direct link connecting each pair of sites.

Let the probability that a particular site (link) is available and functions properly during any given attempt at data access be a_S (a_L) and assume that the sites and links malfunction independently of each other.

Example 1.1: Home site and mirror site Quantify the improvement in data availability when each data file F_i has a home or primary site $H(F_i)$, with a copy stored at a mirror site $M(F_i)$. This doubles the aggregate storage requirements for the file system, but allows access to data despite faulty sites and links.

Solution: To quantify the accessibility of a particular file, we note that a site can obtain a copy of F_i if the home site $H(F_i)$ and the link leading to it have not failed OR the home site is inaccessible, but the mirror site $M(F_i)$ can be accessed. Thus, the availability (measure of accessibility) $A(F_i)$ for a mirrored file F_i is:

$$A(F_i) = a_S a_L + (1 - a_S a_L) a_S a_L = 2 a_S a_L - (a_S a_L)^2$$

In deriving the preceding equation, we have assumed that the file must be accessed directly from one of the two sites that holds a copy. Analyzing the problem when indirect accesses via other sites are also allowed is left as an exercise. As a numerical example, for $a_S = 0.99$ and $a_L = 0.95$, we have:

$$A(F_i) = 0.99 \times 0.95 \times (2 - 0.99 \times 0.95) \approx 0.9965$$

In other words, data unavailability has been reduced from 5.95% in the nonredundant case to 0.35% in the mirrored case.

Example 1.2: File triplication Suppose that the no-access probability of Example 1.1 is still too high. Evaluate, and quantify the availability impact of, a scheme where three copies of each data file are kept.

Solution: In this case, the availability of a file F_i becomes:

$$\begin{aligned} A(F_i) &= a_S a_L + (1 - a_S a_L) a_S a_L + (1 - a_S a_L)^2 a_S a_L \\ &= 3 a_S a_L - 3(a_S a_L)^2 + (a_S a_L)^3 \end{aligned}$$

Now, with $a_S = 0.99$ and $a_L = 0.95$, we have:

$$A(F_i) = 0.99 \times 0.95 \times [3 - 3 \times 0.99 \times 0.95 + (0.99 \times 0.95)^2] \approx 0.9998$$

Data unavailability is thus reduced to 0.02%. This improvement in data availability is achieved at the cost of increasing the aggregate storage requirements by a factor of 3, compared with the nonredundant case.

Example 1.3: File dispersion Let us now devise a more elaborate scheme that requires lower redundancy. Each file F_i is viewed as a bit string of length l. It is possible to encode such a file into approximately $5l/3$ bits (a redundancy of 67%, as opposed to 100% and 200% in Examples 1.1 and 1.2, respectively) so that if the resulting bit string of length $5l/3$ is divided equally into five pieces, any three of these $(l/3)$-bit pieces can be used to reconstruct the original l-bit file. Let us accept for now that the preceding data dispersion scheme is actually workable and assume that for each file, we store one of the five pieces thus obtained in a different site in Fig. 1.2. Now, any site needing access to a particular data file already has one of the three required pieces and can reconstruct the file if it obtains two other pieces from the remaining four sites. Quantify the data availability in this case.

Solution: In this case, a file F_i is available if two or more of four sites are accessible:

$$\begin{aligned} A(F_i) &= (a_S a_L)^4 + 4(1 - a_S a_L)(a_S a_L)^3 + 6(1 - a_S a_L)^2(a_S a_L)^2 \\ &= 6(a_S a_L)^2 - 8(a_S a_L)^3 + 3(a_S a_L)^4 \end{aligned}$$

Again, we have assumed that a file fragment must be accessed directly from the site that holds it. With $a_S = 0.99$ and $a_L = 0.95$, we have:

$$A(F_i) = (0.99 \times 0.95)^2 \times [6 - 8 \times 0.99 \times 0.95 + 3(0.99 \times 0.95)^2] \approx 0.9992$$

Data unavailability is thus 0.08% in this case. This result is much better than that of Example 1.1 and is achieved at a lower redundancy as well. It also performs only slightly worse than the triplication method of Example 1.2, which has considerably greater storage overhead.

In Examples 1.1-1.3, we ignored several important considerations such as how redundant data are kept consistent, how malfunctioning sites/links are identified, how recovery is accomplished when a malfunctioning site that has been repaired is to return to service, and, finally, how data corrupted by the actions of an adversary (rather than by a site or link malfunction) might be detected. However, the examples do point to the diversity of methods for coping with dependability problems and to the sophistication required for devising efficient or cost-effective schemes.

1.3 Impairments to Dependability

Impairment to dependability are variously described as *hazards, flaws, defects, faults, errors, malfunctions, failures,* and *crashes*. There are no universally agreed upon definitions for these terms, causing different and sometimes conflicting usage. Attempts at presenting precise definitions for the terms and standardizing their use have not been very successful. In the following paragraphs, we review two key proposals on how to view and describe impairments to dependability.

Members of the Newcastle Reliability Project, led by Professor Brian Randell, have advocated a hierarchic view [Ande82]: A (computer) system is a set of *components* (themselves systems) which interact according to a *design* (another system). The recursion stops when we arrive at atomic systems whose internal structures are of no interest at the particular level of detail with which we are concerned. System *failure* is defined as deviation of its behavior from that predicted (required) by the system's authoritative specification. Such a behavioral deviation results from an erroneous system state. An *error* is a part of an erroneous state which constitutes a difference from a valid state. The cause of the invalid state transition which first establishes an erroneous state, is a *fault* in one of the system's components or in the system's design. Similarly, the component's or design's failure can be attributed to an erroneous state within the corresponding (sub)system resulting from a component or design fault, and so on. Therefore, at each level of the hierarchy, "*the manifestation of a fault will produce errors in the state of the system, which could lead to a failure*" (Fig. 1.3).

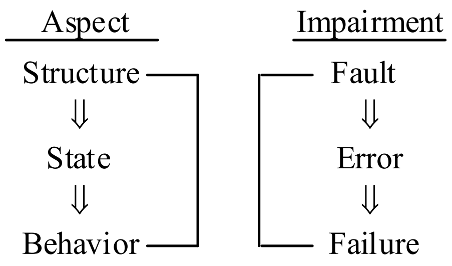


Fig. 1.3 Schematic diagram of the Newcastle hierarchical model and impairments within one level (the fault-error-failure cycle).

With the preceding model, *failure* and *fault* are simply different views of the same phenomenon. This is quite elegant and enlightening but introduces problems by the need for continual establishment of frames of reference when discussing the causes (faults) and effects (failures) of deviant system behavior at various levels of abstraction. While it is true that a computer system may be viewed at many different levels of abstraction, it is also true that some of these levels have proved more useful in practice. Avižienis [Aviz82] takes four of these levels and proposes the use of distinct terminology for impairments to dependability ("*undesired events*," in his words) at each of these levels. His proposal can be summarized in the cause-effect diagram of Fig. 1.4.

$$\begin{tikzcd}
\underline{\text{Universe}} & \underline{\text{Impairment}} \\
\text{Physical} \arrow[d, Rightarrow] & \text{Failure} \arrow[d, Rightarrow] \\
\text{Logical} \arrow[d, Rightarrow] & \text{Fault} \arrow[d, Rightarrow] \\
\text{Informational} \arrow[d, Rightarrow] & \text{Error} \arrow[d, Rightarrow] \\
\text{External} & \text{Crash}
\end{tikzcd}$$

Fig. 1.4 **Cause-effect diagram for Avižienis' four-universe model of impairments to dependability.**

There are a few problems with the preceding choices of names for undesired events. The term "failure" has traditionally been used both at the lowest and the highest levels of abstraction; viz, *failure rate, failure mode,* and *failure mechanism* used by electrical engineers and device physicists alongside *system failure, fail-soft operation,* and *fail-safe system* coming from computer architects. To comply with the philosophy of distinct naming for different levels, Avižienis retains "failure" at the physical level and uses "crash" at the other end. However, this latter term is unsuitable. Inaccuracies or delays, beyond what is expected according to system specifications, can hardly be considered "crashes" in the ordinary sense of the term.

Furthermore, the term "crash" puts the emphasis on system operation rather than task accomplishment and is thus unsuitable for fail-soft computer systems (such systems are like airplanes in that they *fail* much more often than they *crash*!). We are thus led to a definition of failure for computer systems that parallels that used by forensic experts in structural engineering [Carp96]: *Failure is an unacceptable difference between expected and observed performance.*

Another problem is that there are actually three external views of a computer system. The maintainer's external view consists of a set of interacting subsystems that must be monitored for detecting possible malfunctions in order to reconfigure the system or, alternatively, to guard against dangerous consequences (such as total system crash). The operator's external view, which is more abstract than the maintainer's system-level view, consists of a black box capable of providing certain computational services. Finally, the end user's external view is shaped by the system's reaction to particular situations or requests.

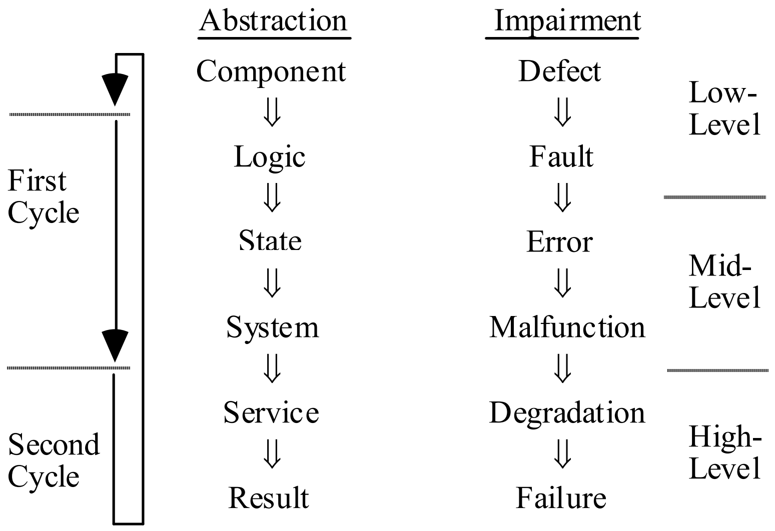


Fig. 1.5 **Cause-effect diagram for an extended six-level view of impairments to dependability.**

Figure 1.5 depicts this extended model. There are two ways in which our extended six-level impairment model can be viewed. The first view, shown on the left edge of Fig. 1.5, is to consider it an unrolled version of the model in Fig. 1.3. This unrolling allows us to talk about two cycles simultaneously without a danger of being verbose or ambiguous. A natural question, then, is why stop at one unrolling. The reason is quite pragmatic. We are looking at dependability from the eyes of the system architect who deals primarily with module interactions and performance parameters, but also needs to be mindful of the circuit level (perhaps even down to switches and transistors) in order to optimize/balance the system from complex angles involving speed, size, power consumption, upward/downward scalability, and so on.

The second view, which is the one that we will use in the rest of this book, is that shown in the right half of Fig. 1.5:

- Low-level impairments, affecting the devices/components or circuit functions, are within the realm of hardware engineers and logic designers.

- Mid-level impairments span the circuit-system interface, which includes the register-transfer level and the processor-switch-memory level.

- High-level impairments, affecting the system as a whole, are of interest not only to system architects but also to system integrators.

Taking into account the fact that a nonatomic component is itself a system, usage of the term “failure” in *failure rate, failure mode,* and *failure mechanism* could be justified by noting that a component is its designer’s end product (system). Therefore, we can be consistent by associating the term “failure” with the highest and the term “defect” with the lowest level of abstraction. The component designer’s *failed system* is the system architect’s *defective component*. However, to maintain a consistent point of view throughout the book, we will use the somewhat unfamiliar component-level terms *defect rate*, *defect mode*, and *defect mechanism* from now on.

One final point: Computer systems are composed of hardware and software elements. Thus, the reader may be puzzled by a lack of specific mention of impairments to software dependability in our discussions thus far. The reason for this is quite practical. Whereas one can meaningfully talk about defects, faults, errors, malfunctions, degradations, and failures for software, it is the author’s experience that formulating statements, or describing design methods, that apply to both hardware and software, requires some stretching that makes the results somewhat convoluted and obscure. Perhaps, this would not be the case if the author possessed greater expertise in software dependability.

Nevertheless, for the sake of completeness, we will discuss a number of algorithm and software design topics in the later parts of the book, with the hope that some day the discussion of dependable hardware and software can be truly integrated.

Anecdote: The development of the six-level model of Fig. 1.5 began in 1986, when the author was a Visiting Professor at the University of Waterloo in Canada. Having six levels is somewhat unsatisfactory, since successful models tend to have seven levels. However, despite great effort expended in those days, the author was unable to add a seventh type of impairment to the model. Undeterred by this setback, the author devised the seven states of a system shown in Fig. 1.6.

1.4 A Multilevel Model

The field of dependable computing deals with the procurement, forecasting, and validation of computer system dependability. According to our discussions in Section 1.3, impairments to dependability can be viewed from six abstraction levels. Thus, subfields of dependable computing can be thought of as dealing with some aspects of one or more of these levels. Specifically, we take the view that a computer system can be in one of seven states: *Ideal*, *defective*, *faulty*, *erroneous*, *malfunctioning*, *degraded*, or *failed*, as depicted in Fig. 1.6. Note that these states have nothing to do with whether or not the system is "running." A system may be running even in the failed state; the fact that it is failed simply means that it isn't delivering what is expected of it.

Upon the completion of its design and implementation, a system may end up in any of the seven states, depending on the appropriateness and thoroughness of validation efforts. Once in the initial state, the system moves from one state to another as a result of *deviations* and *remedies*. Deviations are events that take the system to a lower (less desirable) state, while remedies are techniques or measures that enable a system to make the transition to a higher state.

The observability of the system state (ease of external recognition that the system is in a particular state) increases as we move downward in Fig 1.6. For example, the inference that a system is "ideal" can only be made through formal proof techniques; a proposition that is currently impossible for complex computer systems. At the other extreme, a failed system can usually be recognized with little or no effort. As examples of intermediate states, the "faulty" state is recognizable by extensive off-line testing, while the "malfunctioning" state is observable by on-line monitoring with moderate effort. It is, therefore, common practice to force a system into a lower state (e.g., from "defective" to "faulty," under *torture testing*) in order to deduce its initial state.

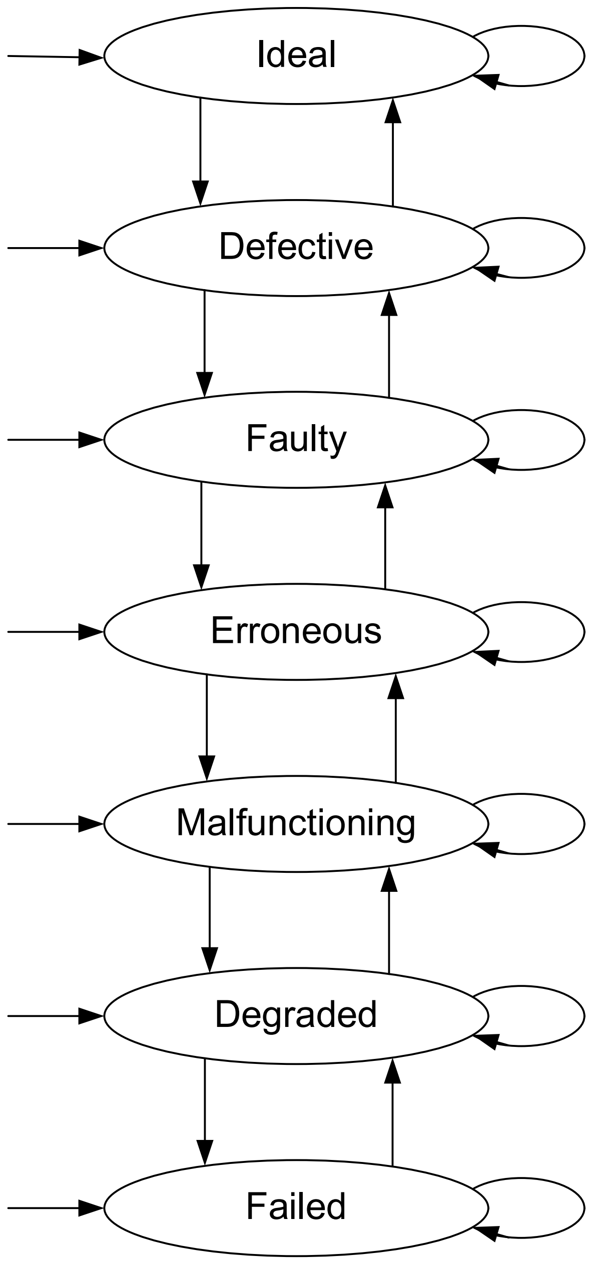


Fig. 1.6 **System states and state transitions in the multilevel model of dependable computing. Horizontal arrows on the left denote entry points. Downward (upward) transitions represent deviations (remedies). Self-loops model tolerance.**

One can associate five attributes with each of the transitions in Fig. 1.6. These attributes are:

(Natural) Cause of the transition
(Natural) Hindrance factors
Facilitation schemes
Avoidance techniques
Modeling methods and tools

For defect-induced failures, the sequence of transitions from defect to failure may be quite slow, owing to large interlevel hindrances or latencies, or so quick as to defy detection. Natural interlevel latencies can be increased through tolerance provisions or reduced for making the deviations more readily observable (because deviations near the bottom of Fig. 1.6 are more easily detected). The former methods, are referred to as defect tolerance, fault tolerance, error tolerance, malfunction tolerance, degradation tolerance, and failure tolerance, while the latter are useful for defect testing, fault testing, error testing, and so on.

We will discuss deviations and remedies in considerable detail in the rest of this book. Here, we present just a few samples of how subfields of dependable computing can be characterized according to their relevance to one or more of these transitions and their attributes.

Fault injection:	cause[$\rightarrow$ faulty]
Fault testing:	facilitation[faulty $\rightarrow$ erroneous]
Defect tolerance:	avoidance[defective $\rightarrow$ faulty]
Error modeling:	modeling[$\rightarrow$ erroneous] + modeling[faulty $\rightarrow$ erroneous]

Thus, a byproduct of the preceding characterization might be an accurate indexing scheme for reference sources in the field of dependable computing; viz, a technical paper may be indexed by listing attributes or relevant transitions as above.

Possible causes for the sideways transitions in Fig. 1.6 include *specification slips* and *implementation mistakes* (including the use of wrong or unfit building blocks). A deviations may be caused by *wearout*, *overloading*, or *external disturbance* and is additionally characterized by its *duration* as being *permanent*, *intermittent*, or *transient*.

Note, however, that even transient deviations can have permanent consequences; for example, a fault-induced error may persist after the fault itself has vanished.

We can also classify deviations by their *extent* (*local* and *global* or *catastrophic*) and *consistency* (*determinate* and *indeterminate* or *Byzantine*). A local deviation can quickly develop into a catastrophic one if not promptly detected by *monitors* and isolated by means of *firewall*s. Transient and indeterminate deviations are notoriously difficult to handle. To see why, imagine feeling ill, but all the symptoms of your illness disappearing each time you visit a doctor. Worse yet, imagine the symptoms changing as you go from one doctor to another for obtaining a second opinion.

1.5 Examples and Analogies

Let us use some familiar every-day phenomena to accentuate the states and state transitions shown in Fig. 1.6 [Parh97]. These examples also show the relevance of such a multilevel model in a wider context. Example 1.5 is similar to Example 1.4, but it illustrates the lateral transitions of Fig. 1.6 and multilevel tolerance methods better. Example 1.6 incorporates both tolerance and avoidance techniques.

Example 1.4: Automobile brake system Relate an automobile brake system to the multilevel model of Fig. 1.6.

Solution: An automobile brake system with a weak joint in the brake fluid piping (e.g., caused by a design flaw or a road hazard) is *defective*. If the weak joint breaks down, the brake system becomes *faulty*. A careful (off-line) inspection of the automobile can reveal the fault. However, the driver does not automatically notice the fault (on-line) while driving. The brake system state becomes *erroneous* when the brake fluid level drops dangerously low. Again, the error is not automatically noticed by the driver, unless a working brake fluid indicator light is present. A *malfunctioning* break system results from the improper state of its hydraulics when the brake pedal is applied. With no brake fluid indicator light, the driver's first realization that something is wrong comes from noticing the *degraded* performance of the brake system (higher force needed or lower deceleration). If this degraded performance is insufficient for slowing down or stopping the vehicle when needed, the brake system has *failed* to act properly or deliver the expected result.

Example 1.5: Automobile tire Relate the functioning of an automobile tire to the multilevel model of Fig. 1.6.

Solution: Take an automobile with one tire having a weak spot on its road surface. The *defect* may be a result of corrosion or due to improper manufacture and inspection. Use of multiple layers or steel reinforcement constitute possible defect tolerance techniques. A hole in the tire is a *fault*. It may result from the defect or caused directly by a nail. Low tire pressure due to the hole, or directly as a result of improper initialization, is viewed as an *error*. Automatic steering compensation leads to error tolerance (at least for a while). A tire that is unfit for use, either due to its pressure dropping below a threshold or because it was unfit to begin with (e.g., too small), leads to a *malfunction*. A vehicle with multiple axles or twin tires can tolerate some tire malfunctions. In the absence of tolerance provisions, one can still drive an automobile having a flat or otherwise unfit tire, but the performance (speed, comfort, safety, etc.) will be seriously *degraded*. Even a vehicle with three or more axles suffers performance degradation in terms of load capacity when a tire malfunctions. Finally, as a result of the preceding sequence of events, or because someone forgot to install a vital subsystem, the entire automobile system may *fail*.

Example 1.6: Organizational decision-making Relate the decision-making processes in a small organization (e.g., those related to staff promotions) to the multilevel model of Fig. 1.6.

Solution: *Defects* in the organization's staff promotions policies may cause improper promotions, viewed as *faults*. The ensuing ineptitudes and dissatisfactions are *errors* in the organization's state. The organization's personnel or departments probably start to *malfunction* as a result of the errors, in turn causing an overall *degradation* of performance. The end result may be the organization's *failure* to achieve its goals. Many parallels exist between organizational procedures and dependable computing terms such as *defect removal* (external reviews), *fault testing* (staff evaluations), *fault tolerance* (friendly relations, teamwork), *error correction* (openness, alternate rewards), and *self-repair* (mediation, on-the-job training).

Example 1.7: Leak and drainage analogies Discuss the similarities of avoidance and tolerance methods in the model of Fig. 1.6 to those of stopping leaks in a water system and using drainage to prevent leaks from causing extensive damage.

Solution: Figure 1.7 depicts a system of six concentric water reservoirs. Pouring water from above corresponds to defects, faults, and other impairments, depending on the layer(s) being affected. These impairments can be avoided by controlling the flow of water through valves or tolerated through the provision of drains of acceptable capacities for the reservoirs. The system fails if water ever gets to the outermost reservoir. This may happen, for example, by a broken valve at some layer combined with inadequate drainage at the same and all outer layers. Wall heights between adjacent reservoirs correspond to natural interlevel latencies in the multilevel model of Fig. 1.6. Water overflowing from the outermost reservoir into the surrounding area is the analog of a computer failure adversely affecting the larger physical, corporate, or societal system.

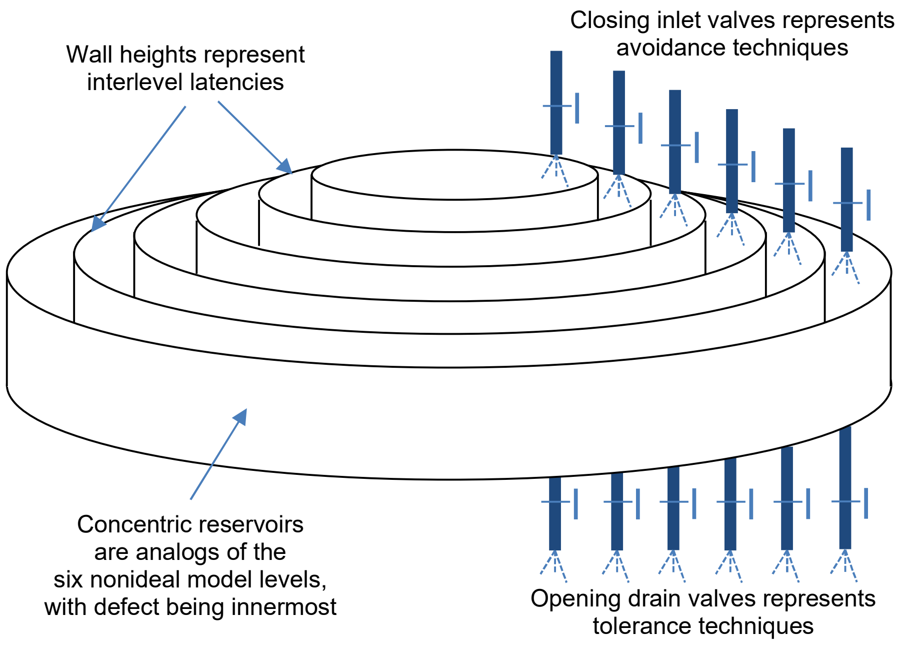

Closing inlet valves represents avoidance techniques
Wall heights represent interlevel latencies
Concentric reservoirs are analogs of the six nonideal model levels, with defect being innermost
Opening drain valves represents tolerance techniques

Fig. 1.7 **An analogy for our multilevel model of dependable computing. Defects, faults, errors, malfunctions, degradations, and failures are represented by pouring water from above. Valves represent avoidance and tolerance techniques. The goal is to avoid overflow.**

1.6 Dependable Computer Systems

We will describe a number of dependable computer systems after learning about the methods used in their design and implementation. However, it is desirable to have a brief preview of the types of system that can benefit from these methods in way of motivation. As mentioned in Section 1.2, three main categories of dependable computer systems can be distinguished. These are reviewed in the following paragraphs. Most modern general-purpose computers also include dependability enhancement features, but these are often toned-down versions of the methods used in the following system classes to make them implementable within more stringent cost constraints. For examples of dependability enhancement methods found in general-purpose computers, see [Siew92], pp. 427-523.

Long-life systems: Long life computer systems are needed in application domains where maintenance is impossible or costly. In the first category, we have computers on board spacecraft, particularly those for deep space planetary probes. In a multiyear space probe, for example, it is imperative that the computer be still fully functional near the end of the mission, when the greatest payoffs in terms of discoveries and data collection are expected. The second category includes machines installed in remote, treacherous, or hostile environments that cannot be reached easily or safely. Requirements in the design of such systems are similar, except for cost constraints, which are likely to be less stringent for space applications. Take the case of the Galileo Orbiter which, with help from a separate Probe module, collected data on Jupiter. The Galileo architecture was based on the earlier Voyager system that explored the outer planets of the solar system, with a main difference that the Galileo design used microprocessors extensively (a total of 27 in the computing and instrumentation systems). Module replication, error coding, activity monitoring, and shielding were some of the many diverse methods used to ensure a long life.

Safety-critical systems: The failure of flight control computers on commercial aircraft, safety monitors in nuclear power plants, and process control systems in chemical plants can have dire consequences. Early designs for safety-critical applications included Carnegie-Mellon University’s C.vmp (voted multiprocessor) and Stanford University’s SIFT (software-implemented fault tolerance) systems. Both designs were based on the notion of voting on the results of multiple independent computations. Such an approach appears inevitable in view of the need for quick recovery from malfunctions in a real-time system with hard deadlines. The C.vmp system used three microprocessors operating in lock step, with a hardware voter connected to the system bus performing bit-by-bit voting to prevent the propagation of erroneous information to the memory. In SIFT, the off-the-shelf computing elements were connected via custom-designed buses and bus interfaces. Each computing element had an executive, along with software voting procedures that could mask out an erroneous result received from a malfunctioning unit. A task requiring data from another task got them from units that executed the task (typically by 2-out-of-3 voting), with the actual multiplicity of a task being a function of its criticality. Subsequently, August Systems chose the SIFT philosophy for its dependable computers aimed at the industrial control market.

High-availability systems: Research and development in high-availability computers was pioneered by AT&T in the early 1960s in order to meet the stringent availability goals for its electronic switching systems (ESS). In the early versions of the system, two processors ran in tight synchronism, with multiple hardware comparators matching the contents of various registers and buffers. A mismatch caused an interrupt to occur and diagnostics to be run for identifying the malfunctioning unit and eventually the faulty circuit pack. ESS processors have gone through many design modifications over the years. Some of these were necessitated by the changing technology while others resulted from field experience with previous designs. A second major application area for high-availability computers is in on-line transaction processing systems of the types found in banking and e-commerce. Although several sophisticated users of this type had custom-designed highly dependable systems as far back as the mid 1960s, commercial marketing of ready-made systems for such applications was started by Tandem Computers in the late 1970s. Tandem's early machines, named "NonStop", and their successors were designed with the aim of preventing any single hardware or software malfunction from disabling the system. This objective was achieved by hardware and informational redundancy as well as procedural safeguards, backup processes, consistency checks, and recovery schemes in a distributed-memory multiprocessor, with redundant buses connecting its various subsystems.

Problems

1.1 High-performability systems

To the classes of high-reliability, high-safety, and high-availability systems, characterized in Section 1.1, one might add the class of high-performability systems. Suggest terms similar to those listed for the preceding classes (e.g., nonstop = fail-soft = robust = high-availability) to characterize this new class. Justify your choices.

1.2 The many facets of dependability

With reference to the three classes of fail-slow, fail-safe, and fail-soft systems, briefly characterized in Section 1.1, discuss what each of the following terms might mean and how the resulting classes of dependable systems are related to the above:

a. Fail-stop

b. Fail-hard

c. Fail-fast

d. Fool-proof

e. Tamper-proof

f. Resilient

g. Secure

1.3 The need for dependability

Suppose that the system failure rate does not grow linearly with the number n of transistors but is rather a sublinear function of n. Pick a suitable function for the failure rate of an n-transistor system, draw the corresponding reliability curve on Fig. 1.1, and discuss the resulting changes in the three arguments presented at the end of Section 1.1.

1.4 Data availability in distributed systems

With reference to our discussion of data availability in Section 1.2:

a. Write an expression for the accessibility measure $A(\mathrm{F}_i)$ of Example 1.1 as a function of $u = 1 - a_\mathrm{S}a_\mathrm{L}$ and provide intuitive justification for the result.

b. Repeat part a for Example 1.2.

1.5 Data availability in distributed systems

The keen reader may have observed that in the Examples 1.1-1.3 of Sections 1.2, links are much less reliable than sites. Suppose that we connect each pair of sites by two different links, each one allowing access independently with probability $a_\mathrm{L} = 0.95$. Redo Examples 1.1-1.3, calculating the numerical accessibility measure in each case. Compare the results with those of the original examples and discuss.

1.6 Data availability in a ring network

Redo examples 1.1, 1.2, and 1.3, assuming that the 5 sites are interconnected as a bidirectional ring network, rather than a complete network. Note that with ring connectivity, it no longer makes sense to assume that access to the data is allowed only via a direct link. When data is accessed indirectly, all nodes and links on the indirect path must be functional for access to be successful. In your analysis, consider the worst case regarding where data copies are located relative to the user site.

1.7 Data availability in distributed systems

With reference to our discussion of data availability in Section 1.2:

a. Consider Examples 1.1-1.3 and assume that there are six, rather than five, sites in the distributed system. Thus, in Example 1.3, an l-bit data file is encoded into six $(l/3)$-bit pieces, any three of which can be used to reconstruct the original file. The data redundancy of 100% for this 3-out-of-6 scheme is identical to that of mirroring in Example 1.1. Redo Examples 1.1-1.3 and compare the resulting data access probabilities.

b. In general, with $2k$ sites and encoding of files such that any k of their $2k$ pieces, stored one per site, can be used for reconstruction, a data redundancy identical to mirroring, but with greater accessibility, is obtained. Plot the data access probabilities for mirroring and the preceding k-out-of-$2k$ scheme in a $2k$-site distributed system, for $1 \le k \le 8$, and discuss the results.

1.8 Data availability in distributed systems

With reference to our discussion of data availability in Section 1.2:

a. Quantify the probability of being able to access a data file F_i in Example 1.1 if, whenever direct access is impossible due to link malfunctions, the requesting site can obtain a copy of the desired file indirectly via the other two sites, if they have access to the required information.

b. Repeat part a for Example 1.2.

c. Repeat part a for Example 1.3.

1.9 Impairments to dependability

With reference to the two-cycle interpretation shown on the left edge of Fig. 1.5, consider two other unrollings, one beyond (preceding) the component level and another beyond (following) the result level. Briefly state how one might characterize the abstraction levels and impairments for the resulting cycles.

1.10 Multilevel model of dependable computing

Relate the following notions to the transitions in Fig. 1.6 and their attributes in a manner similar to the examples provided near the end of Section 1.4.

a. Preventive maintenance

b. Design for testability

c. Fault modeling

d. Error-detecting codes

e. Error-correcting codes

f. Self-repairing systems

g. Yield improvement for integrated circuits

h. Checkpointing

1.11 Multilevel model of dependable computing

a. Consider the floating-point division flaw in early Pentium processors. Place this flaw and its consequences and remedies in the context of the model in Fig. 1.6.

b. Repeat part a for the Millennium bug, aka the Y2K (year-2000) problem, which would have caused some programs using dates with 2-digit year fields to fail when the year turned from 1999 to 2000 if the problem were not fixed in time.

c. Repeat part a for a third widespread hardware or software flaw that you identify.

1.12 Multilevel model of dependable computing

Relate the following situations to the analogy in Fig. 1.7.

a. A parallel computer system is designed to tolerate up to two malfunctioning processors. When a malfunction is diagnosed, the integrated-circuit card that holds the malfunctioning processor is isolated, removed, and replaced with a good one, all in about 15 minutes.

b. When the Pentium division flaw was uncovered, a software package was modified through "patches" to avoid operand values for which the flaw would lead to errors. Upon replacing the Pentium chip with a redesigned chip (without the flaw), the patched software was not modified.

1.13 Dependable computer systems

Demonstrate the orthogonality of the three attributes of long life, high availability, and safety by discussing how:

a. Long-life systems are not necessarily highly available or safe.

b. Highly available systems could be short-lived and/or unsafe.

c. Safety-critical systems may be neither long-life nor nonstop.

1.14 The human immune system

Avizienis [Aviz04] has suggested the human immune system as a suitable model for developing a hardware-based infrastructure for fault tolerance. Discuss his proposal in a couple of paragraphs and relate it to the multilevel model of Fig. 1.6.

1.15 The risks of poorly designed user interfaces

Olsen [Olse08] has described an incident in which an on-line banking customer lost $100,000 due to a simple keying error, which (the customer alleges) should have been caught by the system's user interface. Study the article and write a one-page report about the nature of the error and why poorly designed user interfaces constitute major risk factors in computer-based systems.

1.16 The Excel 2007 design flaw

According to news stories published in the last week of September 2007, the then newest version of Microsoft Excel contained a flaw that led to incorrect values in rare cases. For example, when multiplying 77.1 by 850, 10.2 by 6,425 or 20.4 by 3,212.5, the number 100,000 was displayed instead of the correct result 65,535. Similar errors were observed for calculations that produced results close to 65,536. Study this problem using Internet sources and discuss in *one typed page* (single spacing is okay) the nature of the flaw, why it went undetected, exactly what caused the errors, and how Microsoft dealt with the problem.

1.17 The USS Yorktown incident

USS Yorktown, one of the first "smart ships" deployed by the US Navy, was the subject of this 1998 news headline: "Software Glitches Leave Navy Smart Ship Dead in the Water." Write a 2-page investigative report about the incident, what caused it, and actions taken to prevent similar failures.

1.18 Overheating batteries in airliners

In mid-January 2013, the world's entire fleet of Boeing-787 ("Dreamliner") aircraft was grounded while the lithium-ion battery systems, believed to have caused multiple incidents of fire on board the aircraft, were investigated. In late January, it was determined that overcharging of the batteries led to the overheating and resulting fires. Boing insisted that such battery overcharges were highly unlikely, given that multiple systems are in place to prevent them from happening. Investigate the problem and write a 2-page report of the incidents and the associated corrective actions, focusing on the multiple prevention mechanisms claimed by Boeing and why they did not stop the overheating.

1.19 Reliability and robustness vs. efficiency

Read the viewpoint [Ackl13] explaining why undue emphasis on efficiency undermines reliability and robustness/resilience. Moshe Vardi also discusses the problem in a lecture (starting at the 7:45 mark of the following YouTube video), entitled "Lessons from COVID-19: Efficiency vs. Resilience": https://www.youtube.com/watch?v=SjGXbIosMQA

a. Write a 200-word abstract for [Ackl13], summarizing its main points and argument.

b. Repeat part a for Moshe Vardi's lecture.

1.20 Sharing a secret

Eleven scientists collaborate on a project and they want to store project documents in a safe that can be opened only when at least 6 of the 11 are present. Show that if this is to be done using ordinary locks and keys, the minimal solution requires that the safe be equipped with 462 locks and each scientist provided with 252 keys. [*Hint:* The numbers 462 and 252 equal $\binom{11}{5}$ and $\binom{10}{5}$, respectively, where $\binom{n}{m}$ is the number of ways you can choose m items from among n.]

1.21 Simple, avoidable design errors

Computer maker Lenovo recalled more than 0.5 million laptop power cords in late 2014, citing their tendency to overheat, melt, burn, and spark. It is difficult to imagine how such a design problem can exist in one the simplest items used in connection with a computer; a device that has been designed and built by numerous vendors for decades. Investigate the problem using on-line sources and present a single-page typeset report (single-spacing is okay) about the problem and its causes

1.22 Building trust from untrustworthy components

The ultimate goal of designers of dependable systems is to ascertain trustworthiness of system results, despite using components that are not totally trustworthy. Read the paper [Seth15] and present your assessment of the practicality of this goal and some of the methods proposed in a single-spaced typed page.

1.23 Risks of automation

Read the paper [Neum16] by Peter G. Neumann, moderator of the "Inside Risks" forum, write a half-page abstract for it, and answer the following questions:

a. What is total-system safety? (Provide a definition.)

b. Which area needs more work in the coming years: aviation safety or automotive safety?

c. What is the most important risk associated with the Internet of Things (IoT)?

d. Why are the requirements for security and law enforcement at odds?

1.24 Risks of infrastructure deterioration

In September 2018, gas explosions rocked a vast area in northeastern Massachusetts, leading to the loss of one life, many injuries, and destruction of property. Investigations revealed that just before the explosions, pipe pressure was 12 times higher than the safe limit. Using Internet sources, write a one-page report on this incident, focusing on how/why pressure monitors, automatic shut-off mechanisms, and human oversight failed to prevent the disaster.

1.25 The troubles of Boeing 737 Max 8

Following two crashes of Boeing 737 Max 8 passenger jets in late 2018 and early 2019, killing hundreds, airlines and various aviation authorities around the world grounded the planes until crash causes could be determined and attendant design flaws corrected. When Boeing 737 Max 8 was tested following its introduction, certain stability problems were detected, but rather than redesigning the plane, Boeing chose to augment them with a software system to compensate for the problems in flight. Both crashes were attributed to flaws in the aforementioned software and lack of certain monitoring instruments that could have warned the pilots of emerging challenges. As of late 2019, the planes remained grounded, because Boeing's purported fixes have not satisfied aviation authorities. Using on-line sources, study the causes of the two crashes, reasons for grounding the planes, and results of investigations conducted after the grounding, presenting your results in a 2-page report.

1.26 Risks of trusting the physics of sensors

Read the paper [Fu18] and answer the following questions:

a. Why do the authors think that there are risks involved in the physics of sensors?
b. How can a voice-activated system be tricked into exhibiting malicious behavior?
c. Which other application areas involving sensors are particularly vulnerable?
d. In what ways do current educational systems hinder the design of secure embedded systems?

1.27 Outdated software on safety-critical systems in the US

The glitch that led to hundreds of flight cancellations and thousands of flight delays in early January 2023 was due to defective software introduced into the US FAA system by outside contractors. The outdated system was already a cause of much frustration for airline pilots, even before the software failure that took it out. Using Internet sources, write a one-page, typeset report with single spacing, describing the problem, how long it affected airlines' operations, and how it was resolved.

1.28 Certification of safety-critical systems

Certification of computer systems is an important requirement, which has become even more crucial, owing to a significant growth in software complexity. Existing certification methods are remnants of days when safety violations arose primarily from hardware failures and malfunctions. Read the article [Leve23] and outline in no more than 2 typed pages the need for taking a fresh look at the safety-certification problem and what you consider to be the three most-promising changes to current certification methods.

References and Further Readings

[Ackl13] Ackley, David H., “Beyond Efficiency,” *Communications of the ACM*, Vol. 56, No. 10, pp. 38-40, October 2013.

[Ande82] Anderson, T. and P. A. Lee, “Fault Tolerance Terminology Proposals,” *Proc. Int’l Symp. Fault-Tolerant Computing*, 1982, pp. 29-33.

[Aviz82] Avižienis, A., “The Four-Universe Information System Model for the Study of Fault Tolerance,” *Proc. Int’l Symp. Fault-Tolerant Computing*, 1982, pp. 6-13.

[Aviz04] Avižienis, A., J.-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Trans. Dependable and Secure Computing*, Vol. 1, No. 1, pp. 11-33, January-March 2004.

[Aviz04a] Avižienis, A., “Dependable Systems of the Future: What Is Still Needed?” *Proc. 18th IFIP World Computer Congress*, Toulouse, August 2004.

[Carp96] Carper, K. L., “Construction Pathology in the United States,” *Structural Engineering International*, Vol. 6, No. 1, pp. 57-60, February 1996.

[Cart82] Carter, W. C., “A Time for Reflection,” *Proc. Int’l Symp. Fault-Tolerant Computing*, 1982, p. 41.

[Fu18] Fu, K. and W. Xu, “Risks of Trusting the Physics of Sensors” (“Inside Risks” Column), *Communications of the ACM*, Vol. 61, No. 2, pp. 20-23, February 2018.

[Hosf60] Hosford, J. E., “Measures of Dependability,” *Operations Research*, Vol. 8, No. 1, pp. 53-64, January/February 1960.

[Jain11] Jain, M. and R. Gupta, “Redundancy Issues in Software and Hardware Systems: An Overview,” *Int’l J. Reliability, Quality and Safety Engineering*, Vol. 18, No. 1, pp. 61-98, 2011.

[Lapr82] Laprie, J.-C., “Dependability: A Unifying Concept for Reliable Computing,” *Proc. Int’l Symp. Fault-Tolerant Computing*, 1982, pp. 18-21.

[Lapr85] Laprie, J.-C., “Dependable Computing and Fault Tolerance: Concepts and Terminology,” *Proc. Int’l Symp. Fault-Tolerant Computing*, 1985, pp. 2-11.

[Leve23] Leveson, N. G. and J. P. Thomas, "Certification of Safety-Critical Systems" (Inside Risks Column), *Communications of the ACM*, Vol. 66, No. 10, pp. 22-26, October 2023. https://dl.acm.org/doi/pdf/10.1145/3615860

[Neum16] Neumann, P. G., “Risks of Automation: A Cautionary Total-System Perspective of our Cyberfuture,” *Communications of the ACM*, Vol. 59, No. 10, pp. 26-30, October 2016.

[Olse08] Olsen, K. A., “The $100,000 Keying Error,” *IEEE Computer*, Vol. 41, No. 4, pp. 108 & 106-107, April 2008.

[Parh78] Parhami, B., “Errors in Digital Computers: Causes and Cures,” *Australian Computer Bulletin*, Vol. 2, No. 2, pp. 7-12, March 1978.

[Parh94] Parhami, B., “A Multi-Level View of Dependable Computing,” *Computers & Electrical Engineering*, Vol. 20, No. 4, pp. 347-368, 1994.

[Parh97] Parhami, B., “Defect, Fault, Error, ... , or Failure?” *IEEE Trans. Reliability*, Vol. 46, No. 4, pp. 450-451, December 1997.

[Seth15] Sethumadhavan, S., A. Waksman, M. Suozzo, Y. Huang, and J. Eum, “Trustworthy Hardware from Untrusted Components,” *Communications of the ACM*, Vol. 58, No. 9, pp. 60-71, September 2015.

[Siew92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 2nd Ed., 1992.

2 Dependability Attributes

"The shifts of fortune test the reliability of friends."

Cicero

"Then there is the man who drowned crossing a stream with an average depth of six inches."

W. I. E. Gates

Topics in This Chapter

Based on our discussions in Chapter 1, computer system dependability is not a single concept; rather, it possesses several different facets or components. These include reliability, availability, testability, maintainability (the so-called "ilities"), graceful degradation, robustness, safety, security, and integrity. In this chapter, we quantify some of the key attributes of a dependable computer or computer-based system, explore the relationships among these quantitative measures, and show how they might be evaluated in rudimentary cases, with appropriate simplifying assumptions. More detailed examination of dependability modeling techniques will be presented in Chapter 3, covering combinational models, and Chapter 4, introducing state-space models.

2.1 Aspects of Dependability

In Chapter 1, we briefly touched upon the notions or reliability, safety, and availability, as different facets of computer system dependability. In this chapter, we provide precise definitions for these concepts and also introduce other related and distinct aspects of computer systems dependability, such as testability, maintainability, serviceability, graceful degradation, robustness, resilience, security, and integrity. Table 2.1 shows the list of concepts that will be dealt with, along with typical qualitative usages and associated quantitative aspects or measures. For example, in the case of reliability, we encounter statements such as "this system is ultrareliable" (qualitative usage) and "the reliability of this systems for a one-year mission is 0.98" (quantitative usage).

We devote the remainder of this section to a brief review of some needed concepts from probability theory. The review is intended as a refresher. Readers who have difficulties with these notions should consult one of the introductory probability/statistics textbooks listed at the end of the chapter; for example, [Papo90].

Let E be one of the possible outcomes of some experiment. By

$$\text{prob}[E] = 0.1$$

we mean that if the experiment is repeated many times, under the same conditions, the outcome will be E in roughly 10% of the cases. For example

$$\text{prob}[\text{System } S \text{ fails within 10 weeks}] = 0.1$$

means that out of 1000 systems of the same type, all operating under the same application and environmental conditions, about 100 will fail within 10 weeks.

Table 2.1 Dependability-related terms with their most common qualitative usages and quantifications (if any).

Term	Qualitative Usage(s)	Quantitative Measure(s)
Availability	Highly available High-availability Continuously available	(Pointwise) Availability Interval availability MTBF, MTTR
Integrity	High-integrity Tamper-proof Fool-proof	
Maintainability	Easily maintainable Maintenance-free Self-repairing	
Performability		Performability MCBF
Reliability	Reliable Highly reliable High-reliability Ultrareliable	Reliability MTTF or MTFF
Resilience	Resilient	
Robustness	Robust	Impairment tolerance count
Safety	High-safety Fail-safe	Risk
Security	Highly secure High-security Fail-secure	
Serviceability	Easily serviceable	
Testability	Easily testable Self-testing Self-checking	Controllability Observability

Abbreviations used: MCBF = mean computation between failures; MTBF = mean time between failures; MTFF = mean time to first failure; MTTF = mean time to failure; MTTR = mean time to repair.

Thus, probability values have physical significance and can be determined via experimentation. Of course, when the probabilities are very small, it may become impossible or costly to conduct the requisite experiments. For example, experimental verification that

$$\text{prob[Computer } C \text{ fails within 10 weeks]} = 10^{-6}$$

requires experimentation with many millions of computers.

When multiple outcomes are of interest, we deal with composite, joint, and conditional probabilities satisfying the following:

$$\begin{aligned}
\text{prob[not } A] &= 1 - \text{prob}[A] && \text{(2.1.CJC)} \\
\text{prob}[A \cup B] &= \text{prob}[A] + \text{prob}[B] - \text{prob}[A \cap B] \\
&= \text{prob}[A] + \text{prob}[B] \quad \text{if } A \text{ and } B \text{ are mutually exclusive} \\
\text{prob}[A \cap B] &= \text{prob}[A] \times \text{prob}[B] \quad \text{if } A \text{ and } B \text{ are independent} \\
\text{prob}[A \mid B] &= \text{prob}[A \cap B] / \text{prob}[B] \quad \{\text{read: probability of } A\text{, given } B\}
\end{aligned}$$

Suppose that we have observed 20 identical systems under the same conditions and measured the time to failure for each. The top part of Fig. 2.1 shows the distribution of the time to failure as a scatter plot. The *cumulative distribution function* (CDF) for the time to failure x (life length of the system), defined as the fraction of the systems that have failed before a given time t, is shown in the middle part of Fig. 2.1 in the form of a staircase. Of course with a very large sample, we would get a continuous CDF curve that goes from 0 for $t = 0$ to 1 for $t = \infty$. Finally, the *probability density function* (pdf) is shown at the bottom of Fig. 2.1. CDF represents the area under the pdf curve; i.e. the following relationships hold:

$$F(t) = \text{prob}[x \le t] = \int_0^t f(x)dx \qquad \text{(2.1.CDF)}$$

$$f(t) = \text{prob}[t \le x \le t + dt] / dt = dF(t) / dt \qquad \text{(2.1.pdf)}$$

Based on the preceding, the interpretation of the pdf $f(t)$ in Fig. 2.1 is that the probability of the system failing in the time interval $[t, t + dt]$ is $f(t)\, dt$. So, where the dots in the scatter plot are closer together, $f(t)$ assumes a larger value.

Once the CDF or pdf for a random variable has been determined experimentally, we might try to approximate it by a suitable equation and carry out various probabilistic calculations based on such an analytical model. Examples of commonly used distributions include uniform, normal, exponential, and binomial (Fig. 2.2).

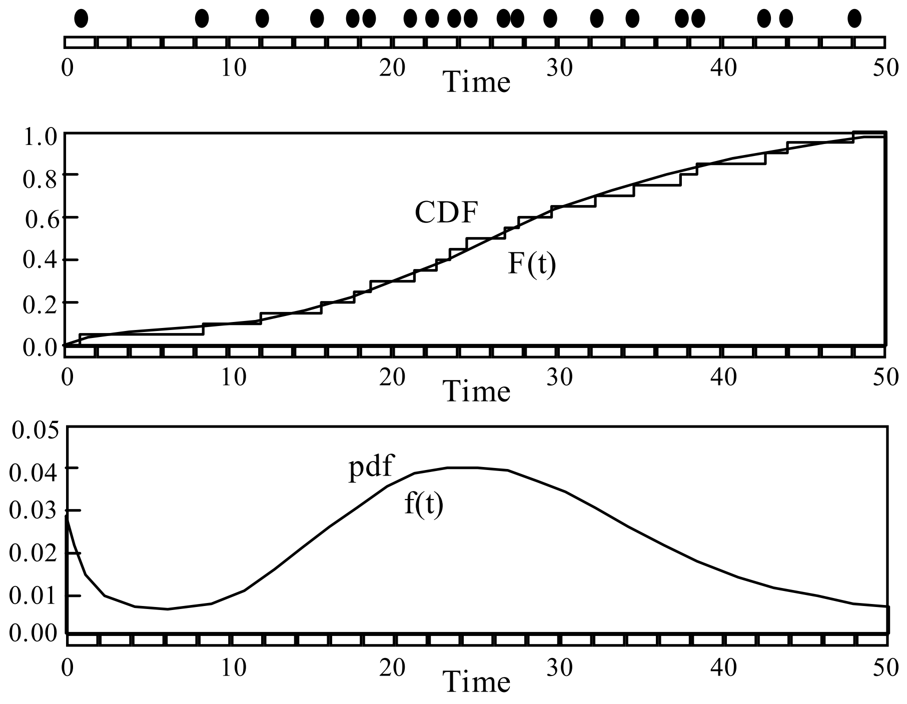


Fig. 2.1 **Scatter plot for the random variable representing the lifetime of a system, along with its cumulative distribution function (CDF) and probability density function (pdf).**

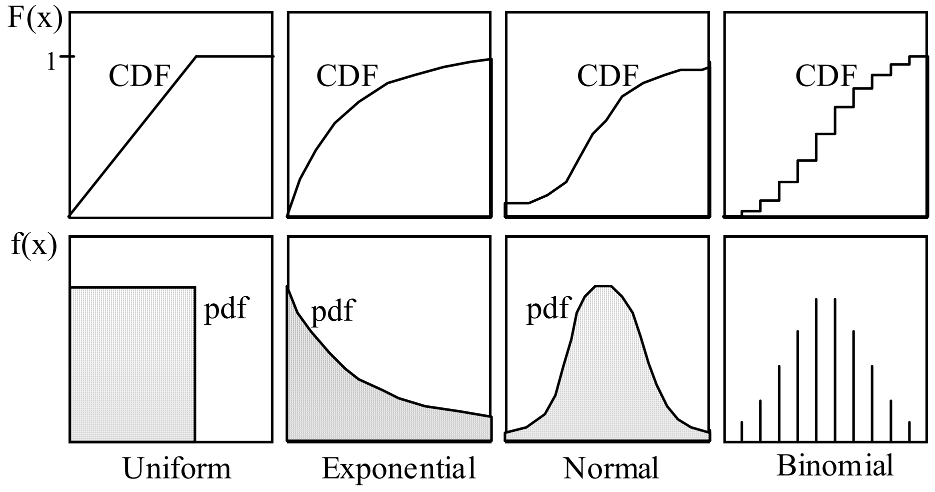


Fig. 2.2 **Some commonly used probability distributions, defined by their CDF and pdf graphs.**

Given a random variable x with a known probability distribution, its expected value, denoted as $E[x]$ or E_x, is defined as:

$$E_x = \int_{-\infty}^{\infty} x f(x) dx \quad \text{for continuous distributions} \quad (2.1.\text{EV})$$

$$= \sum_k x_k f(x_k) \quad \text{for discrete distributions}$$

The interpretation of E_x is that it is the mean of the values observed for x over a large set of experiments. The *variance* σ_x^2, and *standard deviation* σ_x, of a random variable x are indicators of the spread of x values relative to E_x:

$$\sigma_x^2 = \int_{-\infty}^{\infty} (x - E_x)^2 f(x) dx \quad \text{for continuous distributions} \quad (2.1.\text{Var1})$$

$$= \sum_k (x_k - E_x)^2 f(x_k) \quad \text{for discrete distributions}$$

Based on the preceding definition, we easily find:

$$\sigma_x^2 = E[(x - E_x)^2] = E[x^2] - (E_x)^2 \quad (2.1.\text{Var2})$$

When dealing with two random variables, the notion of *covariance* ψ is of some interest:

$$\begin{aligned}\psi_{x,y} &= E[(x - E_x) \times (y - E_y)] \\ &= E[x \times y] - E_x \times E_y\end{aligned} \qquad (2.1.\text{Cov})$$

Given the covariance $\psi_{x,y}$, one can define the *correlation coefficient*

$$\rho_{x,y} = \psi_{x,y} / (\sigma_x \times \sigma_y) \qquad (2.1.\text{Corr})$$

which is the expected value of the product of centered and normalized random variables $(x - E_x)/\sigma_x$ and $(y - E_y)/\sigma_y$.

When $\rho_{x,y} = 0$, we say that x and y are *uncorrelated* and we have $E[x \times y] = E_x \times E_y$. Independent random variables are necessarily uncorrelated, but the converse is not always true (it is true for the normal distribution, though).

2.2 Reliability and MTTF

Reliability is an important engineering concept whose theoretical development was apparently started by von Braun during World War II in Germany after the first series of ten V-1 missiles all blew up on the launching pads [Henl81]. Principles of reliability engineering can be found in many book (e.g., [Lewi87], [Tobi86]). Reliability is an appropriate measure of dependability for a two-state system that starts in a fully functional or "good" state and moves to a "failed" state when it can no longer operate as specified. System functionality is completely lost when the transition into the failed state occurs (Fig. 2.3).

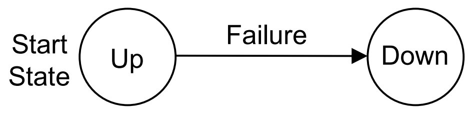


Fig. 2.3 Two-state (nonrepairable) system.

The reliability $R(t)$, defined as the probability that the system remains in the "Good" state throughout the time interval $[0, t]$, was the only dependability measure of interest to early designers of dependable computer systems. Such systems were typically used for spacecraft guidance and control where repairs were impossible and the system was effectively lost upon the first failure. Thus, the reliability $R(t_M)$ for the mission duration t_M accurately reflected the probability of successfully completing the mission and thus achieving acceptable reliabilities for large values of t_M (the so-called *long-life* systems) became the main challenge. Reliability is related to the CDF of the system lifetime, also known as *unreliability*, by:

$$F(t) = 1 - R(t) \qquad \text{(2.2.Unrel)}$$

Let $z(t)\,dt$ be the probability of system failure between times t and $t + dt$. The function $z(t)$ is called the *hazard function*. Then, the reliability $R(t)$ satisfies:

$$R(t + dt) = R(t)\,[1 - z(t)\,dt] \qquad \text{(2.2.Haz1)}$$

This is simply a statement of the fact that for a system to be functional until time $t + dt$, it must have been functional until time t and must not fail in the interval $[t, t + dt]$ of duration dt. From the preceding equation, we obtain:

$$dR(t)/R(t) = -z(t)\,dt \qquad (2.2.\text{Haz2})$$

$$R(t) = \exp(-\int_0^t z(x)dx) \qquad (2.2.\text{Haz3})$$

For a constant *hazard rate* λ, that is, for $z(t) = \lambda$, we obtain the exponential reliability law which we took for granted in Section 1.1.

$$R(t) = e^{-\lambda t} \qquad (2.2.\text{Exp})$$

The hazard function $z(t)$, reliability $R(t)$, CDF of failure $F(t)$, and pdf of failure $f(t)$ are related as follows:

$$z(t) = f(t)/R(t) = f(t)/[1 - F(t)] \qquad (2.2.\text{Haz4})$$

We can thus view $z(t)$ as the conditional probability of failure occurring in the time interval $[t, t + dt]$, given that failure has not occurred up to time t. With a constant hazard rate λ, or exponential reliability law, failure of the system is independent of its age, that is, the fact that it has already survived for a long time has no bearing on its failure probability over the next unit-time interval.

Note that the reliability $R(t)$ is a monotonic (nonincreasing) function of time. Thus, when the survival of a system for the entire duration of a mission of length t_M is at issue, reliability can be specified by the single numeric index $R(t_M)$. Mean time to failure (MTTF), or mean time to first failure (MTFF), is another single-parameter indicator of reliability. The mean time to failure for a system is given by:

$$\text{MTTF} = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt \qquad (2.2.\text{MTTF})$$

The first equality above is essentially the definition of expected value of the time to failure while the second one, indicating that MTTF is equal to the area under the reliability curve, is easily provable.

In addition to the constant hazard rate $z(t) = \lambda$, which leads to the exponential reliability law, the distributions shown in Table 2.2 have been suggested for reliability modeling.

The Weibull distribution has two parameters: The shape parameter α, and the scale parameter λ. Both exponential and Raleigh distributions are special cases of the Weibull distribution, corresponding to $\alpha = 1$ and $\alpha = 2$, respectively. Similarly, the Gamma distribution covers the exponential and Erlang distributions as special cases ($b = 1$ and b an integer, respectively). The parameters of the various reliability models in Table 2.2 can be derived based on field failure data.

The *Gamma function* $\Gamma(\theta)$, used in the formulas in Table 2.2, is defined as:

$$\Gamma(\theta) = \int_0^\infty x^{\theta-1} e^{-x} dx \qquad (2.2.\text{Gam1})$$

In particular, $\Gamma(1/2) = \sqrt{\pi}$, $\Gamma(1) = \Gamma(2) = 1$, $\Gamma(k) = (k-1)!$ when k is an integer, and:

$$\Gamma(\theta + 1) = \theta\,\Gamma(\theta) \qquad \text{for any } \theta \qquad (2.2.\text{Gam2})$$

For this reason, the Γ function is called *generalized factorial*. This last equation allows us to compute $\Gamma(\theta)$ recursively based on the values of $\Gamma(\theta)$ for $1 \le \theta < 2$.

The discrete versions of the exponential, Weibull, and normal distributions are known as geometric, discrete Weibull, and binomial distributions, respectively. The *geometric distribution* is obtained by replacing $e^{-\lambda}$ by the discrete probability q of survival over one time step and time t by the number k of time steps, leading to the reliability equation:

$$R(k) = q^k = (1-p)^k \qquad (2.2.\text{Geom})$$

In the case of the *discrete Weibull distribution*, $e^{-\lambda^\alpha}$ is replaced with q, and t with k, leading to:

$$R(k) = q^{k^\alpha} = (1-p)^{k^\alpha} \qquad (2.2.\text{Weib})$$

Closed-form formulas are generally hard to obtain for the parameters of interest with the discrete Weibull distribution. Finally, the *binomial distribution* is characterized by the reliability equation:

$$R(k) = 1 - \sum_{j=0}^{k} \binom{n}{j} p^j q^{n-j} \qquad 0 \le k \le n \qquad \text{(2.2.Bino)}$$

When n is large, the binomial distribution can be approximated by the normal distribution with parameters $\mu = np$ and $\sigma = \sqrt{npq}$.

Table 2.2 Some commonly assumed continuous failure distributions and their associated reliability and MTTF formulas.

Distribution	$z(t)$	$f(t)$	$R(t) = 1 - F(t)$	MTTF
Exponential	λ	$\lambda e^{-\lambda t}$	$e^{-\lambda t}$	$1/\lambda$
Rayleigh	$2\lambda(\lambda t)$	$2\lambda(\lambda t)e^{-(\lambda t)^2}$	$e^{-(\lambda t)^2}$	$(1/\lambda)\sqrt{\pi}/2$
Weibull	$\alpha\lambda(\lambda t)^{\alpha-1}$	$\alpha\lambda(\lambda t)^{\alpha-1}e^{-(\lambda t)^\alpha}$	$e^{-(\lambda t)^\alpha}$	$(1/\lambda)\,\Gamma(1+1/\alpha)$
Erlang		$\frac{\lambda}{(k-1)!}(\lambda t)^{k-1}e^{-\lambda t}$	$e^{-\lambda t}\sum_{i=0}^{k-1}(\lambda t)^i/i!$	k/λ
Gamma		$\frac{\lambda}{\Gamma(b)}(\lambda t)^{b-1}e^{-\lambda t}$		
Normal*		$\frac{1}{\sigma\sqrt{2\pi}}\, e^{-(t-\mu)^2/(2\sigma^2)}$		

(*) Reliability, and MTTF formulas for the normal distribution are quite involved. One can use numerical tables listing the values of the integral $(1/\sqrt{2\pi})\int_{-\infty}^{t} e^{-x^2/2}dx$ to evaluate $R((t-\mu)/\sigma)$.

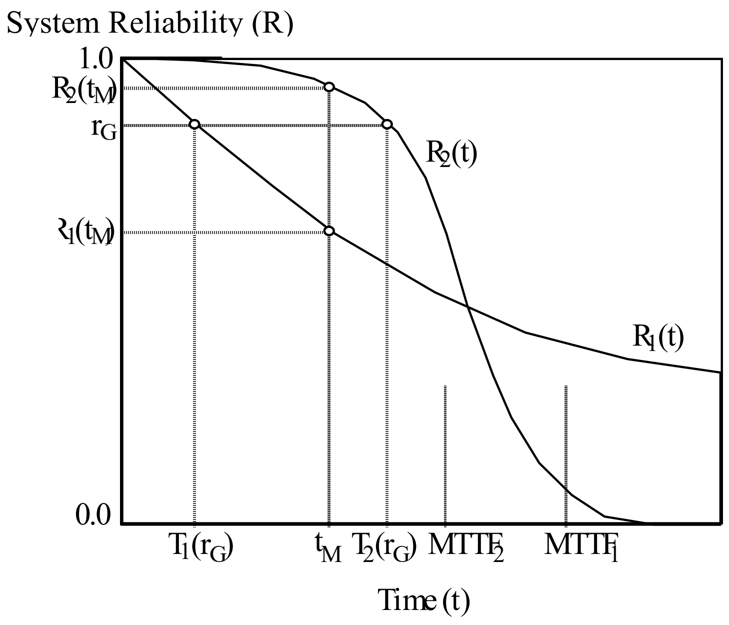


Fig. 2.4 **Example reliability functions for systems 1 and 2.**

Referring to Fig. 2.4, we observe that the time t_1 to the first failure is a random variable whose expected value is the system's MTTF. Even though it is true that a higher MTTF implies a higher reliability in the case of nonredundant systems, the use of MTTF is misleading when redundancy is applied. For example, in Fig. 2.4, System 1 with reliability function $R_1(t)$ is much less reliable than System 2 with reliability function $R_2(t)$ for the mission duration t_M, but it has a longer MTTF. Since usually $t_M \ll$ MTTF, the shape of the reliability curve is much more important than the numerical value of MTTF.

The *reliability difference* $R_2 - R_1$ and *reliability gain* R_2/R_1 are natural measures for comparing two systems having reliabilities R_1 and R_2. In order to facilitate the comparison of highly reliable systems (with reliability values very close to 1), several comparative measures have been suggested including *reliability improvement factor*, of System 2 over System 1, for a given mission time t_M

$$\mathrm{RIF}_{2/1}(t_M) = [1 - R_1(t_M)] / [1 - R_2(t_M)] \qquad (2.2.\mathrm{RIF})$$

reliability improvement index, of System 2 over System 1, for the mission time t_M

$$\mathrm{RII}_{2/1}(t_M) = \log R_1(t_M) / \log R_2(t_M) \qquad (2.2.\mathrm{RII})$$

mission time extension for a given reliability goal r_G

$$\text{MTE}_{2/1}(r_G) \;=\; T_2(r_G) - T_1(r_G) \;=\; R_2^{-1}(r_G) - R_1^{-1}(r_G) \qquad \text{(2.2.MTE)}$$

and *mission time improvement factor* for a given reliability goal r_G

$$\text{MTIF}_{2/1}(r_G) \;=\; T_2(r_G) \,/\, T_1(r_G) \;=\; R_2^{-1}(r_G) \,/\, R_1^{-1}(r_G) \qquad \text{(2.2.MTIF)}$$

where the *(mission) time function* T is the inverse of the reliability function R. Thus, $R(T(r)) = r$ and $T(R(t)) = t$.

Example 2.1: Comparing system reliabilities Systems 1 and 2 have constant failure rates of $\lambda_1 = 1/\text{yr}$ and $\lambda_2 = 2/\text{yr}$. Quantify the reliability advantage of System 1 over System 2 for a one-month period.

Solution: The reliabilities of the two systems for a one-month period are $R_1(1/12) = e^{-1\times 1/12} = 0.9200$ and $R_2(1/12) = e^{-2\times 1/12} = 0.8465$. The reliability advantage of System 1 over System 2 can be quantified in the following ways:

$R_1(1/12) - R_2(1/12) = 0.9200 - 0.8465 = 0.0735$

$R_1(1/12) \,/\, R_2(1/12) = 0.9200 \,/\, 0.8465 = 1.0868$

$\text{RIF}_{1/2}(1/12) = (1 - 0.8465) \,/\, (1 - 0.9200) = 1.9188$

$\text{RII}_{1/2}(1/12) = \log 0.8465 \,/\, \log 0.9200 = 1.9986$

For a reliability goal of 0.9, the mission time extension of System 1 over System 2 is derived as

$\text{MTE}_{1/2}(0.9) = (-\ln 0.9)(1/\lambda_1 - 1/\lambda_2) = 0.0527 \text{ yr} = 19.2 \text{ days}$

while the mission time improvement factor of System 1 over System 2 is:

$\text{MTIF}_{1/2}(0.9) = \lambda_2 / \lambda_1 = 2.0$

Example 2.2: Analog of Amdahl's law for reliability Amdahl's law states that if in a unit-time computation a fraction f doesn't change and the remaining fraction $1 - f$ is speeded up to run p times as fast, the overall speedup will be $s = 1 \,/\, (f + (1 - f)/p)$. Show that a similar formula applies to the reliability improvement index after improving the failure rate for some parts of a system.

Solution: Consider a system with two segments, having failure rates ϕ and $\lambda - \phi$, respectively. Upon improving the failure rate of the second segment to $(\lambda - \phi)/p$, we have $\text{RII} = \log R_{\text{original}} \,/\, \log R_{\text{improved}} = \lambda \,/\, (\phi + (\lambda - \phi)/p)$. Letting $\phi / \lambda = f$, we obtain: $\text{RII} = 1 \,/\, (f + (1 - f)/p)$

2.3 Availability, MTTR, and MTBF

As mentioned earlier, at first reliability was the only measure of interest in evaluating computer system dependability. The advent of time-sharing systems brought with it a concern for the continuity of computer service (the so-called *high-availability* systems) and thus minimizing the "down time" became a prime concern. *Interval availability*, or simply *availability*, $A(t)$, defined as the fraction of time that the system is operational during the interval $[0, t]$, is the natural dependability measure in this respect. The limit A of $A(t)$ as t tends to infinity, if it exists, is known as the *steady-state availability*.

$$A = \lim_{t \to \infty} A(t) \qquad (2.3.\text{Av1})$$

Availability is a function not only of how rarely a system fails but also of how quickly it can be repaired upon failure. Thus, the time to repair is important and *maintainability* is used as a qualitative descriptor for ease of repair (i.e., faster or less expensive maintenance procedures).

Clearly, maintainability is closely related to availability in that high availability cannot be achieved without attention to maintenance speed-up techniques. *Serviceability* is sometimes used as a synonym for maintainability. As the concern with dependability spread from highly advanced special-purpose systems to commercial environments, terms such as maintainability and serviceability became a permanent part of computer jargon. The attention to ease of maintenance is not new, as even some second-generation computers had extensive hardware and software aids for this purpose [Cart64].

The probability $a(t)$ that a system is available at time t is known as its *pointwise availability* (which is the same as reliability when there is no repair). To take repair into consideration, one can consider a repair rate function $z_r(t)$ which results in the probability of unsuccessful repair up to time t having an equation similar to reliability, with various distributions possible. For example, one can have exponentially distributed repair times, with repair rate μ:

$$\text{prob[repair is not completed in } t \text{ time units]} = e^{-\mu t}$$

Consider a two-state repairable system, as shown in Fig. 2.5. The system begins operation in the “Up” state, but then moves back and forth between the two states due to failures and successful repairs. The duration of the system staying in the “Up” state is a random variable corresponding to the time to first failure, while that of the “Down” state is the time to successful repair.

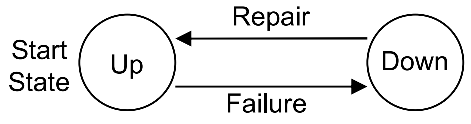


Fig. 2.5 Two-state repairable system.

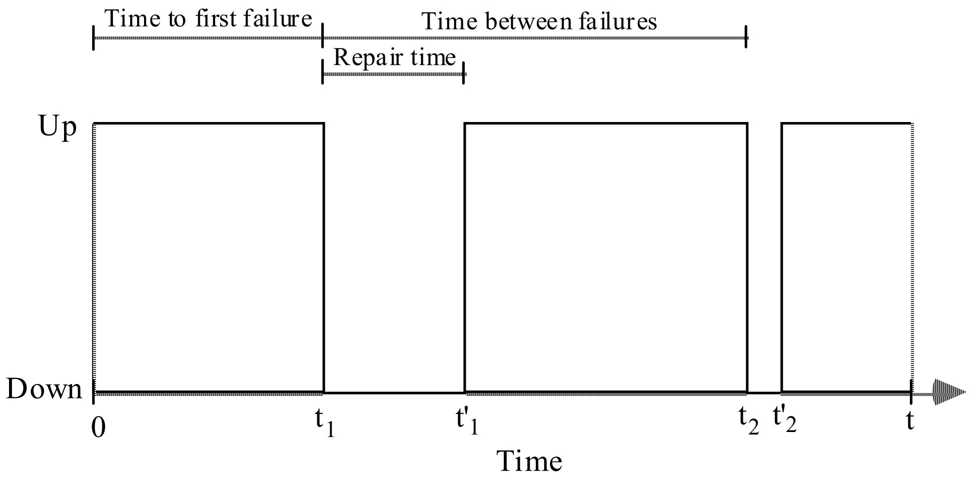


Fig. 2.6 System up and down times contributing to its availability.

Fig. 2.6 depicts the variations of the state of an example repairable two-state system with time. Until time t_1, the example system in Fig. 2.6 is continuously available and thus has an interval availability of 1. After the first failure at time t_1, availability drops below 1 and continues to decrease until the completion of the first repair at time t'_1. The repair time is thus $t'_1 - t_1$. The second failure occurs at time t_2, yielding a time between failures of $t_2 - t_1$. Over a period of time, the expected value of $t'_i - t_i$ and $t_{i+1} - t_i$ are known as *mean time to repair* (MTTR) and *mean time between failures* (MTBF), respectively.

In the special case of $z_r(t) = \mu$, i.e. a constant repair rate, the steady-state availability for a system with a constant failure rate λ is:

$$A = \frac{\mu}{\lambda + \mu} \tag{2.3.Av2}$$

We will formally derive the preceding equation in Chapter 4 as a simple example of state-space modeling. For now, we present an intuitive justification by noting that, with exponential failure and repair time distributions, we have MTTF = $1/\lambda$ and MTTR = $1/\mu$, leading to

$$A = \frac{1/\lambda}{1/\lambda + 1/\mu} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}} \tag{2.3.Av3}$$

where MTBF = MTTF + MTTR = $1/\lambda + 1/\mu$ is the *mean time between failures*.

Pointwise availability $a(t)$ and interval availability $A(t)$ are related as follows:

$$A(t) = (1/t) \int_0^t a(x)\, dx \tag{2.3.Av4}$$

Both pointwise and interval availability are relatively difficult to derive in general. However, for most practical purposes, the steady-state availability A can be used in lieu of pointwise availability. This is because if $a(t)$ can be assumed to be a constant, then it must be equal to $A(t)$ by the preceding equation. Interval availability being a constant in turn implies $A(t) = A$. As an example, if a system is available 99% of the time in the long run, then we can assume that at any given time instant, it will be available with probability 0.99.

A high-availability computer system must be *robust* and *resilient*. A standard dictionary defines "robustness" as *strength, vigor, roughness, health*, the opposite of *delicate, sickly*, and "resilience" as *an ability to recover from or adjust easily to misfortune or change*, a synonym for *elasticity*. Anderson [Ande85] defines a resilient computing system as one that is "capable of providing dependable service to its users over a wide range of potentially adverse circumstances" and notes that trustworthiness and robustness are the key attributes of such a system. He adds that a *robust* computer system "retains its ability to deliver service in conditions which are beyond its normal domain of operation, whether due to harsh treatment, or unreasonable service requests, or misoperation, or the impact of faults, or lack of maintenance, etc."

Example 2.3: Availability formula Consider exponential and repair laws, with failure rate λ and repair rate μ. In the time interval $[0, t]$, we can expect λt failures which take $\lambda t/\mu$ time units to repair, on the average. Thus, for large t, the system will be under repair for $\lambda t/\mu$ time units out of t time units, yielding the availability $1 - \lambda/\mu$. Is there anything wrong with this argument, given that the availability was previously derived to be $A = 1 - \lambda/(\lambda + \mu)$?

Solution: The number of expected failures is actually slightly less than λt, because the system is operational only in a fraction A of time t, where A is the availability. Correcting the argument, we note that $A\lambda t$ failures are expected over the available time At, yielding an expected repair time of $A\lambda t/\mu$ time units. Thus, the availability A satisfies the equation $A = 1 - A\lambda/\mu$, where the last term is the fraction of the time t that is spent on repair. This yields $A = 1/(1 + \lambda/\mu) = \mu/(\lambda + \mu)$.

Continual increase in system complexities and the difficulties in testing for initial system verification and subsequent preventive and diagnostic maintenance have led to concern for *testability*. Since any preventive or diagnostic maintenance procedure is based on testing, maintainability and testability are closely related. Testability is often quantified by the complementary notions of *controllability* and *observability*. In the context of digital circuits, controllability is an indicator of the ease with which the various internal points of the circuit can be driven to desired states by supplying values at the primary inputs. Similarly, observability indicates the ease with which internal logic values can be observed by monitoring the primary outputs of the circuit [Gold79].

In practice, computer systems have more than two states. There may be multiple operational states where the system exhibits different computational capabilities. Availability analysis for gracefully degrading systems will be discussed along with performability in Section 2.4. It is noteworthy that performability and similar terms, some of which were listed in the preceding paragraph, are sometimes collectively referred to as the "-ilities". Several other informally defined "-ilities" can be found in the literature (survivability, reconfigurability, diagnosability, and so on), although none has found widespread acceptance.

2.4 Performability and MCBF

Widespread use of multiprocessors and gracefully degrading systems, that did not obey the all-or-none mode of operation implicit in conventional reliability and availability models, caused some difficulties in dependability evaluation. Consequently, *performability* was suggested as a relevant measure. Again the desirability of a simple numeric measure led to the suggestion of mean computation before failure (MCBF), although the use of this measure did not become as widespread as the MTTF and MTBF of the earlier era. These concepts will be discussed in the remainder of this section.

The performability of a gracefully degrading system at time t depends on the set of resources available, the computational capability provided by these resources, and the "worth" associated with each capability level. As such, complete discussion of performability is beyond the scope of this section. Rather we choose to illustrate the tools and techniques by means of a simple example.

Consider a dual-processor computer system with two performance levels; both processors working (worth = 2) and only one processor working (worth = 1), ignoring all other resources. If the processors fail one at a time, and are repaired one at a time, then the system's state diagram is as shown in Fig. 2.7. In Chapter 4, we will show how the steady-state probabilities for the system being in each of its states can be determined. If these probabilities are p_{Up2}, p_{Up1}, and $1 - p_{\mathrm{Up2}} - p_{\mathrm{Up1}}$, then, the performability of the system is:

$$P = 2p_{\mathrm{Up2}} + p_{\mathrm{Up1}}$$

As a numerical example, $p_{\mathrm{Up2}} = 0.92$ and $p_{\mathrm{Up1}} = 0.06$ lead to $P = 1.90$. In other words, the performance level of the system is equivalent to 1.9 processors on the average, with the ideal performability being 2.

When processors fail and are repaired independently, and if Processor i has a steady-state availability A_i, then performability of the system above becomes:

$$P = A_1 + A_2$$

More generally, i.e., when the resources are not identical, each availability A_i of a resource must be multiplied by the worth of that resource. Note that the independent repair assumption implies that maintenance resources are not limited. If there is only one repair person, say, then this assumption may not be valid.

A fail-soft, or gracefully degrading, system may be said to be “available” when its performance is at or above a minimum threshold. Thus, one can readily derive the availability figure of merit based on performability calculations. Assuming that the system of Fig. 2.7 is available in states Up2 and Up1, its availability with our preceding assumptions will be $A = p_{\mathrm{Up2}} + p_{\mathrm{Up1}} = 0.92 + 0.06 = 0.98$.

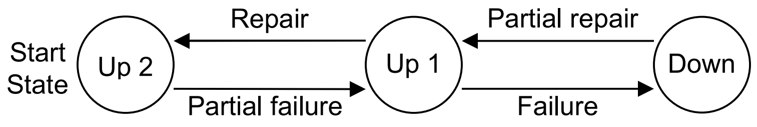


Fig. 2.7 **Three-state repairable system with different performance parameters in its two nonfailed states.**

Example 2.4: Performability improvement factor With a low rate of failure, performability will be close to its ideal value. For this reason, a performability improvement factor, PIF, can be defined in a manner similar to RIF, given in equation 2.2.RIF. For the two-processor system analyzed in the preceding paragraphs, determine the PIF relative to a fail-hard system that would go down when either processor fails.

Solution: The performability of the fail-hard system is readily seen to be $2p_{\mathrm{Up2}} = 2 \times 0.92 = 1.84$. Given the ideal performability of 2, the performabilities of our fail-hard and fail-soft systems in relative terms are $1.84/2 = 0.92$ and $1.90/2 = 0.95$, respectively. Thus:

$$\mathrm{PIF}_{\text{fail-soft/fail-hard}} = (1 - 0.92) / (1 - 0.95) = 1.6$$

Note that like RIF, PIF is useful for comparing different designs. The result 1.6 does not have any physical significance. The performability ratio in this example is $0.95 / 0.92 = 1.033$, which is a true indicator of the increase in expected performance.

Figure 2.8 depicts the variations of the state of an example three-state repairable system with time. Up to time t_1, the example system in Fig. 2.8 is continuously available with maximal performance. After the partial failure at time t_1, performance drops. Subsequently, at time t_2, the system fails completely. After partial repair is completed at time t'_2, system performance goes up and is eventually restored to its full level at time t'_1. The full repair time is, therefore, $t'_1 - t_1$, with partial repair taking $t'_2 - t_2$ time.

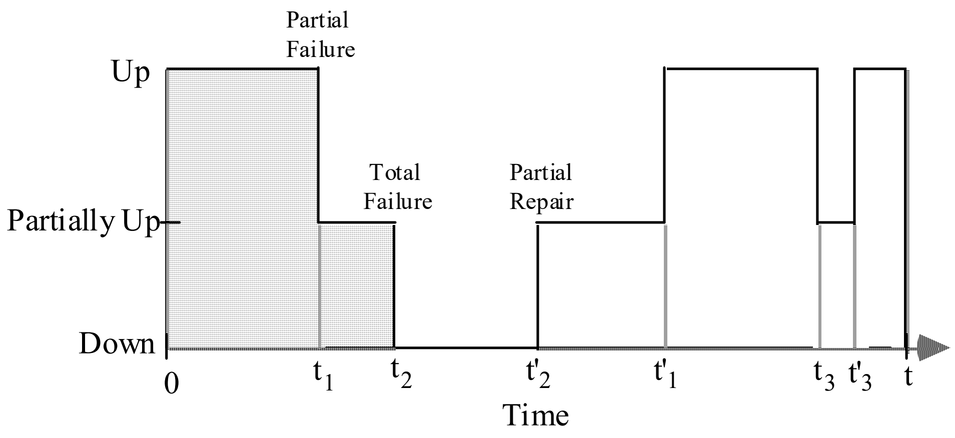


Fig. 2.8 **System up, partially up, and down times contributing to its performability.**

The shaded area in Fig. 2.8 represents the computational power that is available prior to the first total failure. If this power is utilized in its entirety, then the shaded area represents the amount of computation that is performed before total system failure. The expected value of this parameter is known as *mean computation before failure* (MCBF). Since no computation is performed in the totally failed state, MCBF can be viewed as representing *mean computation between failures*. Thus, MCBF is to performability as MTBF is to reliability.

Note that performability generalizes both reliability and performance. For a system that never fails, performability is the same as performance. For a system that has a single performance level (all or none), performability is synonymous with reliability; i.e., performance level is 100% iff the system has not failed.

One approach to computing MCBF is to use:

$$\text{MCBF} = \text{Performability} \times \text{MTTF} \qquad (2.4.\text{MCBF})$$

If availalability and MTTR are known, MTTF can be found from equation 2.3.Av3.

2.5 Integrity and Safety

The measures discussed thus far deal with the operation and performance of the computer system (and frequently only with the hardware) rather than with the integrity and success of the computations performed. Neither availability nor performability distinguishes between a system that experiences 30 two-minute outages per week and one that fails once per week but takes an hour to repair. Increasing dependence on transaction processing systems and safety-critical applications of computers has led to new concerns such as integrity, safety, security, and privacy. The first two of these concerns, and the corresponding dependability measures, are treated in this section. Security and privacy will be discussed in Section 2.6.

The two attributes of *integrity* and *safety* are similar; integrity is inward-looking and relates to the capacity of a system to protect its computational resources and data under adverse circumstances, while safety is outward-looking and pertains to consequences of incorrect actions for the system environment and users. One can examine system integrity by assigning the potential causes and consequences of failures to a number or a continuum of “severity” classes. If computational resources and data are not corrupted due to low-severity causes, then the system fares well on *integrity*. If the failure of a system seldom has severe external consequences, then the system is high on *safety*.

Integrity is a qualitative system attribute, although certain aspects of it can be quantified. System integrity can be ensured via rapid fault/error detection, frequent data back-ups, mechanisms that can isolate malfunctioning subsystems, and restoration via hot-swapped modules. A high-integrity system continues to provide reasonable service, while also protecting data files and other system resources, in the face of undesirable events originating from inside or outside the system.

Safety, on the other hand, is almost always quantified. Leveson [Leve86] defines *safety* as “the probability that conditions [leading] to mishaps (*hazards*) do not occur, whether or not the intended function is performed”. Central to the quantification of safety is the notion of *risk*. A standard dictionary defines risk as “the *possibility* of loss or injury”. Reliability engineers use *probability* instead of possibility. The expected loss or risk associated with a particular failure is a function of both its severity and its probability. More precisely:

$$\underset{[\textit{consequence / unit time}]}{\text{risk}} = \underset{[\textit{events / unit time}]}{\text{frequency}} \times \underset{[\textit{consequence / event}]}{\text{magnitude}} \qquad (2.5.\text{Risk1})$$

An alternate form of the risk equation is:

$$\text{risk} = \text{probability} \times \text{severity} \qquad (2.5.\text{Risk2})$$

The “magnitude” parameter in Eqn. (2.5.Risk1) or “severity” in Eqn. (2.5.Risk2) is measured in some agreed-upon unit. In many cases, this is done by associating a dollar cost with the occurrence of each undesirable event.

For example, the approximate individual risk (early fatality probability per year) associated with motor vehicle accidents is 3×10^{-4} which is about 10 times the risk of drowning, 100 times the risk of railway accidents, and 1000 times the risk of being killed by a hurricane ([Henl81], p. 11). Individual risks below 10^{-6} per year are generally deemed acceptable. Computer scientists and engineers have so far only scratched the surface of safety evaluation techniques and much more work in this area can be expected in the coming years.

To put our discussion of safety on the same footing as those of reliability, availability, and performability, we envisage the three-state system model of Fig. 2.9. Certain adverse condition may cause the system to fail in an unsafe manner; the probability of these events must be minimized. Safe failures, on the other hand, are acceptable. The provision of a separate “Safe Down” state, rather than merging it with the “Up” state, is useful in that the two states may be given different weights in deriving numerical figures of merit of various kinds. Furthermore, if we add transitions corresponding to repair from each “Down” state to the “Up” state, we can quantify not only the risk of unsafe operation but also the chances that the backup (manual) system may be stretched beyond its capacity owing to overly long repair time.

Consider for example the more elaborate state model depicted in Fig. 2.10. Here, we model the fact that a safe failure condition may turn into an unsafe one if the situation is not handled properly and the possibility that the system can recover from a safe failure through repair.

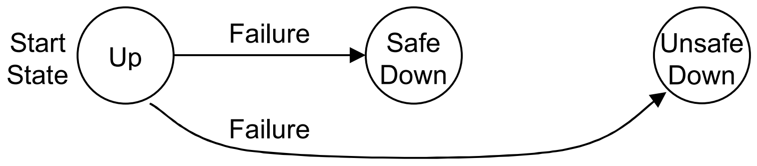


Fig. 2.9 **Three-state nonrepairable system with safe and unsafe failed states.**

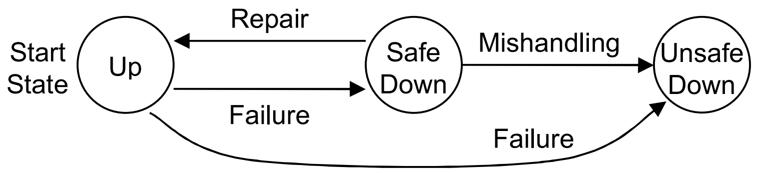


Fig. 2.10 **Three-state repairable system with safe and unsafe failed states.**

In both Figs. 2.9 and 2.10, one may use multiple unsafe failed states, one for each level of severity, say. In this way, the probabilities of ending up in the various unsafe states can be used for risk assessment using Eqn. (2.5.Risk2). State-space modeling techniques are discussed in Chapter 4.

2.6 Privacy and Security

In many application contexts, the bulk of impairments to dependability are human-related. Such factors include operator errors (e.g., due to carelessness or improper response to safety warnings) and malicious attacks (hackers, viruses, and the like). Desirable system attributes pertaining to the impairments just mentioned include privacy and security. Privacy is compromised, for example, when confidential or personal data are disclosed to unauthorized parties, either due to inadvertent error or as a consequence of malicious manipulation. Security is breached, for example, when account information for a bank customer is incorrectly modified owing to inadvertent error (such as modifying the account balance in an ATM cash withdrawal attempt, without dispensing the cash) or malicious action.

Despite several decades of research on privacy and security in computing systems, these two aspects have resisted quantitative assessment. In theory, security can be quantified in the same manner as safety, that is, by considering frequency or probability of a security breach as one factor, and magnitude or severity of the event as another. However, quantifying both factors is substantially more difficult in the case of security, compared with safety. One aspect of the difficulty pertains to the fact that security breaches are often not accidental, so they are ill-suited to a probabilistic treatment.

We end this section by noting that system security is orthogonal to both reliability and safety. A system that automatically locks up when a security breach is suspected may be deemed highly secure but it may not be very reliable or safe in the traditional sense.

Problems

2.1 Mean time to failure

A particular computer model has a constant failure rate λ. What percentage reduction in λ would lead to MTTF improvement by 25%? By 100%?

2.2 Calculating event probabilities

Let, in a class of n students, A be the event that no student is born in December and B be the event that at least two students have identical birthdays (only the day and month, not the year).

a. Which event, A or B, is more likely in a class with $n = 10$? Fully justify your answer.

b. For what value of n are the two events A and B roughly equiprobable?

c. Let an event be likely if its probability exceeds 0.9. For what values of n is event B likely?

2.3 The birthday paradox

Let in a class of n students, p be the probability that at least two students have identical birthdays (only the day and month, not the year). What is the smallest value of n for which $p > 1/2$? This is known as the birthday paradox, because the answer is much smaller than what one might think.

2.4 The Monty Hall problem

This problem is named after one-time host of the TV game show "Let's Make a Deal." Imagine that you are a contestant on this game show and there is a prize behind one of three doors with equal probabilities. You pick door A. The host opens door B to reveal that there is no prize behind it. He then gives you a chance to switch to door C, that is, to get the potential prize behind door C instead of door A. Is it better to make the switch or to stick to your original choice? *Hint:* A possible argument is that the prize is equally likely to be behind door A or door C, and this situation does not change by the opening of door B. A second argument is that when you chose door A, you picked the prize with probability 1/3 and missed it with probability 2/3. So, after opening door B, the probability that the prize is behind door C increases to 2/3.

2.5 Computing expected values

We roll five standard dice, with faces marked 1-6. With each roll, we record a number (not necessarily an integer) as follows. If a number appears more than any other (2, 3, 4, or 5 times), we record that number. If two numbers appear twice each, we record their average. If all five numbers are different, we record the median of the five values.

a. Argue informally that the expected value of the number recorded is 3.5.

b. Present a formal proof of the result of part a.

2.6 Interval availability

Plot the interval availability $A(t)$ as a function of t, $00{:}00 \le t \le 24{:}00$, for a two-state repairable system that fails at time 3:20, goes back into operation at time 3:35, fails again at time 18:30, becomes operational again a time 19:10, and remains operational until $t = 24{:}00$. Explain the shape of the curve obtained.

2.7 Mission time for a given reliability

Find the mission-time function $T(r) = R^{-1}(r)$ for as many of the distributions shown in Table 2.2 as possible.

2.8 Order statistics

We roll n standard dice, with faces marked 1-6, and record the largest number L and the smallest number S that appear. If we repeat this experiment many times, L and S can be viewed as random integer variables ranging in [1, 6]. Let $p_k[j] = \text{prob}\{k \text{ dice are rolled, and } L = j\}$ and $q_k[j] = \text{prob}\{k \text{ dice are rolled, and } L < j\}$. Clearly, $q_k[1] = 0$ and $\Sigma_{1 \le j \le 6}\, p_k[j] = 1$, for all k.

a. Prove that $p_n[j] = 6^{-n} + (j - 1)\, p_{n-1}[j] + (1/6)\, q_{n-1}[j]$.

b. Starting from $p_1[j] = 1/6$ and $q_1[j] = (j - 1)/6$, tabulate the values of $p_n[j]$ and $q_n[j]$ for $n \le 6$, using the equality of part a.

c. Calculate the expected value $E[L]$ for $n \le 6$ and plot it as a function of n.

d. Prove that $E[L] + E[S] = 7$, regardless of n.

2.9 Availability of a two-state system

A system never fails but is shut down for 30 minutes of preventive maintenance after every two hours of operation. The first two hours of operation begin at time 00:00.

a. Plot the availability of this system in the interval $[0, t]$, for $00{:}00 \le t \le 24{:}00$, as a function of t.

b. Derive an expression of the interval availability $A(t)$ as a function of t. *Hint:* The expression will involve "floor" or "ceiling" operations.

c. Show that the availability $A(t)$ of this system tends to 0.8 for large t.

2.10 System with two operational states

In the three-state system of Fig. 2.7, add a transition from the Up2 state to the Failed state. Supply an appropriate label for the new transition and explain why it makes sense to consider this transition. Does a reverse transition from the Failed state to the Up2 state also make sense?

2.11 Properties of expected value

Let x and y be random variables and let a and b denote constants. Prove the following results:

a. If $f(x)$ is symmetrical about a, that is, $f(a + x) = f(a - x)$, then $E[x] = a$

b. $E[x + y] \;=\; E[x] + E[y]$

c. $E[a \times x + b] \;=\; a \times E[x] + b$

2.12 Privacy and security

One of the most significant break-ins for a cellular telephone system was uncovered in Greece during early 2005. Read the article [Prev07] and use online sources describing the same incident to answer the following questions in a single typed page (single-spacing is okay) as completely as possible.

a. Write an abstract (200 words or less) that describes the indicent being reported.

b. The article's main theme are privacy and security. However, system dependability features (reliability, serviceability, maintainability) are also involved. Describe in 200 words or less the interplay between dependability features and the focal points of privacy and security.

c. Speculate on whether similar incidents could have happened in more technologically advanced countries within the same time frame (2004-05).

2.13 Interval failure probability

a. What is the probability that a system with reliability function $R(t)$ fails in the time interval $[a, b]$?

b. Consider the special case of a constant hazard rate λ, corresponding to an exponential reliability formula, and explain why the derived interval failure probability from part a is not $\lambda(b - a)$.

c. Consider a time c in the interval $[a, b]$. What is the conditional probability that the system of part a failed in the interval $[a, c]$, given the knowledge that it definitely failed in the interval $[a, b]$?

d. Discuss the special case of part c for a constant hazard rate λ.

2.14 Secure data storage

A real number (secret) must be shared among three people so that no one of them knows the number but any two can cooperate to discover what it is. Consider the following secret-sharing scheme. Each person i is given a pair of real numbers a_i and b_i, so that $a_ix + b_i$ defines a line in the x-y plane. The three lines intersect at a point whose x coordinate is the secret number [Blak79].

a. What is the informational redundancy in this scheme?

b. Does any of the three persons have partial information about the secret?

c. Will the scheme work just as well if the secret number is an integer?

d. Discuss the advantages and disadvantages of the following scheme for sharing two secret numbers. The scheme works as above, except that the y coordinate of the intersection point corresponds to the second secret number.

2.15 Probability concepts

During World War II, a hypothetical city laid out as a 10-by-10 grid of equal-size blocks was hit by 200 randomly dropped bombs. Thus, the probability of any particular bomb hitting a specific city block was 1/100 and each block was hit by an average of two bombs.

a. Find the probability of a particular city block not being hit by bombs at all.

b. Derive the expected number of city blocks hit by two or more bombs.

2.16 Probability concepts

A particular data file can be stored in any one of 10 locations, numbered 1 through 10. The probability of the data file being in location i in inversely proportional to i. In other words, the probability of the data file being in location 5 is twice that of it being in location 10. Assuming that the inspection of each location takes 1 time unit, what is the expected length of time needed to retrieve the data file? Does the expected value depend on the order in which we inspect the 10 locations?

2.17 Mean time to failure

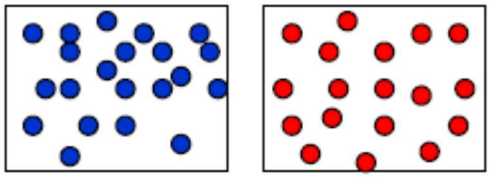
A particular computer model has the failure rate λ.

a. If a large number m of identical machines of this type run continuously for a period of time equal to their MTTF, how many are expected to be still working at the end? *Hint:* The answer isn't $m/2$.

b. By what time should we expect 90% of the machines to have failed?

2.18 Probability concepts

Which one of the two patterns below is more random? Explain your answer.



2.19 Probability concepts

a. Which is more probable at your home or office: a power failure or an Internet outage? Which is likely to last longer?

b. Which surgeon would you prefer for an operation that you must undergo: Surgeon A, who has performed some 500 operations of the same type, with 5 of his patients perishing during or immediately after surgery, or Surgeon B, who has a perfect record in 25 operations?

c. Which do you think is more likely: the event that every student in a class of 10 was born in the first half of the year or the event that at least two students were born on the same day of the year?

2.20 Mean time to failure

Using integration by parts, show that the definition of MTTF in eqn. (2.2.MTTF) and the second integral in that same equation are equivalent.

2.21 Coin-flipping puzzles

a. You are joining Coin Flippers of America and your dues will be decided by flipping a coin, until a 5-toss pattern of your choosing appears. For example, if you choose HHHHH and it takes 36 flips before the pattern appears, your annual dues will be $36. Should you choose HHHHH, HTHTH, or HHHTT? Does it even matter?

b. After joining Coin Flippers of America, you enter a tournament and face the first opponent. Each of you picks a different head-tail sequence of length 5, and a coin is flipped repeatedly. The player whose sequence appears first is the winner. What sequence would you choose if you were to go first?

2.22 Overbooking by airlines

JetBlack Airlines has determined that on average 5% of those making flight reservations do not show up. The company has thus decided to sell 78 tickets on each 75-passenger flight.

a. What is the probability that every passenger showing up will get a seat on such a flight?

b. How can the airline increase the probability of being able to accommodate all passengers to at least 90%?

2.23 Comparing event probabilities

A city has two hospitals with maternity wards: a large one, where an average of 64 babies are born per day, and a small one, with an average of 8 daily births. On average, half of the new arrivals in each hospital are boys and half are girls. One day, however, one of the hospitals had three times as many boys born as girls. In which hospital is this more likely to have occurred?

2.24 The birthday paradox extended

The birthday paradox (see Problem 2.3) tells us that with a fairly small number of randomly distributed birthdays, it is more likely than not to have 2 on the same day. With 75 birthdays, the odds of having 2 on the same day becomes 99.9%; that is, almost certain. These results are counter-intuitive, hence the designation “paradox.” Consider the corresponding result for having 3 birthdays on the same day. In other words, what is the smallest number of randomly distributed birthdays that would make it more likely than not to have 3 birthdays on the same day?

2.25 Amdahl’s reliability law

Express the main idea presented in [Parh15] in 200 or fewer words; that is, write an abstract for the paper.

2.26 Reliability inversion

The actual reliability of a highly-reliable system is unknowable, so designers try to obtain lower bounds on system reliability as part of the design evaluation process in order to assess whether the system is good enough for a particular application. If we have two systems with actual reliabilities R_1 and R_2 and reliability lower bounds r_1 and r_2, with $r_1 < r_2$, we cannot deduce that $R_1 < R_2$. The condition $r_1 < r_2$ and $R_1 > R_2$ is known as reliability inversion [Parh20]. Why is this a problem and what can we do about it?

2.27 Trustworthiness of AI systems

Read the paper [Wing 21] and, based on what you learn from it, add at least two terms to the diagram in Slide 56 of our textbook's Part 1 (which lists the -ilities, plus safety, robustness, and so on). Explain your choice of the terms and why they are important. Why do you think the application of formal systems to AI faces additional challenges?

2.28 Modeling the cost of human fatalities and injuries

Study the reference [USDT13] and prepare a one-page, single-spaced summary of its most-important points using a one-paragraph introduction, followed by bullet points.

References and Further Readings

[Ande85] Anderson, T. A. (ed.), *Resilient Computing Systems*, Collins, London, 1985. Also: Wiley, New York, 1986.

[Andr02] Andrews, J. D. and T. R. Moss, *Reliability and Risk Assessment*, American Society of Mechanical Engineering, 2nd ed., 2002.

[Bent99] Bentley, J. P., *Reliability and Quality Engineering*, Addison-Wesley, 2nd ed., 1999.

[Blak79] Blakley, G., "Safeguarding Cryptographic Keys," *Proc. AFIPS Nat'l Computer Conf.*, 1979, pp. 313-317.

[Cart64] Carter, W. C., H. C. Montgomery, R. J. Preiss, and H. J. Reinheimer, "Design of Serviceability Features for the IBM System/360," *IBM J. Research and Development*, Vol. 8, No. 2, pp. 115-126, April 1964.

[USDT13] US Department of Transportation, "Treatment of the Value of Preventing Fatalities and Injuries in Preparing Economic Analysis," Revised Guidance, 2013. On-line document: https://www.transportation.gov/sites/dot.dev/files/docs/VSL%20Guidance_2013.pdf

[Gold79] Goldstein, L. H., "Controllability/Observability Analysis of Digital Circuits", *IEEE Trans. Circuits and Systems*, Vol. 26, No. 9, pp. 685-693, September 1979.

[Henl81] Henley, E. J. and H. Kumamoto, *Reliability Engineering and Risk Assessment*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Leve86] Leveson, N. G., "Software Safety: Why, What, and How?" *Computing Surveys*, Vol. 18, No. 2, pp. 125-163, June 1986.

[Levi15] Levitin, G., L. Xing, B. W. Johnson, and Y. Dai, "Mission Reliability, Cost and Time for Cold Standby Computing Systems with Periodic Backup," *IEEE Trans. Computers*, Vol. 64, No. 4, pp. 1043-1057, April 2015.

[Lewi87] Lewis, E. E., *Introduction to Reliability Engineering*, Wiley, New York, 1987.

[Nguy16] Nguyen, T. A., D. S. Kim, and J. S. Park, "Availability Modeling and Analysis of a Data Center for Disaster Tolerance," *Future Generation Computer Systems*, Vol. 56, pp. 27-50, March 2016.

[Papo90] Papoulis, A., *Probability & Statistics*, Prentice Hall, 1990.

[Parh15] Parhami, B., "Amdahl's Reliability Law: A Simple Quantification of the Weakest-Link Phenomenon," *IEEE Computer*, Vol. 48, No. 7, pp. 55-58, July 2015.

[Parh20] Parhami, B., "Reliability Inversion: A Cautionary Tale," *IEEE Computer*, Vol. 53, No. 6, pp. 28-33, June 2020.

[Prev07] Prevelakis, V. and D. Spinellis, "The Athens Affair: How Some Extremely Smart Hackers Pulled off the Most Audacious Cell-Network Break-in Ever," *IEEE Spectrum*, Vol. 44, No. 7, pp. 26-33, July 2007.

[Ross72] Ross, S. M., *Introduction to Probability Models*, Academic Press, New York, 1972.

[Sahn96] Sahner, R. A., K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*, Kluwer, 1996.

[Shoo02] Shooman, M. L., *Reliability of Computer Systems and Networks*, Wiley, New York, 2002.

[Siew92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 2nd ed., 1992.

[Tobi86] Tobias, P. A. and D. C. Trindade, *Applied Reliability*, Van Nostrand Reinhold, New York, 1986.

[Wing21] Wing, J. M., "Trustworthy AI," *Communications of the ACM*, Vol. 64, No. 10, pp. 64-71, October 2021.

3 Combinational Modeling

"Torture numbers, and they'll confess to anything."

Gregg Easterbrook

"Doubt is not a pleasant condition, but certainty is absurd."

Voltaire

"No papers for this session will be published. The purpose of this is to permit the speakers to be very candid regarding the various computer disasters which they are describing."

From "abstract" of the session on "Anatomies of Computer Disasters" in Proc. First Int'l Conf. Computing in Civil Engineering, 1981

Topics in This Chapter

Combinational, or stateless, models allow various system-level dependability parameters to be calculated from the relevant parameters of the component parts or subsystems that comprise the system. In this chapter, we introduce a number of basic tools or building blocks for dependability evaluation: case analysis, series systems, parallel systems, and *k*-out-of-*n* systems. We then apply and extend these tools to the task of dependability analysis by means of three widely applicable graphical representations for system dependability modeling. We start with the simple and intuitive reliability block diagrams and end with the widely used fault trees, covering the lesser-known reliability graphs in between.

3.1 Modeling by Case Analysis

Given a set of components, subsystems, or other parts that comprise a system, one can determine the probability of the system being operational by enumerating all possible combinations of good and bad parts that result (or do not result) in system failure. The overall probability of these subcases is the system unreliability (reliability). This method works well when the number of parts is fairly small and their interactions and mutual influences are well understood.

Example 3.1: Reliability modeling of a multiprocessor A multiprocessor system consists of two processors, a common bus, and four memory modules. Given reliabilities for each of the three component types, what is the system reliability, assuming that at least one processor and one memory module, connected by the common bus, are needed for system operation.

Solution: The common bus is a critical system part. The system fails if the bus, both processors, or all four memory modules malfunction. This is the same as saying that the system functions properly if the bus, one of the two processors, and one of the four memory modules work. This has the probability $R = r_b[1 - (1 - r_p)^2][1 - (1 - r_m)^4]$.

Example 3.1 was simple enough to allow us to write the pertinent reliability equation directly. We now illustrate how case analysis can be used to reduce the complexity of reliability evaluation using the same simple example.

First consider two cases: the bus system works (probability r_b) or it does not work (probability $1 - r_b$). We begin constructing a tree, as in Fig. 3.1, where the root labeled "No information" has two children, labeled "Bus okay" and "Bus bad." If we are interested in enumerating the operational system configurations, we can ignore the subtree corresponding to "Bus bad." We next focus on the processors and form three subtrees for the "Bus okay" branch, labeling them "Both processors okay" (probability r_p^2), "One processor okay" (probability $2r_p(1 - r_p)$), and "Both processors bad" (probability $(1 - r_p)^2$). We can merge the first two of these branches and assigning the resulting "At least one processor okay" branch the probability $2r_p - r_p^2$, because the two are identical with respect to the proper functioning of the system. Continuing in this manner, we arrive at all possible leaf nodes associated with working configurations. Adding the probabilities of these leaf nodes yields the overall system reliability. We can stop expanding each branch as soon as the reliability equation for the corresponding state can be written directly.

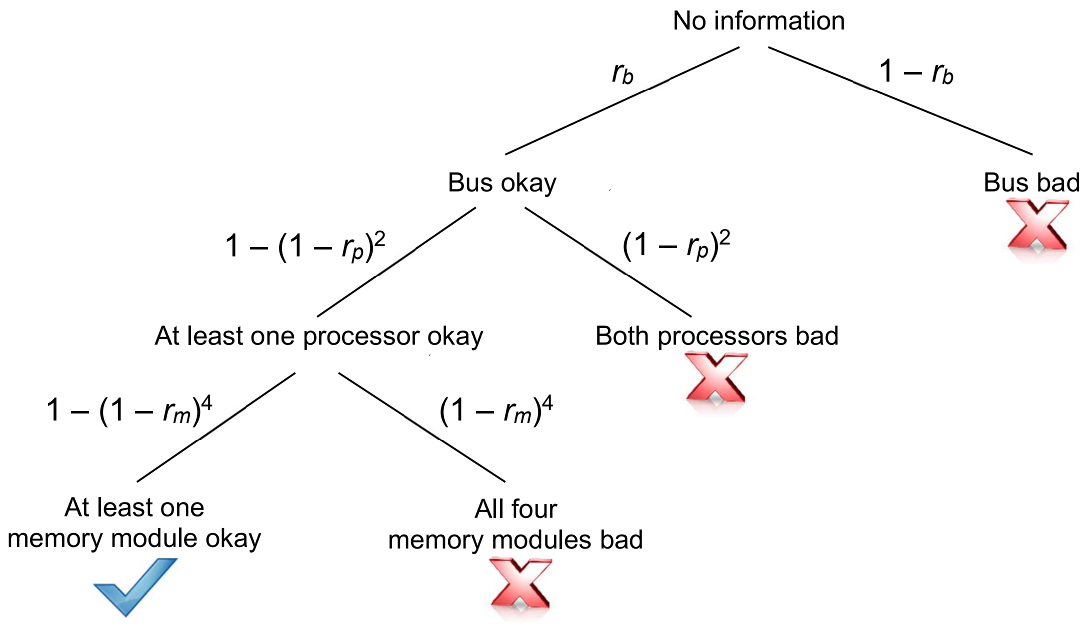


Fig. 3.1 Example of reliability evaluation by case analysis.

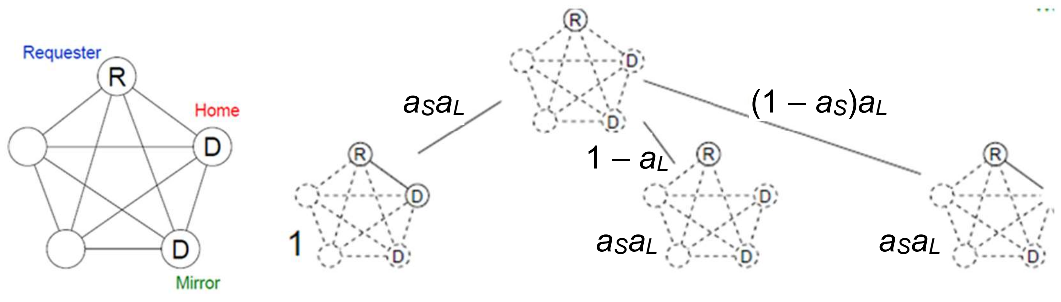
We further illustrate the method of case analysis with two additional examples.

Example 3.2: Data availability modeling with home and mirror sites Use case analysis to derive the availability of data for the system in Example 1.1.

Solution: The required case-analysis tree is depicted in Fig. 3.2, leading to the availability equation $A = a_S a_L + (1 - a_S a_L) a_S a_L = 2a_S a_L - (a_S a_L)^2$.

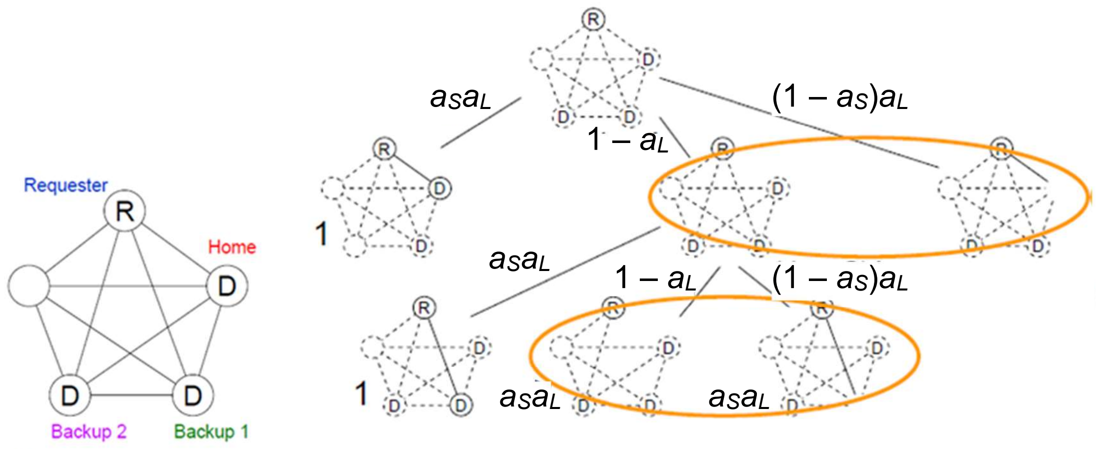
Example 3.3: Data availability modeling with triplication Use case analysis to derive the availability of data for the system in Example 1.2.

Solution: The required case-analysis tree is depicted in Fig. 3.3, leading to the availability equation $A = a_S a_L + (1 - a_S a_L) a_S a_L + (1 - a_S a_L)^2 a_S a_L = 3a_S a_L - 3(a_S a_L)^2 + (a_S a_L)^3$.



(a) System configuration (b) Case analysis tree

Fig. 3.2 Case analysis used to derive the data availability equation for Example 1.1 (home and mirror sites).

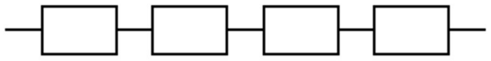


(a) System configuration (b) Case analysis tree

Fig. 3.3 Case analysis used to derive the data availability equation for Example 1.2 (home site and two backups).

3.2 Series and Parallel Systems

A *series system* of *n* components is one in which the proper operation of each of the *n* components is required for the system to perform its function. Such a system is represented as in Fig. 3.4a, with each rectangular block denoting one of the subsystems.



(a) Block diagram

(b) Example with valves

Fig. 3.4 Series system block diagram and example with valves that are prone to stuck-on-shut failures.

Given the reliability R_i for the ith component of a series system, the overall reliability of the system, assuming independence of failures in the subsystems, is given by:

$$R = \prod_{i=1}^{n} R_i \qquad (3.2.\text{ser1})$$

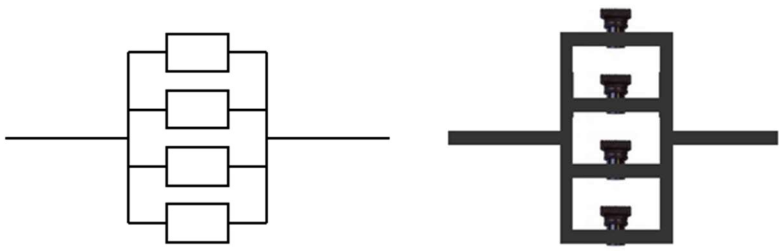
For example, if we place four valves in tandem on a segment of a pipe connected to a reservoir (Fig. 3.4b), with the component valves being prone to stuck-on-shut failures, we have a series system. Note that the term “series” in series system does not imply that the subsystems are physically connected in series in the mechanical or electrical sense. If our valves were prone so stuck-on-open failures only, then a four-unit series system would actually consist of the valves being connected in parallel in the mechanical sense. In a four-way system of parallel valves, the stuck-on-open failure of any one of the valves will cause a stuck-on-open failure at the system level, thus with parallel valves that only fail in the stuck-on-open mode, we have a series system in the reliability theoretic sense.

If the ith component in a series system has a constant hazard rate λ_i, thus having exponential reliability, the overall system will have exponential reliability with the hazard rate $\Sigma\lambda_i$. This is a direct consequence of equation (3.2.ser). With repairable components, having hazard rate λ_i and repair rate μ_i, the availability of a series system of *n* components is related to the individual module availabilities $A_i = \mu_i / (\lambda_i + \mu_i)$ by:

$$A = \prod_{i=1}^{n} A_i \qquad (3.2.\text{ser2})$$

Equation (3.2.ser2) is valid when component failures/repairs are independent and we have a separate repairperson or facility for each unit; in other words, concurrent failure of multiple units does not slow down the repair of any one unit.

A *parallel system* of *n* components is one in which the proper operation of a single one of the *n* components is sufficient for the system to perform its function. Such a system is represented as in Fig. 3.5a, with each rectangular block denoting one of the subsystems.



(a) Block diagram (b) Example with valves

Fig. 3.5 **Parallel system block diagram and example with valves that are prone to stuck-on-shut failures.**

Given the reliability R_i for the *i*th component of a parallel system, the overall reliability of the system, assuming independence of failures in the subsystems, is given by:

$$R = 1 - \prod_{i=1}^{n}(1 - R_i) \qquad (3.2.\text{par1})$$

For example, if placing a valve on each of four branches of a pipe (Fig. 3.5b), with the component valves being prone to stuck-on-shut failures, yields a parallel system; we can still control access to the reservoir even if three of the valves fail in the stuck-on-shut mode. Again, the term “parallel” in parallel system does not imply that the subsystems are physically connected in parallel in the mechanical or electrical sense.

If the components in a parallel system are repairable, with the ith component having a hazard rate λ_i and repair rate μ_i, the availability of a parallel system of *n* components is related to the individual module availabilities $A_i = \mu_i / (\lambda_i + \mu_i)$ by:

$$A = 1 - \prod_{i=1}^{n}(1 - A_i) \qquad (3.2.\text{par2})$$

Equation (3.2.par2) is valid when component failures/repairs are independent and we have a separate repairperson or facility for each unit; in other words, concurrent failure of multiple units does not slow down the repair of any one unit.

Reliability and availability equations for series and parallel systems are quite simple. This does not mean, however, that proper application of these equations does not require careful thinking. The following example illustrates that care must be exercised in performing even simple dependability analyses.

Example 3.4: A two-way parallel system In a passenger plane, the failure rate of the cabin pressurizing system is 10^{-5}/hr and the failure rate of the oxygen-mask deployment system is also 10^{-5}/hr. What is the probability of loss of life due to both systems failing during a 10-hour flight?

Possible solution 1: Given the assumption of failure independence, both systems fail together at a rate of 10^{-10}/hr. Thus, fatality probability for a 10-hour flight is $10^{-10} \times 10 = 10^{-9}$. Fatality odds of 1 in a billion or less are generally deemed acceptable in safety-critical systems.

Possible solution 2: The probability of the cabin-pressurizing system failing during a 10-hour flight is 10^{-4}. The probability of the oxygen-mask system failing during the flight is also 10^{-4}. Given the assumption of independence, the probability of both systems failing together during the flight is 10^{-8}. This latter probability is higher than acceptable norms for safety-critical systems.

Analysis: So, which of the two solutions is correct? Neither one. Here's why. When we multiply the two per-hour failure rates and then take the flight duration into account, we are assuming that only the failure of the two systems within the same hour is catastrophic. This produces the optimistic reliability estimate $1 - 10^{-9}$. When we multiply the two flight-long failure rates, we are assuming that the failures of both systems would be catastrophic, no matter when each occurs during the flight. This produces the pessimistic reliability estimate $1 - 10^{-8}$. The reader should be able to supply examples of when the two systems fail at different times during a flight, without leading to a catastrophe.

The simple reliability equation (3.2.par1) for a parallel system is based on the assumption that all *n* subsystems contribute to the proper system functioning at the same time, and each is capable of performing the entire job, so that the failure of up to $n - 1$ of the *n* subsystems will be noncritical. This happens, for example, if we have *n* concurrently operating ventilation systems in a lab, each with its own power supply, in order to ensure the proper removal of hazardous fumes. If the capacity of one of the subsystems is inadequate and we need at least two of them to perform the job, we no longer have a parallel system, but a 2-out-of-*n* system (see Section 3.3). Similarly, if only one of the subsystems is active at any given time, with the others activated in turn upon detection of a failure, then equation (3.2.par1) is valid only if failure detection is perfect and instantaneous, and the activation of spares is always successful.

The simplest way to account for imperfect failure detection and activation of spares in a parallel system is via the use of a *coverage* parameter c, with $c < 1$. The coverage parameter is defined as the probability that the switchover from an operating module to a spare module goes without a hitch. Thus, in a two-unit parallel system in which the primary module has reliability r_1 and the spare has reliability r_2, the system reliability is:

$$R = r_1 + (1 - r_1)cr_2 \qquad (3.2.\text{cov1})$$

Equation (3.2.cov1) essentially tells us that the two-way parallel system with imperfect coverage will work if unit 1 works, or if unit 1 fails, but the switchover is successful and unit 2 works. With modules having identical reliability r, equation (3.2.cov1) becomes:

$$R = r[1 + c(1 - r)] = r\,\frac{1-c^2(1-r)^2}{1-c(1-r)} \qquad (3.2.\text{cov2})$$

The rightmost expression in equation (3.2.cov2) allows us to generalize the reliability equation to the case of an n-way parallel system with imperfect coverage c:

$$R = r\,\frac{1-c^n(1-r)^n}{1-c(1-r)} \qquad (3.2.\text{cov3})$$

Deriving equation (3.2.cov3) is left as an exercise. The crucial impact of coverage on system reliability is evident from Fig. 3.6, assuming a module reliability of $r = 0.95$.

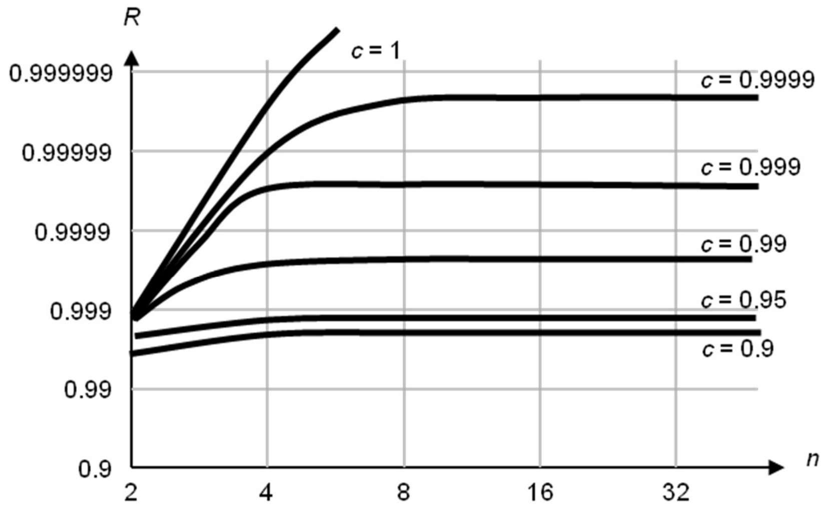


Fig. 3.6 Adding spares to a parallel system is unhelpful in the absence of good coverage *c*. Module reliability is *r* = 0.95.

Note that, in practice, the coverage factor is not a constant, but deteriorates with more spares. In this case the depiction of the effect of coverage in Fig. 3.6 may be viewed as optimistic. So, adding a large number of spares is not only unhelpful (as suggested by the saturation effect in Fig. 3.6), but it may actually be detrimental to system reliability.

3.3 Classes of *k*-out-of-*n* Systems

In a *k*-out-of-*n* system, there are *n* modules, the proper functioning of any *k* of which is sufficient for system operation. Note that both series (*n*-out-of-*n*) and parallel (1-out-of-*n*) systems are special cases of this more general class. For example, if you have one spare tire in the trunk of your car, then, ignoring the possible difference between a spare tire and a regular tire, your tire system is a 4-out-of-5 system. You can continue driving your car as long as at most one tire malfunctions, assuming successful switchover from a regular tire to the spare tire. If you carry two spare tires in your trunk, then your tire system may be described as a 4-out-of-6 system.

One of the most commonly used systems of this type is a 2-out-of-3 system, depicted in Fig. 3.7. This redundancy scheme is known as *triple modular redundancy* (TMR) and relies on a decision circuit, or voter, to deduce the correct output based on the outputs it receives from three concurrently operating modules.

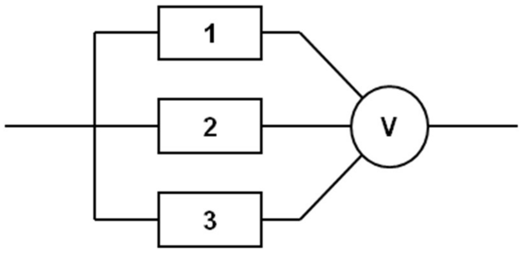


Fig. 3.7 Triple modular redundancy with voting.

Assuming a perfect (always working) voting unit, the reliability of a TMR system with module reliabilities r_1, r_2, and r_3 is:

$$R = r_1r_2r_3 + r_1r_2(1 - r_3) + r_2r_3(1 - r_1) + r_3r_1(1 - r_2) \qquad (3.2.\text{TMR1})$$

In the special case of identical modules of reliability r, equation (3.2.TMR1) becomes $R = 3r^2 - 2r^3$. Accounting for an imperfect voting unit with reliability r_v, a TMR system with identical modules has reliability:

$$R = r_v(3r^2 - 2r^3) \qquad (3.2.\text{TMR2})$$

Assuming exponential reliability $r = e^{-\lambda t}$ for each of the three modules and taking the voter to be perfectly reliable, the MTTF parameter of a TMR system can be obtained based on eqns. (2.2.MTTF) and (3.2.TMR2) with $r_\mathrm{v} = 1$:

$$\mathrm{MTTF}_\mathrm{TMR} = = \int_0^\infty R(t)dt = \int_0^\infty [3e^{-2\lambda t} - 2e^{-3\lambda t}]dt = 5/(6\lambda) \qquad (3.2.\mathrm{MTTF})$$

Note that even though the reliability of a TMR system is greater than that of a single module, its MTTF deteriorates from 1/λ to 5/(6λ).

The reliability equation for a k-out-of-n system with an imperfect voting unit and identical modules is:

$$R = r_\mathrm{v}\left[\sum_{j=k}^{n} \binom{n}{j} r^j (1-r)^{n-j}\right] \qquad (3.2.\mathrm{kofn})$$

In the special case of odd n with $k = (n + 1)/2$, the k-out-of-n scheme uses *majority voting* and is sometimes referred to as *n-modular redundancy* (NMR). It is readily seen from equations (3.2.TMR2) and (3.2.kofn) that TMR and NMR methods lead to significant reliability improvement only if the voting unit is much more reliable than the modules performing the system functions.

A key element in the application of k-out-of-n redundancy schemes, and their special cases of majority voting, is the design of appropriate "voting" circuits. Considerations in the design of voting circuits are discussed in Chapter 12.

Replicating the voters and performing the entire computation in three separate and independent channels is one way of removing the voting circuits from the critical system core. Figure 3.8 shows how voter triplication in a TMR system will allow voter failures as well as module failures to be tolerated. As the oval dashed boxes indicate, the voter reliability can be lumped with module reliability, instead of it appearing separately, as in equations (3.2.TMR2) and (3.2.kofn).

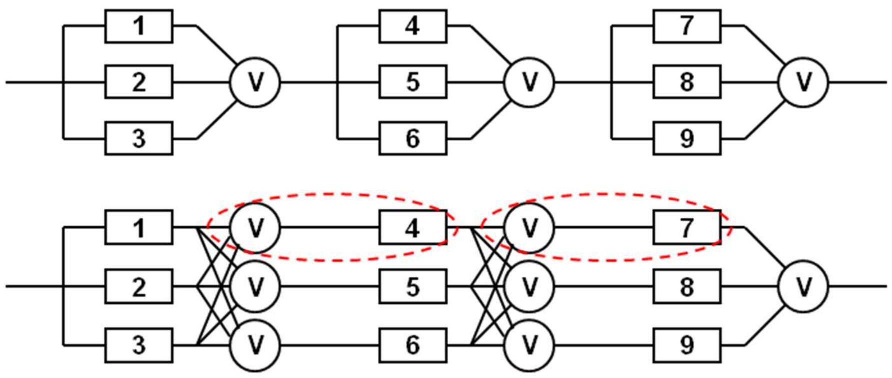


Fig. 3.8 TMR system with nonreplicated and replicated voting units.

Note that in writing the reliability equation (3.2.TMR1) for a TMR system, we have pessimistically assumed that any two module failures will render the system useless. This is not always the case. For example, if the modules produce single-bit outputs, then when the output of one module is stuck-on-1, while a second module's output is stuck-on-0, the system can still produce the correct output, despite the occurrence of double module failures. Such *compensating failures*, as well as situations where problems are detected because the multiple modules produce distinct erroneous results, leading to a lack of majority agreement, are discussed in Chapter 12.

Two variants of k-out-of-n systems also merit discussion, although they are not in common use for modeling computer systems. We first note that the type of k-out-of-n system we have covered thus far can be called k-out-of-n:G system, with the added qualifier "G" designating that the system is "good" when at least k of its n modules are good. We may also encounter k-out-of-n:F systems, in which the failure of any k or more subsystems is tantamount to system failure. Clearly, a k-out-of-n:F system is identical to an $(n - k + 1)$-out-of-n:G system. So, the new notation is unnecessary for the type of systems we have been discussing.

A consecutive k-out-of-n:G system is one in which the n modules are linearly ordered, say, by indexing them from 1 to n, with the failure of any k consecutive modules causing system failure. So, for example, such a system may not be able to function with exactly k working modules, unless these k modules happen to be consecutive.

Example 3.5: Consecutive 3-out-of-5:G system Three values can be transmitted over three of five buses, using shift switches at the interface, as depicted in Fig. 3.9. Shift switches are controlled using a common set of control signals that puts all of them in the upward, straight, or downward connection state. Such a reconfiguration scheme is easier and less costly to implement than arbitrary (crossbar) connectivity and is thus quite popular.

Solution: The reliability of this system is different from an ordinary 3-out-of-5 system, because, for example, the outage of the middle bus line is not tolerated, even if it is the only one. Let each bus line have reliability r and assume that the switches are perfect. The system works when all 5 bus lines are okay, or any 4 are okay, except if the middle bus line is the bad one (4 cases in all), or if the set of 3 bus lines {1, 2, 3}, {2, 3, 4}, or {3, 4, 5} are good, with the remaining 2 being bad. Adding the three terms corresponding to the cases above, we get the system reliability equation: $R = r^5 + 4r^4(1 - r) + 3r^3(1 - r)^2 = 3r^3 - 2r^4$.

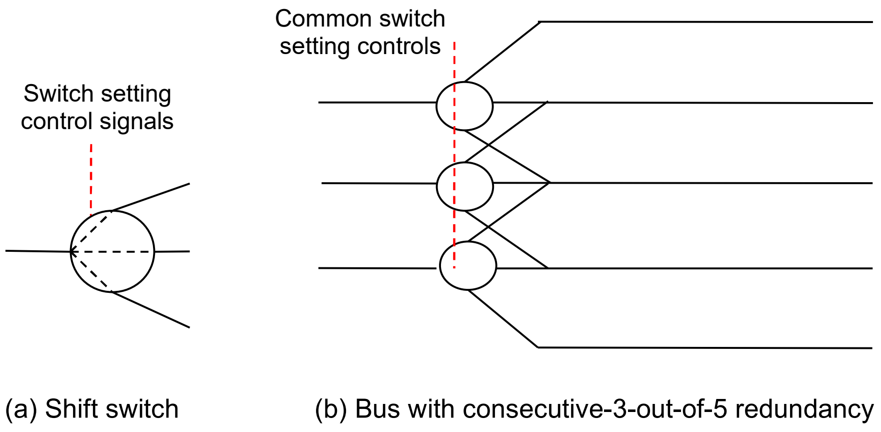


Fig. 3.9 **A consecutive 3-out-of-5:G system.**

The second variant is the class of consecutive k-out-of-n:F systems. Here, any k or more consecutive failed modules render the system nonfunctional. So, such a system may tolerate more than k module failures, provided the failed modules are not consecutive.

Example 3.6: Consecutive 2-out-of-*n*:F system Consider a system of n street lights, where the lights provide a minimum level of illumination deemed adequate for safety, unless two or more consecutive lights are out. What is the reliability of this consecutive 2-out-of-n:F system?

Solution: The reliability of this system is different from an ordinary (n – 1)-out-of-n system, because, for example, the safety criterion is met even if every other light is out. Let each street light have reliability r. Let $f(n)$ be the reliability for a consecutive 2-out-of-n:F system. Then, we can write $f(n) = r\,f(n-1) + r(r-1)\,f(n-2)$, with $f(1) = 1$ and $f(2) = 2r - r^2$. The two terms in the equation for $f(n)$ correspond to the two possible cases for the first street light. If that light is working, then the system will be okay if the remaining $n - 1$ lights do not suffer 2 consecutive outages. Otherwise, if the first light is out, then the second light must be working, and the remaining $n - 2$ lights should not have 2 consecutive outages. The recurrence and its associated initial conditions allow us to compute $f(n)$ for any value of n, either numerically for a given value of r or symbolically for arbitrary r. For example, we find $f(5) = r^2 + 3r^3 - 4r^4 + r^5$.

In some consecutive k-out-of-n:G or k-out-of-n:F systems, the module indexing is considered circular, rather than linear. In this case, modules n and 1 are viewed as being consecutive, so additional modes for the system being operational (k-out-of-n:G) or failing (k-out-of-n:F) become possible.

3.4 Reliability Block Diagrams

A reliability block diagram (RBD) is simply a combination of modules connected in series or parallel forms. The example RBD in Fig. 3.11 may represent a multiprocessor, with A, F, and G being critical resources (such as bus, shared memory, and power supply), and B-D and C-E being two separate processors with their associated memory modules. Or the same diagram may represent an office with three critical employees and four clerks that can pair up in a particular way to perform the tasks required of them, with one pair (B-D or C-E) being adequate for the expected functions. The latter example is quite useful for making the point that an RBD does not represent electrical or mechanical linking of modules, but rather their interactions in terms of system reliability.

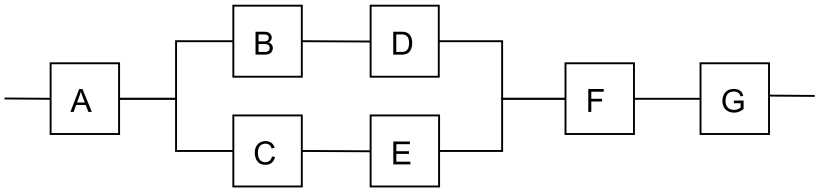


Fig. 3.11 Example reliability block diagram.

A reliability block diagram is best understood in terms of its *success paths*. A success path is simply a path through the modules, that leads from one side of the diagram to the other. In the case of Fig. 3.11, the success paths are A-B-D-F-G and A-C-E-F-G. By definition, the system modeled by a reliability block diagram is functional if all the modules on at least one success path are functional.

The reliability equation corresponding to an RBD can be easily derived by applying the series and parallel reliability equations (3.2.ser1) and (3.2.par1). In the case of the RBD in Fig. 3.11, using r_X to denote the reliability of module X, we have:

$$R = r_A [1 - (1 - r_B r_D)(1 - r_C r_E)] r_F r_G \qquad (3.4.RBD1)$$

If all modules in Fig. 3.11 have the same reliability r, equation (3.4.RBD1) reduces to $R = r^5(2 - r^2)$. Note that because $2 - r^2 > 1$, the system modeled is more reliable than a series system with five identical modules, as one would expect.

Example 3.7: Parallel-series and series-parallel systems Denoting the reliability of module j in Fig. 3.12 as r_j:

a. Derive the reliability equation for the parallel-series system of Fig. 3.12a.
b. Derive the reliability equation for the series-parallel system of Fig. 3.12b.
c. Compare the reliability expressions derived in parts a and b and discuss.

Solution: For parts a and b, we use equations (3.2.ser1) and (3.2.par1) in turn.

a. $R_a = 1 - (1 - r_1 r_2)(1 - r_3 r_4)$
b. $R_b = [1 - (1 - r_1)(1 - r_3)]\ [1 - (1 - r_2)(1 - r_4)]$
c. After some simple algebraic manipulation, the difference of the reliabilities for parts a and b is found to be $R_b - R_a = r_1 r_4(1 - r_2)(1 - r_3) + r_2 r_3(1 - r_1)(1 - r_4)$. Because the difference is always positive, the series-parallel configuration of Fig. 3.12b always offers better reliability compared with the parallel-series arrangement of Fig. 3.12a. We should have been able to predict this advantage, which is precisely due to Fig. 3.12b surviving when modules 1 and 4 are operational, while modules 2 and 3 have failed, or vice versa.

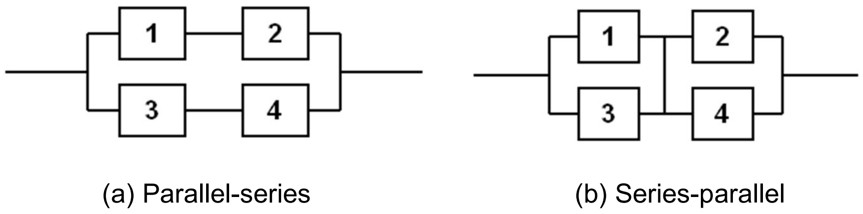


Fig. 3.12 **Parallel-series and series-parallel example reliability block diagrams.**

The basic series-parallel RBDs discussed thus far can be extended in several different ways. One way of extending RBDs is to allow k-out-of-n structures in addition to n-out-of-n (series) and 1-out-of-n (parallel) constructs. Such a structure is drawn as a set of parallel blocks with a suitable notation indicating that k out of the n blocks must be functional. This can take the form of an annotation next to the blocks, or a voter-like connector on the right-hand side on the parallel group into which the label "k of n" or "k / n" is inscribed. The use of such k-out-of-n structures does not complicate the derivation of the reliability equation: we simply use equation (3.3.kofn) in this case, in lieu of equation (3.2.par1).

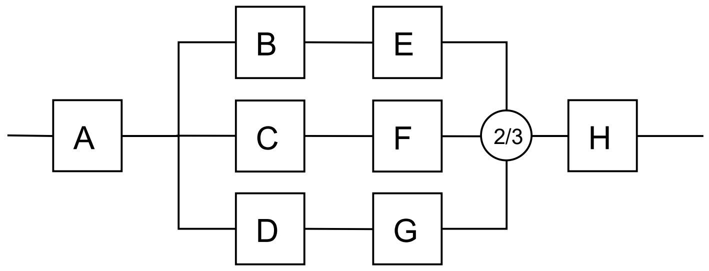


Fig. 3.13 Example of extended RBDs, with *k*-out-of-*n* structures.

A second way to extend RBDs is to allow connectivity patterns that are more general than series-parallel. For example, the bridge pattern of Fig. 3.14 would constitute such an extended RBD. In this example, one may view module 5 is being capable of replacing modules 2 and 3 when the latter interact with modules 1 and 4 (but not when module 3 should cooperate with module 6).

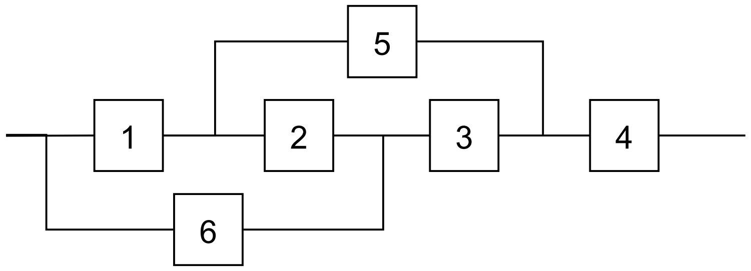


Fig. 3.14 Example of an extended RBD, allowing more general connectivity patterns than series-parallel.

A third way to extend RBDs is to allow repeated blocks [Misr70]. Figure 3.15 depicts two ways of representing a 2-out-of-3 structure, using parallel-series and series-parallel connection of blocks A, B, and C, with repetition.

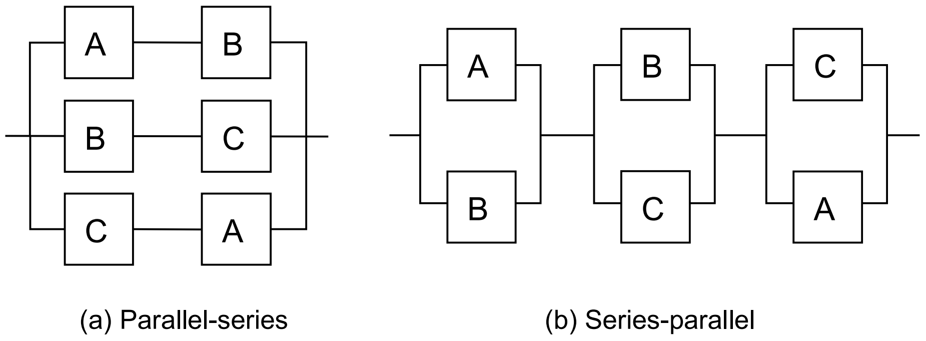


Fig. 3.15 **Two ways of representing a 2-out-of-3 structure by means of repeated modules.**

When RBDs are not of the simple series/parallel variety or when they have repeated elements, special methods are required for their analysis. The following two examples demonstrate the method.

Example 3.8: Non-series/parallel RBDs Consider the extended RBD in Fig. 3.14 and denote the reliability of module *i* by r_i. Derive an expression for the overall system reliability.

Solution: The system functions properly if a string of healthy modules connect one side of the diagram to the other. Let's consider two cases based on the status of module 3. If module 3 works (replacing it with a line directly connecting modules 2 and 6 to module 4) or does not work (disconnecting modules 2 and 6 from module 4). We thus get the system reliability equation $R = r_3R_{3\text{good}} + (1 - r_3)R_{3\text{bad}}$, where $R_{3\text{good}}$ and $R_{3\text{bad}}$ are conditional reliabilities for the two cases just mentioned. Our solution is complete upon noting that $R_{3\text{bad}} = r_1r_5r_4$ and, after a few steps of series/parallel combinations, $R_{3\text{good}} = [1 - [1 - r_1(1 - (1 - r_2)(1 - r_5))](1 - r_6)]r_4$.
Note: If we do case analysis based on the status of module 2, we would have to take care that when module 2 is good, the unallowed success path (M6, M5, M4) is not erroneously introduced.

Example 3.9: Extended RBDs with repeated elements Consider an extended RBD that is 3-way parallel, with each of the parallel branches being a series connection, as follows: (a) 1-5-4, (b) 1-2-3-4, and (c) 6-3-4. Boxes with the same number denote a common module. So, for example, the two occurrences of 1 in the diagram represent a common module 1. This RBD may be viewed as equivalent to that in Fig.3.14, in that it has the same success paths. So the analysis of this example is another way of solving Example 3.8. Derive a reliability expression for this RBD.

Solution: The inequality $R \le 1 - \Pi_i(1 - R_{i\text{th success path}})$ provides an upper bound on system reliability. The reason that the expression on the right-hand side represents an upper bound rather than an exact value is that is takes multiple occurrences of the same module as having independent failures. In the case of our example, we get $R \le 1 - (1 - r_1r_5r_4)(1 - r_1r_2r_3r_4)(1 - r_6r_3r_4)$. It turns out that if we multiply out the parenthesized terms on right-hand side of the foregoing inequality, but do this by ignoring the higher powers of each reliability term, an exact reliability formula results. For our example, the process just outlined yields $R = r_3r_4r_6 + r_1r_2r_3r_4 - r_1r_2r_3r_4r_6 + r_1r_4r_5 - r_1r_3r_4r_5r_6 - r_1r_2r_3r_4r_5 + r_1r_2r_3r_4r_5r_6$.

Thus far, we have taken the modules in an RBD to be independent of each other. More sophisticated models take the possibility of common-cause failures into account or allow the failure of some modules to affect the proper functioning of others, perhaps after a randomly variable amount of time [Levi13].

3.5 Reliability Graphs

A reliability graph (RG) is a schematic representation of system components, their interactions, and their roles in proper system operation in a manner that is more general than RBDs. An RG is an acyclic directed graph, with edges corresponding to system components. There are unique source and sink nodes, typically drawn on the left side and right side of the diagram, respectively, with a directed path from source to sink defining a success path. As the names imply, a source node has no incoming edges, while a sink node has no outgoing edges. Some edges are labeled "∞" and correspond to hypothetical modules that are infinitely reliable.

Figure 3.16 depicts an example reliability graph having success paths A-E-H-L-N, B-D-G-M, and C-F-H-K-M, among others. A reliability graph can be analyzed by converting it to a number of series/parallel structures through case analysis.

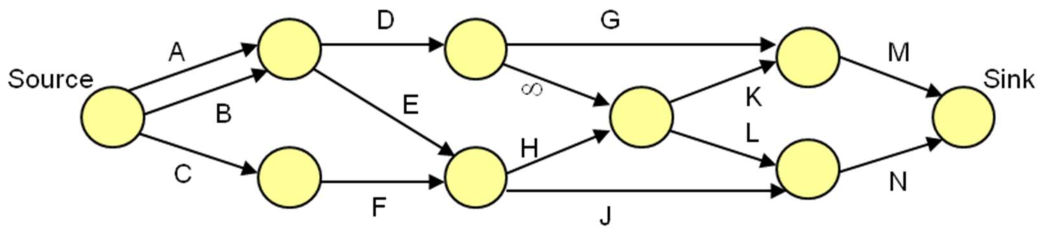


Fig. 3.16 Example of a reliability graph.

Reliability graphs are more powerful than simple series/parallel RBDs in terms of representational power, but they are less powerful than the most general form of fault trees, to be described next.

3.6 The Fault-Tree Method

Fault tree is a tool for top-down reliability analysis. To construct a fault tree, we start at the top with an undesirable event called a “top event” and determine all the possible ways in which the top event can occur. The fault-tree method allows us to determine, in a systematic fashion, how the top event can be caused by individual or combined lower-level undesirable events. Figure 3.17 contains an informal description for the building process as well aa some of the pertiment symbols used.

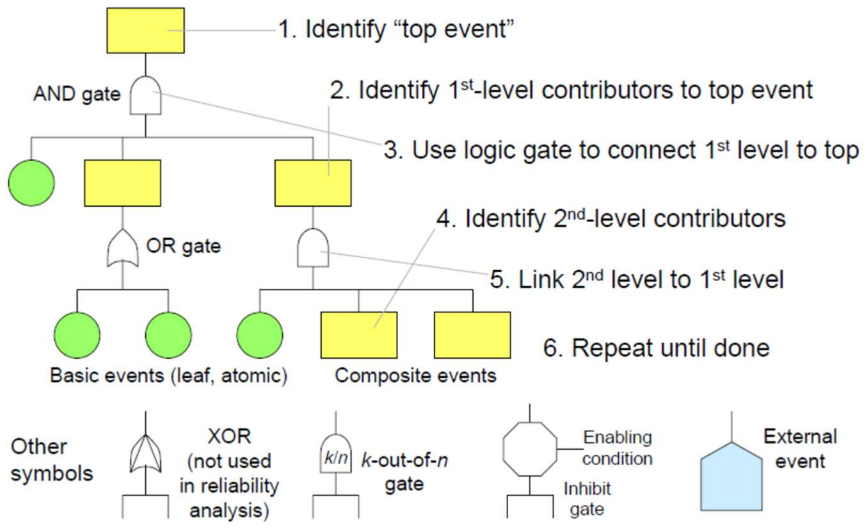

1. Identify “top event”
AND gate
2. Identify 1st-level contributors to top event
3. Use logic gate to connect 1st level to top
OR gate
4. Identify 2nd-level contributors
5. Link 2nd level to 1st level
6. Repeat until done
Basic events (leaf, atomic)
Composite events
Other symbols
XOR (not used in reliability analysis)
k/n
k-out-of-n gate
Enabling condition
Inhibit gate
External event

Fig. 3.17 Constructing a fault tree and some of the pertinent symbols.

For example, if the top event is being late for work, its immediate causes might be clock radio not turning on, family emergency, or the bus not running on time. Going one level down, the clock radio might fail to turn on due to the coincidence of a power outage and its battery being dead.

Once a fault tree has been built, it can be analyzed in at least two ways: using the cut-set method and via conversion to a reliability block diagram.

A cut set is any set of initiators so that the failure of all of them induces the top event. A minimal cut set is a cut set for which no subset is also a cut set. As an example, for the fault tree of Fig. 3.18, the minimal cut sets are $\{a, b\}$, $\{a, d\}$, and $\{b, c\}$. The equivalent RBD for a given fault tree is one that has the same minimal cut sets. An example is depicted in Fig. 3.18. Note that the equivalent RBD mat not be unique.

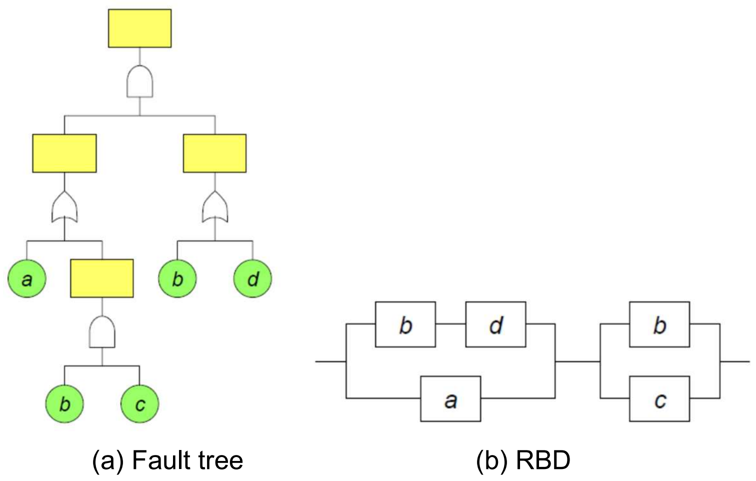


Fig. 3.18 An example fault tree and its RBD equivalent.

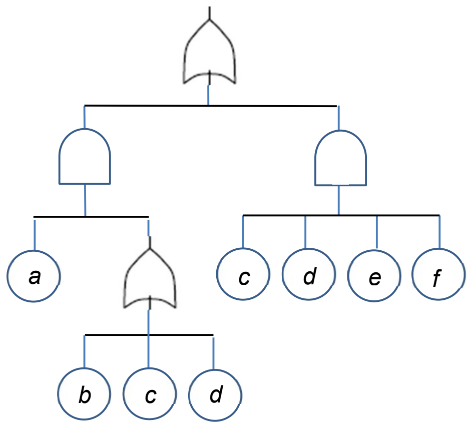
Other than allowing us to probabilistically analyze a fault tree, cut sets also help in common-cause failure assessment and exposition of system vulnerability: a small cut ses indicates high vulnerability. The notion of path set is the dual of cut set. A path set is any set of initiators so that if all of them are fault-free, the top event is inhibited. One can drive the path sets for a fault tree by exchanging AND and OR gates and then obtaining the cut sets for the transformed tree.

Example 3.10: Fault trees and RBDs Consider a system exhibiting the minimal cut set $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, $\{c, d, e, f\}$.

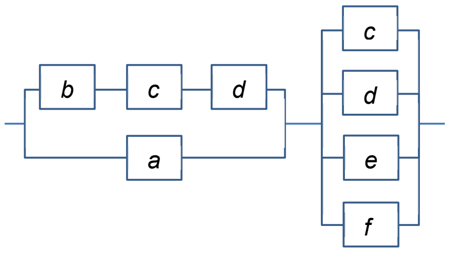
a. Construct a fault tree for the system.
b. Derive an equivalent RBD.
c. What is the path set for this example?

Solution:

a. A fault tree for the system is depicted in Fig. 3.19a.
b. One possible RBD is depicted in Fig. 3.19 b.
c. Exchanging AND and OR gates in Fig. 3.19a, we find the path set of the original diagram as the cut set of the transformed diagram, thus obtaining: $\{a, c\}$, $\{a, d\}$, $\{a, e\}$, $\{a, f\}$, $\{b, c, d\}$.



(a) Fault tree



(b) RBD

Fig. 3.19 Fault tree and RBD associated with Example 3.10.

In conclusion, we note that the combinational models introduced in this chapter constitute a hierarchy in terms of their representational power [Malh94]. At the top of this hierarchy, we have FTs with repeated elements. Reliability graphs are next, with somewhat less representational power. Finally, at the bottom of the hierarchy we have RBDs and ordinary FTs (no repeated elements), with these latter two models being equivalent in the sense of having identical representational power.

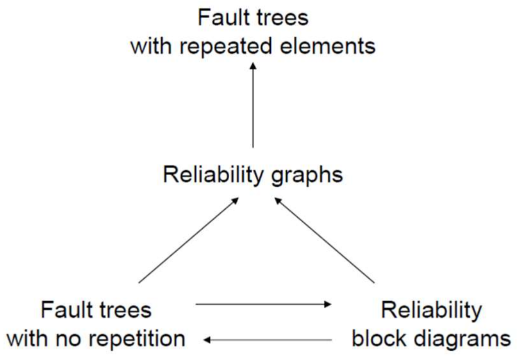


Fig. 3.20 The hierarchy of combinational reliability models.

Problems

3.1 Series and parallel resistors

a. Consider n resistors connected in series, with the dominant failure mode being an open-circuit between the resistor's two terminals (the resistance becoming infinite). Describe this set of resistors as a series or parallel system.

b. Repeat part a, this time assuming only short-circuit failures (the resistance becoming zero).

c. Repeat part a for the parallel connection of n resistors susceptible to open circuit failures.

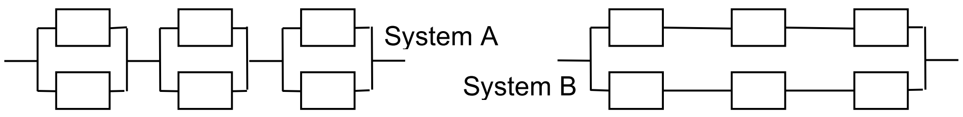
d. Repeat part c, this time assuming only short-circuit failures (the resistance becoming zero).

3.2 Series and parallel diodes

Repeat problem 3.1, replacing every occurrence of the word "resistor" with the word "diode." Note that an ideal, properly functioning diode has infinite resistance in the backward direction and zero resistance in the forward direction.

3.3 Series-parallel system reliability

Consider the following two series-parallel systems composed of six modules each. Assume identical modules of reliability r.



a. Write the reliability equations for the two systems and determine the conditions under which system A is more reliable than system B.

b. Repeat part a in the more general case when there are m pairs, rather than 3 pairs, of modules arranged as series connection of m parallel pairs or parallel connection of two m-module series chains.

c. Generalize the conclusions of part b to a case where instead of parallel pairs, we have k-wide parallel connections. In other words, assume that there are km modules in all, with the parallel parts being k-wide and the series parts being of length m.

3.4 Modeling of *k*-out-of-*n* systems

Figure 3.15 depicts two ways of modeling a 2-out-of-3 system as an RBD with repeated elements.

a. Is it possible to use an RBD with fewer than six blocks to achieve the same effect? Note that we are excluding the RBD variant in Fig. 3.13, which would need only three blocks.

b. Present at least three RBDs that model a 3-out-of-5 system.

c. What is the minimum number of blocks in an RBD that models a k-out-of-n system?

3.5 Combinational model types

A multiprocessor system is composed of an interconnection network N and two processors P_1 and P_2, each with its own memory (M_1 and M_2) and two disk drives (D_1 and E_1 for P_1, D_2 and E_2 for P_2). A processor can function as long as the processor itself, its memory unit, and one of its two disk drives are operational. Model this system by means of the following and obtain the corresponding reliability expressions.

a. Reliability block diagram

b. Reliability graph

c. Fault tree

3.6 Consecutive *k*-out-of-*n* systems

Prove that if we switch from linear indexing to circular indexing, reliability improves for consecutive *k*-out-of-*n*:G systems and deteriorates for consecutive *k*-out-of-*n*:F systems.

3.7 Circular consecutive *k*-out-of-*n* systems

a. Reformulate the redundant bus arrangement of Example 3.5 so that it corresponds to circular consecutive 3-out-of-5:G system and derive the resulting reliability.

b. Reformulate the street-lights application of Example 3.6 so that it corresponds to circular consecutive 2-out-of-*n*:F system and calculate the resulting reliability.

c. Propose an example of a consecutive *k*-out-of-*n*:F system that occurs in computer or computer-based systems.

3.8 RBD for a multiprocessor system

At the beginning of Section 3.4, Fig. 3.11 was described as possibly modeling a multiprocessor system. Can you characterize the RBD of Fig. 3.13 in the same manner? Discuss.

3.9 Modeling of a redundant resistor

Five identical resistors, each of resistance r, are connected into a bridge (Fig. 3.12b, with a fifth resistor included in the middle connection) and together act as one resistor. Each resistor fails to open with probability p and to short with probability q. Any end-to-end resistance value between $r/2$ and $2r$ is deemed acceptable. Derive an expression for the reliability of the overall component.

3.10 Reliability of *k*-out-of-*n* systems

A system consists of 3 processors and 8 memory modules with failure rates of 1 and 2 per 1000 hours, respectively. All other parts of the system are perfectly reliable.

a. What is the probability that no subsystem fails in a one-month period?

b. Assuming no repair, what is the probability that in a one-month period at least 2 processors and 4 memory modules remain available (i.e., a minimal system survives)?

c. If you had enough money to add one processor or one memory module for part b, which one would you choose and why?

3.11 Modeling of coverage in parallel systems

a. Model a 2-way parallel system with modules M_1 and M_2 having reliabilities r_1 and r_2 and imperfect coverage c by placing a special module representing the imperfect coverage on the parallel path for M_2. Then, derive the reliability equation as a function of r_1, r_2, and c.

b. Use induction on n to prove equation (3.2.cov3) for the reliability of an n-way parallel system with imperfect coverage.

c. How would equation (3.2.cov3) change if the coverage factor were different after each failure (c_1, c_2, … , c_{n-1}) and module reliabilities were also different (r_1, r_2, … , r_n)?

3.12 Modeling of a two-phase mission

A phased mission is one which requires the availability of different resources in each of several phases of operation. Consider for example a two-phase mission for a computer system with three resources (subsystems) A, B, and C. During phase 1, which lasts T_1 hours, only subsystem A needs to be operational. During phase 2, of duration T_2, proper functioning of subsystem B plus one of the other two subsystems would suffice. Such a mission is deemed a success if both phases are completed with the required resources being operational. Assuming exponential reliabilities, with constant failure rates λ_A, λ_B, and λ_C for the three resources (regardless of their being in operation or idle), write down the reliability equation for the two-phase mission just defined.

3.13 Modeling of a phased mission

Consider a system composed of n resources, numbered 1 to n, having reliabilities $R_1(t)$, $R_2(t)$, . . . , $R_n(t)$, and a ϕ-phase mission in which the set S_j of resources needed in phase j is a subset of S_{j-1} for $2 \le j \le \phi$.

a. Write down an expression for the reliability of the system if the completion of all ϕ phases is required for mission success.

b. Present the special case of your expression assuming exponential reliability formulas.

3.14 Series and parallel systems

Consider four nested rooms, the innermost of which contains a safe. To access the safe, one should go through four locked doors, one per room. Now consider a second security arrangement, with a single large room having four separate doors, any one of which can be used for entry to access the safe.

a. Discuss the suitability of each arrangement, and derive its reliability, when the door locks can only fail by becoming stuck on shut (can't be unlocked).

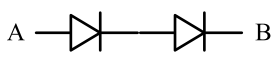
b. Repeat part a for locks that fail by becoming stuck on open (can't be locked).

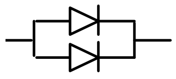
c. Assuming that we are limited to the use of four doors, what alternative security arrangement would you suggest if both types of lock failures mentioned in parts a and b are possible?

d. Derive the reliability of the arrangement you suggested in part c. State your assumptions and show all intermediate steps.

3.15 Combinational modeling

In a circuit, two diodes in series are used in lieu of a single diode between points A and B. Each diode has two failure modes: it fails in the open-circuit mode with probability p and in the closed-circuit (shorted) mode with probability q. The reliability of each diode is thus $1 - p - q$.





a. Write down the reliability equation of the series connection, and show that it is preferable to a nonredundant diode iff open-circuit failures are less likely than closed-circuit failures ($p < q$).

b. Repeat part a for two diodes in parallel replacing a single diode, and derive the corresponding condition.

c. Show that it is possible to arrange 4 diodes so that they act as a 3-out-of-4 system between A and B, given the open-circuit and short-circuit failure modes.

3.16 Assessing reliability

a. Which is more reliable: Plane X or Plane Y that carries four times as many passengers as Plane X and is twice as likely to crash?

b. Which is more reliable: a 4-wheel vehicle with 1 spare tire or an 18-wheeler with 2 spare tires?

3.17 Reliability wall

Consider a parallel computer with p processors running tasks of which a fraction f cannot be parallelized and the remaining fraction $1 - f$ is perfectly parallizable on all p processors. Amdah's constant-task speedup formula, $s = p/[1 + f(p - 1)]$, tells us that while computation speedup increases with p, it can never exceed the upper limit $1/f$ as p approaches infinity. On the other hand, Gustafson's constant-running-time scaled speedup $s = f + p(1 - f)$ continues to grow indefinitely for any $f < 1$ as p approaches infinity. With an increase in the number p of processors in exascale computing, reliability enhancement methods must be employed, which imply time and cost overheads. Checkpointing, for example, must be done more frequently as p increases, implying superlinear overhead in terms of work. If the reliability overhead is superlinear in p, then a reliability wall [Yang12] may inhibit further performance increases via the introduction of additional processors. Show that a reliability wall exists under the Amdahl interpretation of speedup, but not under Gustafson's.

3.18 Series-parallel systems with coincident failures

Consider a highly simplified scheme for modeling coincident failures in series/parallel systems. Systems of interest are composed of modules with identical failure probability p and coincident failure probability c for any two modules, with $c > p^2$. The probability of coincident failures in 3 or more modules is negligible.

a. Write down the reliability equation of the series connection of n such modules.

b. Write down the reliability equation of the parallel connection of n such modules.

c. Discuss how to apply the model defined in this problem to RBDs.

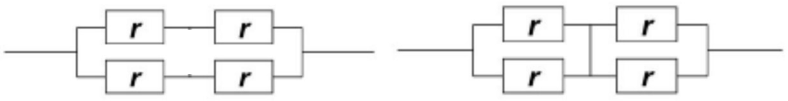
3.19 Short- and open-circuit tolerance for resistors

A resistor can fail by becoming open (infinite resistance) or by short-circuiting (zero resistance). To build a general model that accommodates both extremes as special cases, we assume that one of the resistors in the following networks, each intended to serve as a robust resistor built from 4 identical resistors, can assume the resistance $r + \rho$, with $-r < \rho < \infty$.

a. Derive a formula for the equivalent resistance of the network on the left as a function of r and ρ.

b. Repeat part a for the network on the right.

c. How would you go about choosing one of these redundant networks for a particular application?



3.20 Series-parallel systems

A pumping station has 3 pumps. Pump 1 must be working at all times, while only one of Pump 2 or Pump 3 is needed to be operational. The reliabilities of the three pumps for one month of operation are $R_1 = 0.999$, $R_2 = 0.988$, and $R_3 = 0.975$.

a. What is the failure probability for this system of pumps?

b. Assuming that the exponential reliability law is applicable, find the failure rates for the three pumps and an equivalent overall failure rate for the system of pumps.

c. Discuss the contribution of each pump to the equivalent overall failure rate of part b.

d. Consider improving one of the reliabilities by a small increment ε. If all three options cost the same, which pump would you choose to improve and why?

3.21 *k*-out-of-*n* vs. *k*-out-of-*k* reliability

A 34-out-of-36 system (e.g., a reconfigurable 6-by-6 processor array that can tolerate up to 2 failed elements) is more reliable than a 34-out-of-34 (series) system. Derive the extent of reliability improvement by writing the reliability of a 34-out-of-36 system, with each component having reliability r, as $r^{34}(1 + \alpha)$, where α is a function of r.

3.22 The Dr. Sanjay Gupta problem

The puzzle known as “The Monty Hall Problem,” bearing the name of an old-time game-show host, deals with counter-intuitive probabilistic notions (see Problem 2.4). For the title of this problem dealing with a probabilistic model of testing for diseases, I have borrowed the name of a CNN medical commentator. A newly developed diagnostic test for a particular disease gives the result “positive” with 99% probability if you have the disease and with 2% probability (false positive) if you don’t. Assume that 1% of the residents of a city have that particular disease. A randomly chosen person from the city is administered the test, with the result being “positive.” What is the probability that the person has the disease?

References and Further Readings

[Bent99] Bentley, J. P., *Reliability and Quality Engineering*, Addison-Wesley, 2nd ed., 1999.

[Buza70] Buzacott, J. A., "Network Approaches to Finding the Reliability of Repairable Systems," *IEEE Trans. Computers*, Vol. 19, No. 4, pp. 140-146, November 1970.

[Duga99] Dugan, J. B., "Fault Tree Analysis of Computer-Based Systems," tutorial presentation, accessed on July 22, 2012. http://www.fault-tree.net/papers/dugan-comp-sys-fta-tutor.pdf

[Kuo03] Kuo, W. and M. J. Zuo, *Optimal Reliability Modeling: Principles and Applications*, Wiley, 2003.

[Levi13] Levitin, G., L. Xing, H. Ben-Haim, and Y. Dai, "Reliability of Series-Parallel Systems with Random Failure Propagation Time," *IEEE Trans. Reliability*, Vol. 62, No. 3, pp. 637-647, September 2013.

[Malh94] Malhotra, M. and K. S. Trivedi, "Power-Hierarchy of Dependability Model Types," *IEEE Trans. Reliability*, Vol. 43, No. 3, pp. 493-502, September 1994.

[Misr08] Misra, K. B. (ed.), *Handbook of Performability Engineering*, Springer, 2008. {UCSB library has the e-book}

"Fault Tree Analysis," by Liudong Xing and Suprasad V. Amari, Chap. 38, pp. 595-620.

[NRC81] US Nuclear Regulatory Commission, *Fault-tree Handbook*, 1981, accessed on September 25, 2012. http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf

[Reib91] Reibman, A. L. and M. Veeraraghavan, "Reliability Modeling: An Overview for System Designers," *IEEE Computer*, Vol. 24, No. 4, pp. 49-57, April 1991.

[Sahn96] Sahner, R. A., K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*, Kluwer, 1996.

[Weib13] Weibul.com, "Fault Tree Analysis: An Overview of Basic Concepts," accessed on September 25, 2013. http://www.weibull.com/basics/fault-tree/index.htm

[Wind13] Windchill, "Fault-Tree Reliability Analysis Program," accessed on September 25, 2013. http://www.ptc.com/product/relex/fault-tree

[Xing08] Xing, L. and S. V. Amari, "Fault Tree Analysis," Chap. 38 in *Handbook of Performability Engineering*, ed. by K. B. Misra, Springer, 2008, pp. 595-620.

[Yang12] Yang, X., Z. Wang, J. Xue, and Y. Zhou, "The Reliability Wall for Exascale Supercomputing," *IEEE Trans. Computers*, Vol. 61, No. 6, pp. 767-779, June 2012.

4 State-Space Modeling

"All models are wrong, some are useful."

G. E. P. Box

"When one admits that nothing is certain one must, I think, also admit that some things are much more nearly certain than others."

Bertrand Russell

Topics in This Chapter

State-space models are suitable for modeling of systems that can be in multiple states from the viewpoint of functionality and can move from state to state as a result of deviations (e.g., malfunctions) and remedies (e.g., repairs). Even though many classes of state-space models have been introduced and are in use, our focus will be on Markov models that have been found most useful in practice and possess sufficient flexibility and power to faithfully represent nearly all system categories of practical interest. In this chapter, we introduce Markov chains as models of probabilistically evolving systems and explore their use in evaluating the dependability of computer systems, particularly those with repairable parts.

4.1 Markov Chains and Models

A discrete-time Markov chain can be viewed as a probabilistic sequential machine. Probability values are assigned to transitions, in such a way that the sum of probabilities associated with transitions out of any given state is 1. For example, Fig. 4.1 represents the Markov diagram for a four-state system, with the initial state 0 and transition probabilities as shown on the various edges. Note that the sum of probabilities associated with transitions out of various states does not equal 1, as required. For example, the two transitions out of state 0 have total probability of 0.7. This is because self-loops, or transitions from one state into the same state, are often omitted to reduce clutter. The presence of such self-loops is always implied in our Markov diagrams.

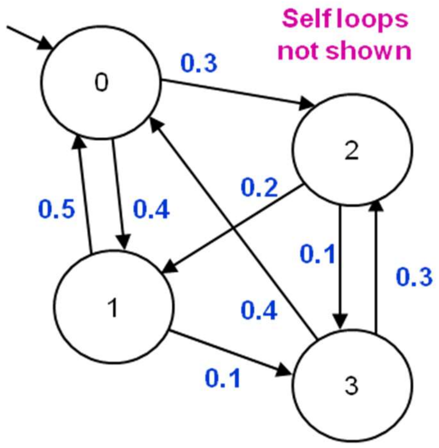


Fig. 4.1 State diagram of an example four-state Markov chain.

A Markov chain can be represented by a transition matrix *M*, where the element $m_{i,j}$ in row *i* and column *j* denotes the probability associated with the transition from state *i* to state *j* When the Markov chain has no transition between two states, the corresponding entry in *M* is 0. The transition matrix M_0 for the Markov chain of Fig. 4.1 is:

$$M_0 = \begin{pmatrix} 0.3 & 0.4 & 0.3 & 0.0 \\ 0.5 & 0.4 & 0.0 & 0.1 \\ 0.0 & 0.2 & 0.7 & 0.1 \\ 0.4 & 0.0 & 0.3 & 0.3 \end{pmatrix} \qquad (4.1.\text{exMm0})$$

A matrix such as the one in eqn. (4.1.exMm0), in which the sum of entries in each row add up to 1, is referred to as a *Markov matrix*. A second example of a Markov matrix is provided by eqn. (4.1.exMm1).

$$M_1 = \begin{pmatrix} 0.5 & 0.2 & 0.1 & 0.2 \\ 0.1 & 0.4 & 0.4 & 0.1 \\ 0.3 & 0.0 & 0.2 & 0.5 \\ 0.2 & 0.6 & 0.0 & 0.2 \end{pmatrix} \qquad (4.1.\text{exMm1})$$

At any given time, our knowledge about the state of an n-state Markov chain can be represented by an n-vector, with its ith element representing the probability of being in state i. For example, the state vector (1, 0, 0, 0) means that the system depicted in Fig. 4.1 is known to be in state 0, (1/2, 1/2, 0, 0) means that it is equally likely to be in state 0 or state 1, and (1/4, 1/4, 1/4, 1/4) denotes complete uncertainty about the state of the system. Clearly, the elements of a state vector must add up to 1.

Starting from state $s^{(0)} = (s_0^{(0)}, s_1^{(0)}, s_2^{(0)}, s_3^{(0)})$ at time 0, one can compute the state vector at time step 1 via multiplying s by the transition matrix M, that is, $s^{(1)} = s^{(0)}M$. More generally, the state vector of the system after k time steps is given by:

$$s^{(k)} = s^{(0)}M^k \qquad (4.1.\text{svec})$$

For example, if the system in Fig. 4.1 is initially in state 0 or 1 with equal probabilities, its state vector after one and two time steps will be:

$$s^{(1)} = (0.5, 0.5, 0, 0) \begin{pmatrix} 0.3 & 0.4 & 0.3 & 0.0 \\ 0.5 & 0.4 & 0.0 & 0.1 \\ 0.0 & 0.2 & 0.7 & 0.1 \\ 0.4 & 0.0 & 0.3 & 0.3 \end{pmatrix} = (0.40, 0.40, 0.15, 0.05)$$

$$s^{(2)} = (0.40, 0.40, 0.15, 0.05) \begin{pmatrix} 0.3 & 0.4 & 0.3 & 0.0 \\ 0.5 & 0.4 & 0.0 & 0.1 \\ 0.0 & 0.2 & 0.7 & 0.1 \\ 0.4 & 0.0 & 0.3 & 0.3 \end{pmatrix}$$
$$= (0.340, 0.365, 0.225, 0.070)$$

A discrete-time Markov chain can be viewed as a sequential machine with no input, also known as an *autonomous sequential machine*. In such a machine, transitions are triggered by the clock signal, with no other external influence. In conventional digital circuits, clock-driven counters are examples of autonomous sequential machines that move through a sequence of states on their own. If we have two or more possible input values, then a general *stochastic sequential machine* results. Such a machine will have a separate transition matrix for each input value and its state vector after *k* time steps will depend on the *k*-symbol input sequence that it receives.

Example 4.1: Stochastic sequential machines Consider a 4-state, 2-input stochastic sequential machine with the state transition probabilities provided by the matrix M_0 of eqn. (4.1.exMm0) for input value 0 and by the matrix M_1 of eqn. (4.1.exMm1) for input value 1. Assuming that the machine begins in state 0, what will be its state vector after receiving the input sequence 0100?

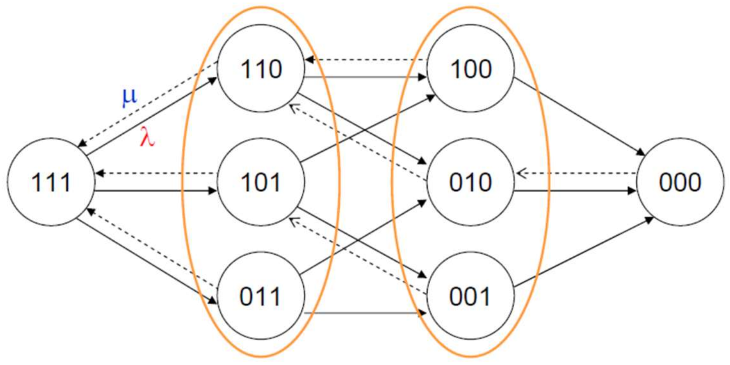
Solution: The initial state vector (1, 0, 0, 0) must be multiplied by $M_0M_1M_0M_0$ to find the final state vector. This process yields the state vector (0.2620, 0.2844, 0.3586, 0.0950).

Continuous-time Markov chains are more useful for modeling computer system performance and dependability. In such chains, transitions are labeled with rates, rather than probabilities. For example, a transition rate of 10^{-6} per hour means over a very short time period of *dt* hours, the transition will occur with probability $10^{-6}dt$. We often use Greek letters to denote arbitrary transition rates, with λ being a common choice for failure rate and μ for repair or service rate.

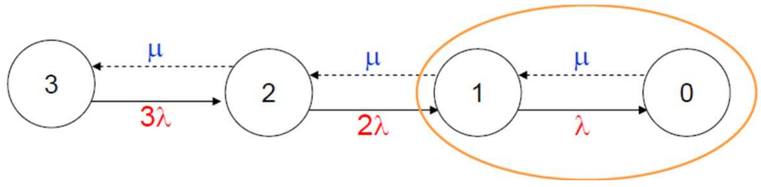
Markov chains are used widely for modeling systems of many different kinds. For example, the process of computer programming may be modeled by considering various states corresponding to problem formulation, program design, testing, debugging, and running [Funk07]. State transitions are labeled with estimated probabilities of going from one state (say, testing) to other states (say, debugging or running). Pattern recognition problems may be tackled by the so-called “hidden Markov models” [Ghah01]. As a final example, a person’s movement between home, office, and various other places of interest can be studied via a Markov model that associates probabilities with going from one of the places to each other place [Ashb03]. Many more applications exist [Bolc06].

Example 4.2: Markov model for a fail-soft multiprocessor Consider a parallel machine with 3 processors, each of which has a failure rate λ and a repair rate μ. When 2 or 3 processors fail, only one can be repaired at a time. The machine can operate with any number of processors, but its performance will be lower with only 1 or 2 processors working, thus providing fail-soft operation. Construct a suitable Markov model for this parallel machine.

Solution: Each of the 3 processors can be in the up (1) or down (0) state, leading to 8 possible states in all. For example the state 111 corresponds to all processors being operational and 110 corresponds to the third one being down. Figure 4.2a depicts the resulting 8-state Markov model and the associated state transitions. We know that the 3 processors are identical with regard to their failure and repair charactristics. If they are also the same in other respects, there is no need to distinguish the 3 states in which a single processor is down or the ones having 2 bad processors. Merging these states, we get the simplified Markov model of Fig. 4.2b. The new failure rates are obtained by summing the values on the merged transitions, but the repair rate does not change. In the event that a single processor cannot provide sufficient computational power for the tasks of interest, we might further merge states 0 and 1 in Fig. 4.2b, given that both of them imply the status of the system being "down." Thus, whether it is appropriate to simplify a Markov model by merging some states depends on the model's semantics, rather then its appearance.



(a) Initial Markov model



(b) Simplified Markov model

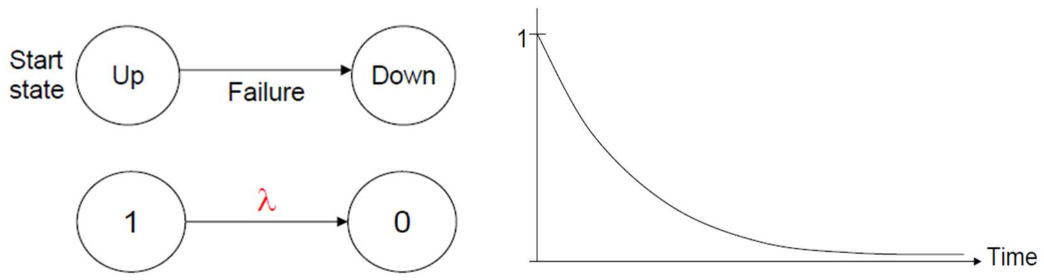
Fig. 4.2 **Markov models for a 3-processor fail-soft system. In part (a), all solid arrows should be labeled λ and all dashed arrows μ.**

4.2 Modeling Nonrepairable Systems

In modeling nonrepairable systems, the reliability equation and parameters such as MTTF are of interest. Such systems eventually fail, so our objective is often to determine the system lifetime and devise method for increasing it.

Example 4.3: Two-state nonrepairable system A nonrepairable system has the failure rate λ. Use a 2-state Markov model to find the reliability of this system as a function of time.

Solution: The requisite Markov model is depicted in Fig. 4.3a. Reliability can be viewed as the probability of the system being in state 1. To find the reliability as a function of time, we note that p_1 changes at the rate $-\lambda$ (failure rate being λ means that the probability of failure over a short time interval dt is λdt). Thus, we can set up the differential equation $p_1' = -\lambda p_1$, which has the solution $p_1 = e^{-\lambda t}$, given the initial condition $p_1(0) = 1$. Figure 4.3b shows a plot of the reliability $R(t) = p_1(t)$ as a function of time. The probability of the system being in state 0 can be found from the identity $p_0 + p_1 = 1$ to be $p_0 = 1 - e^{-\lambda t}$.



(a) System states and Markov model (b) System reliability over time

Fig. 4.3 Markov model and the reliability curve for a 2-state nonrepairable system.

Example 4.4: *n*-module parallel system Consider n lightbulbs lit at the same time, each failing independently at the rate λ. Construct a Markov chain to model this system of n parallel lightbulbs without replacement and use the model to find the expected time until all n lightbulbs die.

Solution: The requisite Markov model has n states, labeled from n (start state, when all lightbulbs are good) down to 0 (failure state) is depicted in Fig. 4.4 (ignore the dashed box, with is for Example 4.5). The expected time to go from state k to state $k - 1$ is $1/(k\lambda)$. Thus, the system's expected lifetime, or the time to go from state n to state 0, is $(1/\lambda)[1/n + 1/(n - 1) + \ldots + 1/2 + 1]$. We see that due to the use of parallel lightbulbs, the lifetime $1/\lambda$ of a single lightbulb has been extended by a factor equal to the sum of the harmonic series, which is $O(\log n)$ for large n.

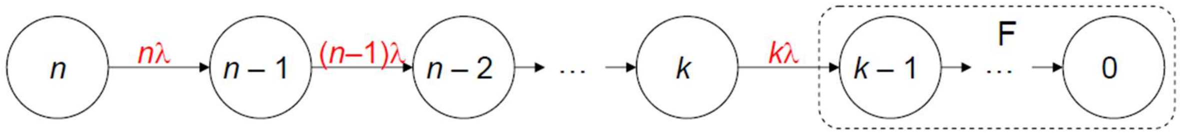


Fig. 4.4 **Markov model for an *n*-module parallel or *k*-out-of-*n* system, with the failure state being state 0 for the former and all the states within the dashed box in the latter.**

Example 4.5: *k*-out-of-*n* nonrepairable systems Construct and solve a Markov model for a nonrepairable *k*-out-of-*n* system.

Solution: The Markov model with $n + 1$ states, in which state i represents i of the n modules being operational, can be simplified by merging the last k states into a single "failure" or "down" state. The following balance equations can be written for the $n - k + 1$ good states:

$p_n' = -n\lambda p_n$

$p_{n-1}' = n\lambda p_n - (n-1)\lambda p_{n-1}$, and more generally, for $j \ge k$,

$p_j' = (j+1)\lambda p_{j+1} - j\lambda p_j$

From the system of $n + k + 1$ differential equations above, we find $p_n = e^{-n\lambda t}$, given the initial condition $p_n(0) = 1$, and more generally, for $j \ge k$,

$p_j = \binom{n}{n-j} (e^{-\lambda t})^j (1 - e^{-\lambda t})^{n-j}$

The problability p_F for the failure state can be obtained from $p_n + p_{n-1} + \ldots + p_k + p_F = 1$.

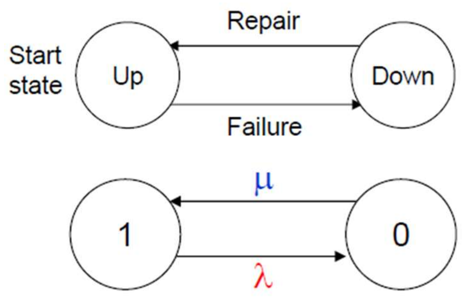
Note that to solve the system of differential equations of Example 4.5, we did not need to resort to the more general method involving LaPlace transform (see Section 4.6), because the first equation was solvable directly and each additional equation introduced only one new dependent variable.

4.3 Modeling Repairable Systems

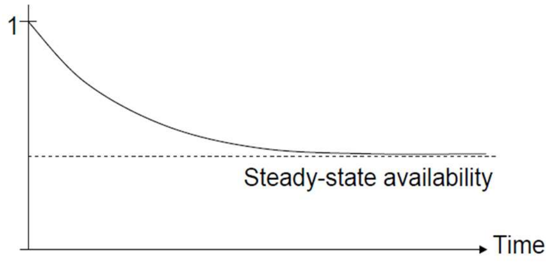
In a repairable system, the effect of failures can be undone by repair actions. One way to model variations in repair time is to associate a *repair rate* with each repair transition, as depicted in Fig. 4.5a. A repair rate μ means that the repair time has the exponential distribution, with the probability of repair action taking more time then t being $e^{-\mu t}$. The effectiveness and timeliness of repair actions can be assessed by evaluating the system's steady-state availability, or the fraction of time the system is expected to be in the "Up" state. The following example shows have the availability can be derived for the simplest possible repairable system.

Example 4.6: Two-state repairable system Consider a repairable system with failure rate λ and repair rate μ. Derive a formula for the steady-state system availability.

Solution: The requisite Markov model is depicted in Fig. 4.5a. Availability can be viewed as the probability of the system being in state 1. To find the steady-state availability, we set up the balance equation $-\lambda p_1 + \mu p_0 = 0$, which indicates that, over the long run, a transition out of state 1 has the same probability as a transition into it. The balance equation above, along with $p_0 + p_1 = 1$, allow us to obtain $p_1 = \mu/(\lambda + \mu)$ and $p_0 = \lambda/(\lambda + \mu)$. We will see later that availability of this system as a function of time is $A(t) = \mu/(\lambda + \mu) + \lambda/(\lambda + \mu)e^{-(\lambda+\mu)t}$, which is consistent with the steady-state availability $A = \mu/(\lambda + \mu)$ just derived.



(a) System states and Markov model



(b) System availability over time

Fig.4.5 Markov model and the availability curve for a 2-state repairable system.

As discussed in Section 2.5, we may want to distinguish different failure states to allow a more refined safety analysis. Another reason for contemplating multiple failure states is to model different rates of failures as well as different rates of repair in a repairable system. Some failures may be repairable, while others aren't, and classes of failures may differ in the difficulty and latency of repair actions. The following example illustrates the Markov modeling process for one class of such systems.

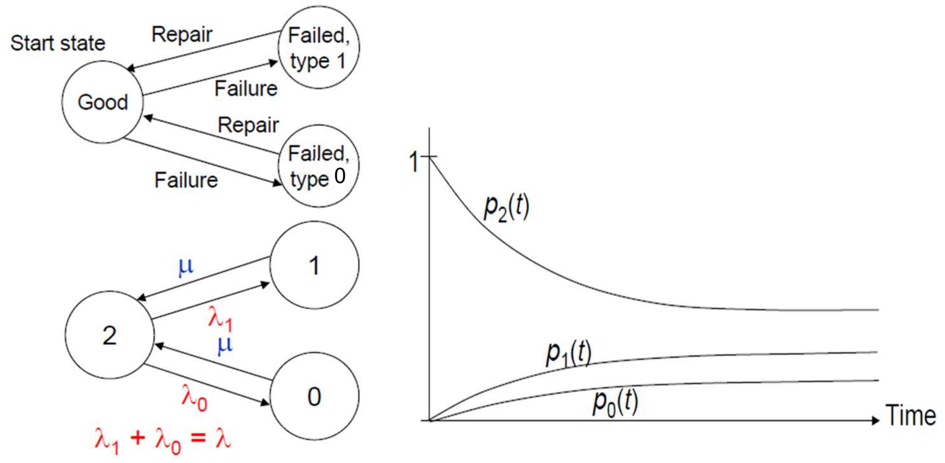
Example 4.7: Repairable system with two failure states Consider a repairable system with failure rate $\lambda = \lambda_1 + \lambda_2$, divided into two parts λ_0 for failures of type 0 and λ_1 for failures of type 1. Assuming the common repair rate μ for both failure types, derive a formula for the steady-state system availability and the probabilities in being in the two failure states.

Solution: The requisite Markov model is depicted in Fig. 4.6a. To obtain steady-state probabilities of the system being in each of its 3 states, we write the following two balance equations and use them in conjuction with $p_0 + p_1 + p_2 = 1$ to find $p_0 = \lambda_0/(\lambda + \mu)$, $p_1 = \lambda_1/(\lambda + \mu)$, and $p_2 = \mu/(\lambda + \mu)$.

$-\lambda p_2 + \mu p_1 + \mu p_0 = 0$

$-\mu p_1 + \lambda_1 p_2 = 0$

Figure 4.6b shows typical variations in state probabilities over time (with the derivation to be discussed later) and their convergence to steady-state values just derived. If penalties or costs c_j are associated with being in the various failed states, analyses of the kind presented in this example allow the computation of the total system risk as $\Sigma_{\text{failure states}}\, c_j p_j$.



(a) System states and Markov model (b) State probabilities over time

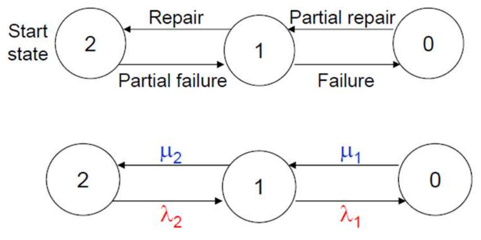
Fig.4.6 Markov model and state probabilities for a repairable system with two failure states.

4.4 Modeling Fail-Soft Systems

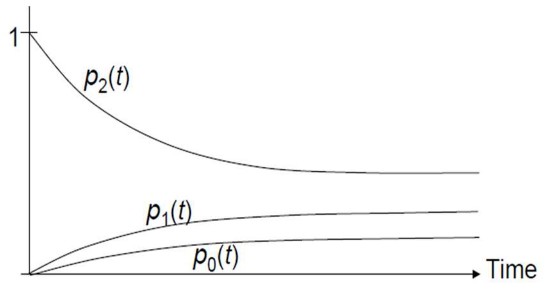
As discussed in Section 2.4, we may want to have multiple operational states, associated with different levels of system capability, to allow for performability analysis. As in the case of multiple failure states, discussed in Section 4.3, different operational states may differ in their failure and repair rates, allowing for more accurate reliability and availability analyses. Such states may also have different rewards or benefits b_j associated with them so that a weighted total benefit $\Sigma_{\text{operational states}}\, b_j p_j$ can be computed. The following examples illustrate the Markov modeling process for such systems.

Example 4.8: Repairable system with two operational states Consider a repairable system with operational states 1 and 2 and failure state 0, as depicted in Fig. 4.7a with its associated failure and repair rates. Derive the probabilities of being in the various states and use them to compute system availability and system performability, the latter under the assumption of the performance or benefit b_2 associated with state 2 is twice that of $b_1 = 1$ for state 1.

Solution: The requisite Markov model is depicted in Fig. 4.7a. To obtain steady-state probabilities for the system states, we write the following two balance equations and use them in conjuction with $p_0 + p_1 + p_2 = 1$ to find $p_0 = \delta\lambda_1\lambda_2/(\mu_1\mu_2)$, $p_1 = \delta\lambda_2/\mu_2$, and $p_2 = \delta$, where $\delta = 1/[1 + \lambda_2/\mu_2 + \lambda_1\lambda_2/(\mu_1\mu_2)]$.
$-\lambda_2 p_2 + \mu_2 p_1 = 0$
$\lambda_1 p_1 - \mu_1 p_0 = 0$
Figure 4.7b shows typical variations in state probabilities over time and their convergence to steady-state values just derived. System availability is simply $A = p_1 + p_2 = \delta(1 + \lambda_2/\mu_2)$. Assuming a performance level of 1 unit in state 1 and 2 units in state 2, system performability is $P = p_1 + 2p_2 = \delta(2 + \lambda_2/\mu_2)$. If the performance level of 2 in state 2 results from 2 processors running in parallel, it might be reasonable to assume $\lambda_2 = 2\lambda_1 = 2\lambda$ and $\mu_2 = \mu_1 = \mu$ (single repair facility). Then, assuming $\rho = \mu/\lambda$, we get $A = \rho(\rho + 2) / (\rho^2 + 2\rho + 2)$ and $P = 2\rho(\rho + 1) / (\rho^2 + 2\rho + 2)$.



(a) System states and Markov model



(b) State probabilities over time

Fig.4.7 Markov model and state probabilities for a repairable system with two operational states.

Example 4.9: Fail-soft systems with imperfect coverage We saw in Section 3.2 that adding parallel redundancy without ensuring accurate and timely malfunction detection and switchover may not be helpful. We also introduced a coverage parameter *c* and used it to derive the reliability of an n-way parallel system under imperfect coverage in eqn. (3.2.cov3). Analyze a repairable fail-soft system composed of 2 processors, so that upon a first processor failure, successful switching to a single-processor configuration occurs with probability *c*.

Solution: The requisite Markov model is depicted in Fig. 4.8. Note that the transition labeled λ_2 in an ordinary fail-soft system has been replaced with two transitions: one labeled $\lambda_2 c$ into state 1, corresponding to successful reconfiguration, and another labled $\lambda_2(1 - c)$ into state 0, representing catastrophic failure upon the first module outage. To obtain steady-state probabilities for the system states, we write the following two balance equations and use them in conjuction with $p_0 + p_1 + p_2 = 1$ to find p_0, p_1, and p_2.
$-\lambda_2 p_2 + \mu_2 p_1 = 0$
$\lambda_2(1 - c)p_2 + \lambda_1 p_1 - \mu_1 p_0 = 0$
Even though we have set up the model in full generality in terms of failure and repair rates, we solve it only for $\lambda_2 = 2\lambda_1 = 2\lambda$ and $\mu_2 = \mu_1 = \mu$. Defining $\rho = \mu/\lambda$ and $\theta = 1/[\rho^2 + (4 - 2c)\rho + 2]$, we get $p_0 = 2[(1 - c)\rho + 1]\theta$, $p_1 = 2\rho\theta$, and $p_2 = \rho^2\theta$. Note that just as reconfiguration can be unsuccessful, so too repair might be imperfect or the system may be unable to reuse a properly repaired module. Thus, the use of coverge is appropriate for repair transitions as well.

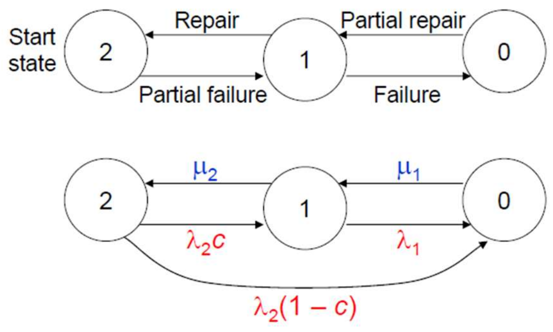


Fig.4.8 Markov model for a fail-soft system with imperfect coverage.

4.5 Solving Markov Models

In the preceding sections, we introduced the method of transition balancing for finding the steady-state probabilities associated with the states of a Markov model. This method is quite simple and is readily automated throught the use of reliability modeling tools or general solvers for systems of linear equation. Occasionally, however, we would like to derive the transient solutions of a Markov chain to study the short-term behavior of a system or to gain insight into how quickly the system reaches its steady state.

To find the transient solutions to a Markov model, we first set up a system of first-order differential equations describing the time-variant balance (rather than the steady-state balance) at each state. We then apply the Laplace transform to the system of differential equations, solve the resulting system of algebraic equations, and find the final solutions via the application of inverse LaPlace transform. The LaPlace transform converts a time-domain function $f(t)$ to its transform-domain counterpart $F(s)$ by:

$$F(s) = Laplace[f(t)] = \int_0^\infty e^{-st} f(t) dt \qquad \text{(4.5.LPT)}$$

By convention, an uppercase letter is used to denote the Laplace transform of a function named with the corresponding lowercase letter. Table 4.1 lists what we need to know about the Laplace transforms of simple functions to solve problems encountered in analyzing dependable systems. More background on Laplace (also written as "LaPlace") transform, and a more extensive table of transforms, can be found in [Wiki15].

Table 4.1 Laplace transforms for some simple functions.

Time-domain function	Transform-domain function
k; constant	k/s
e^{-at}	$1/(s+a)$
$t^k/k!$; $k \ge 0$	$1/s^{k+1}$
$t^k e^{-at}/k!$	$1/(s+a)^{k+1}$
$k.h(t)$; constant k	$k.H(s)$
$h(k.t)$; constant $k > 0$	$H(s/k)/k$
$t.h(t)$	$-dH(s)/ds$
$g(t) + h(t)$	$G(s) + H(s)$
$h'(t)$; derivative of $h(t)$	$s.H(s) - h(0)$; $h(0)$ is the initial value of h

One more technique that we need is that of partial fraction expansion. Here is a brief review, which is adequate for most cases. More details and a large set of examples can be found in [Wiki15a].

Given a fraction $N(s)/D(s)$, where $N(s)$ and $D(s)$ are the numerator and denominator polynomials in s, with $N(s)$ being of a lower degree than $D(s)$, it can be expanded as a sum of fractions of the form $a_i/(s - r_i)$, assuming that the r_i values are the roots of the equation $D(s) = 0$. We assume that the roots are simple. Repeated roots require a modification in this process that we will not discuss here (see Problem 4.21). This partial fraction expansion allows us to convert an arbitrary fraction to a sum of fractions of the forms that appear in the right-hand column of Table 4.1 and thus be able to apply the inverse LaPlace transform to them.

$$\frac{N(s)}{D(s)} = \frac{N(s)}{\prod_{i=1}^{k}(s-r_i)} = \frac{a_1}{s-r_1} + \frac{a_2}{s-r_2} + \ . \ . \ . + \frac{a_k}{s-r_k} \qquad (4.5.\text{pfe1})$$

By converting the sum of the fractions on the right-hand side of equation (4.5.pfe1) to a single fraction having the product of the denominators as its denominator and equating the coefficients of the various powers of s in the numerators of both sides, we readily derive the constants a_i as:

$$a_i = \left[\frac{(s-r_i)N(s)}{D(s)}\right]_{s=r_i} \qquad (4.5.\text{pfe2})$$

The subscript $s = r_i$ is equation (4.5.pfe2) means that the bracketed expression is to be evaluated at $s = r_i$ to yield the value of the constant a_i.

Example 4.10: Two-state repairable systems Find transient state probabilities for the 2-state repairable system of Fig. 4.5a and show that the results are consistent with the transient availability curve depicted in Fig. 4.5b.

Solution: We begin by setting up the balance differential equations for the two states.

$p_1'(t) = -\lambda p_1(t) + \mu p_0(t)$

$p_0'(t) = -\mu p_0(t) + \lambda p_1(t)$

Using Laplace transform, we convert the equations above into algebraic equations, noting the initial conditions $p_1(0) = 1$ and $p_0(0) = 0$.

$sP_1(s) - p_1(0) = -\lambda P_1(s) + \mu P_0(s)$

$sP_0(s) - p_0(0) = -\mu P_0(s) + \lambda P_1(s)$

Noting the initial conditions $p_1(0) = 1$ and $p_0(0) = 0$, we find the solutions:

$P_1(s) = (s + \mu) / [s^2 + (\lambda + \mu)s]$

$P_0(s) = \lambda / [s^2 + (\lambda + \mu)s]$

To apply the inverse Lapalce transform to $P_1(s)$ and $P_0(s)$, we need to convert the right-hand sides of the two equations above into function forms whose inverse Laplace transform are known to us from Table 4.1. This is done via partial-fraction expansion discussed just before this example.

$P_0(s) = \lambda / [s^2 + (\lambda + \mu)s] = a_1/s + a_2/(s + \lambda + \mu)$

The denominators s and $s + \lambda + \mu$ on the right are the factors of the original denominator on the left, while a_1 and a_2 are constants to be determined by insuring that the two sides are equal for all values of s. From this process, we find $a_1 = \lambda/(\lambda + \mu)$ and $a_2 = -\lambda/(\lambda + \mu)$, allowing us to write down the final result.

$p_0(t) = InverseLaplace[a_1/s + a_2/(s + \lambda + \mu)] = \lambda/(\lambda + \mu) - \lambda/(\lambda + \mu)e^{-(\lambda + \mu)t}$

The inverse Laplace transform is similarly applied to $P_1(s)$, yielding:

$p_1(t) = \mu/(\lambda + \mu) + \lambda/(\lambda + \mu)e^{-(\lambda + \mu)t}$

These results are consistent with $p_1(t)$ shown in Fig. 4.5b decreasing from 1 at $t = 0$ to $\mu/(\lambda + \mu)$ in steady state ($t = \infty$).

Example 4.11: Triplicated system with repair The lifetime of a triplicated system with voting (Fig. 3.7) can be extended by allowing repair or replacement to take place upon the first module malfunction. In this way, it is possible for the system to return to full functionality, with 3 working units, before a second module malfunction renders it inoperable. Only if the second malfunction occurs before the completion of repair or replacement will the system experience failure. Analyze the MTTF of such a TMR system with repair.

Solution: The Markov model for this system is depicted in Fig. 4.9. Steady-state probabilities for the system states can be obtained from the following two balance equations, used in conjuction with $p_0 + p_1 + p_F = 1$. Unfortunately, this steady-state analysis is unhelpful, because it leads to $p_0 = p_1 = 0$ and $p_F = 1$, which isn't surprising (why?).

$-3\lambda p_0 + \mu p_1 = 0$

$3\lambda p_0 - (2\lambda + \mu)p_1 = 0$

In general, with a failure state that has no outgoing transitions, a so-called *absorbing state*, we get $p_F = 1$ in steady state.

Time-variant state probabilities can be obtained from the following differential equations, with the initial conditions $p_0(0) = 1$ and $p_1(0) = 0$.

$p_0' = -3\lambda p_0 + \mu p_1$

$p_1' = 3\lambda p_0 - (2\lambda + \mu)p_1$

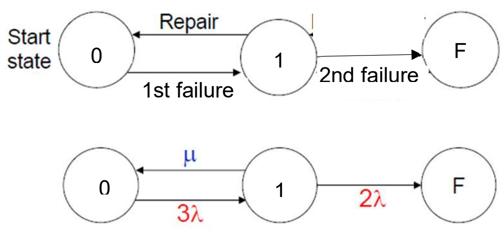
Using Laplace transform, we can solve the equations above to obtain $R(t) = p_0(t) + p_1(t)$. The result, with the notational conventions $\rho = \mu/\lambda$ and $\delta = (\rho^2 + 10\rho + 1)^{1/2}$ becomes:

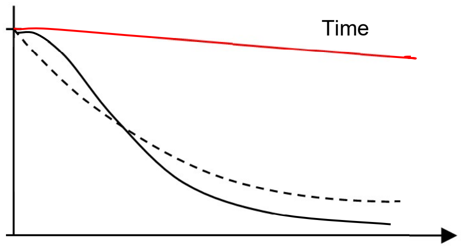
$R(t) = [(\rho + \delta + 5)e^{-(\rho-\delta+5)\lambda t/2} - (\rho - \delta + 5)e^{-(\rho+\delta+5)\lambda t/2}]/(2\delta)$

Integrating $R(t)$, per eqn. (2.2.MTTF), we find, after simplification:

MTTF $= [1 + (\rho - 1)/6]/\lambda = (1 + \rho/5)[5/(6\lambda)]$

This result suggests that the provision of repair extends the MTTF of a TMR system by a factor of $1 + \rho/5$ over that of an nonrepairable TMR system, which in turn has a smaller MTTF (by a factor of 5/6) compared with a nonredundant module. For example, with $\lambda = 10^{-6}$/hr and $\mu = 0.1$/hr, we have $\text{MTTF}_{\text{Module}}$ = 1M hr, MTTF_{TMR} = 0.833M hr, and $\text{MTTF}_{\text{TMR with repair}}$ = 16,668M hr.





(a) System states and Markov model (b) Reliabilities over time

Fig.4.8 Markov model and graphic depiction of reliability and MTTF improvements for a repairable TMR system.

4.6 Dependability Modeling in Practice

A particularly useful Markov model is the so-called *birth-and-death process*. This model is used in queuing-theory analysis, where the customer's arrival rate and providers' service rate determine the queue size and waiting time. Referring to the infinite birth-and-death process depicted in Fi.g 4.10, transition from state j to state $j + 1$ is an arrival or birth. Transition from state j to state $j - 1$ is a departure or death. Closed-form solutions for state probabilities are difficult to obtain in general, but steady-state probabilities are easily obtained:

$$p_j = p_0\lambda_0\lambda_1 \ldots \lambda_{j-1} / (\mu_1\mu_2 \ldots \mu_j) \qquad (4.6.\text{BnD})$$

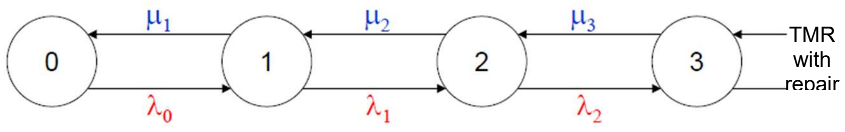


Fig. 4.10 Markov model for a birth-and-death process.

In the finite version of Fig. 4.10, the last state is labeled n and we have an $(n + 1)$-state birth-and-death process. The following two examples deal with special cases of the process shown in Fig. 4.10

Example 4.12: A simple queuing system for bank customers Customers enter a bank at the rate of λ and they are serviced by a single teller at the rate of μ. Even if $\lambda < \mu$, the length of the waiting line can grow indefinitely due to random variations. Use a birth-and-death process to compute the probability of the queue length becoming n for different values of n and determine the average queue length.

Solution: Using eqn. (4.6.BnD), with all the λ_i set to λ and all the μ_i set to μ, and taking $\rho = \mu/\lambda$, we find $p_j = p_0/\rho^j$. From $\Sigma_{j\geq 0}\, p_j = 1$, we obtain $p_0 = 1 - 1/\rho$ and $p_j = (1 - 1/\rho)/\rho^j$. The average queue length is $\Sigma_{j\geq 0}\, jp_j = (1 - 1/\rho)\Sigma_{j\geq 0}\, j/\rho^j = (1 - 1/\rho)\rho/(\rho - 1)^2 = 1/(\rho - 1) = \lambda/(\mu - \lambda)$. Note that for $\rho = 1$, the queue length becomes unbounded, and for a service rate μ that is slightly greater than but very close to the arrival rate λ, the queue can become quite long.

Example 4.13: Gracefully degrading system with *n* identical modules The behavior of a gracefully degrading system with n identical modules can be modeled by an $(n + 1)$-state birth-and-death process, where state j represents the unavailability of j modules (state 0, with all the modules being functional, is the initial state). Find the probability of the system being in state 0, assuming up to s modules can be repaired at once (the so-called $M/M/s$ queue, where M stands for Markov process for failure and repair and s is the number of service stations or providers).

Solution: Figure 4.11 depicts the markov model for the system under consideration. The repair rates used depend on the value of s. For $s = 1$, all repair transitions are labeled μ. For general s, repair transitions are labeled μ, 2μ, … , $s\mu$, from the left side of the diagram, with the remaining transitions, if any, all labeled $s\mu$, the maximum repair rate with s service providers. Applying balance equations and defining $\rho = \mu/\lambda$, we can find the steady-state probabilities of being in the various states as:

$p_j = (n - j + 1)p_{j-1}/(j\rho)$, for $1 \le j \le s$

$p_j = (n - j + 1)p_{j-1}/(s\rho)$, for $s + 1 \le j \le n$

The equations above yield each state probability in terms of p_0:

$p_j = \binom{n}{j}\rho^{-j}p_0$, for $1 \le j \le s$

$p_j = \binom{n}{j}\rho^{-j}[j!/(s!s^{j-s})]p_0$, for $s + 1 \le j \le n$

Using $p_0 + p_1 + \ldots + p_n = 1$, we can compute p_0 to complete the derivation:

$p_0 = 1/[\sum_{i=0}^{s}\binom{n}{j}\rho^{-j} + \sum_{i=s+1}^{n}\binom{n}{j}\rho^{-j}j!/(s!\,s^{j-s})]$

The state probabilities just derived can be used to compute system availability (summing over states in which the system is deemed available) or performability (weighted sum, where the weight for each state is a measure of system performance at that state).

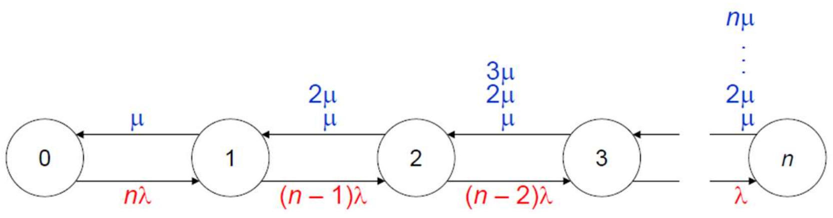


Fig. 4.11 **Markov model for an *n*-module gracefully degrading system. Repair transition rates may vary, depending on the number of servers or repair stations.**

Dependability modeling is an iterative process, as depicted in Fig. 4.12. Even if the modeling approach is chosen correctly at the outset, fine-tuning of assumptions and parameters may be required in multiple iterations, until satisfactory and trustworthy results have been obtained. In carrying out the modeling process, various software aids can be useful. A wide variety of such aids have been developed over the years and many of them are available free of charge to the general public or for academic research.

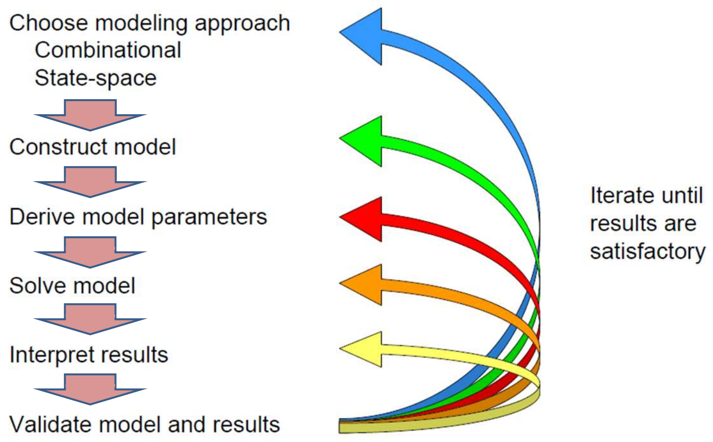


Fig. 4.12 The dependability modeling process.

Examples of software aids for dependability modeling include those offered by PTC Windchill (formerly Relex) [PTCW20] and ReliaSoft [Reli20], companies specializing in reliability engineering, University of Virginia’s Galileo [UVA20], and Iowa State University’s HIMAP [IASU20]. There are also more limited tools associated with Matlab and a number of Matlab-based systems. A 2004 study enumerates a set of possibly useful tools [Bhad04]. A Google search for “reliability analysis tools” will reveal a host of other products and guides.

Problems

4.1 Four-processor fail-soft system

A multiprocessor system with 4 nodes has a per-processor malfunction rate of 1 per 500 hours and a repair rate of 1 per 10 hours. All other system components are assumed to be perfectly reliable for this problem. Each processor can perform 1 unit of computation per hour. The system is deemed available as long as at least one processor is functioning. Derive the system's availability, performability, and MCBF.

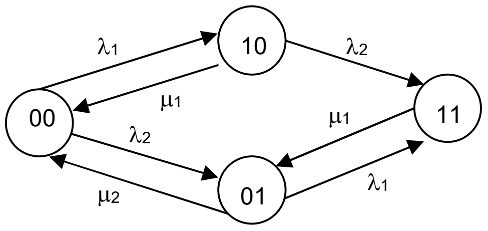
4.2 Automobile with a spare tire

Consider an automobile with four regular tires and one spare tire. The failure rate of a regular tire is λ. A spare fails at the rate $\lambda_1 < \lambda$ when it is in the trunk and at the rate $\lambda_2 > \lambda$ when it is used. When the spare is in use, the replaced failed tire is repaired at the rate μ. Assume no repair for a failed spare in the trunk, because this event is usually not detected. The system fails when any two tires are unusable.

a. Construct a state-space model for this system and derive the associated reliability equation.
b. What do we need to add to the model in order to allow the derivation of steady-state availability?
c. Relate this example to a computer-based system that you describe.

4.3 Two-processor fail-soft system

The following Markov model is known to correspond to a two-processor fail-soft system.



a. Explain the assumptions under which the model was developed. In particular, pay attention to the lack of full symmetry in the state diagram.
b. Solve the simplified model to derive the steady-state probabilities associated with the four states.

4.4 Delayed failure detection

Consider a state-space model for a system that can be in one of three states: G (good), U (bad, failure undetected), F (bad, failure detected). Assume failure rate of λ, repair rate of μ, and failure detection "rate," modeling the latency of failure detection, of δ.

a. Calculate the steady-state availability of this system.
b. Discuss the implications of delayed failure detection by comparing your result with that of a two-state system that has immediate failure detection.

4.5 Modeling a championship series

Two sports teams play a 7-game championship series, with the series ending as soon as one team has won 4 games. The state of the championship series at any time can be represented by the pair (w_1, w_2), where w_i is the number of games won by team i. Assume that team 1 wins each game with probability p, independent of all previous results. Tie scores are not allowed.

a. Model this championship series as a Markov process.

b. Use the model of part a to find the probability C_i that team i (i = 1, 2) wins the championship.

4.6 Modeling a small tournament

Four sports teams play in a single-elimination tournament. In the first round, teams 1 and 4 play each other in one game and teams 2 and 3 in another. Winners of these two games then play for the championship, while the losers play for the third place. The probability that team i wins over team j in a game is p_{ij}, independent of all previous results. Tie scores are not allowed.

a. Model this tournament as a Markov process.

b. Use the model of part a to find the probability C_{ij} that team i finishes in jth place ($1 \le i, j \le 4$).

4.7 A parallel system of lightbulbs with replacement

Consider n lightbulbs, all lit at the same time, that fail independently at the rate λ. In Example 4.4, we analyzed this system under the assumption of no replacement. Assume that a building custodian replaces the failed lightbulbs at a fixed rate μ. Detection of a dead lightbulb is immediate.

a. Construct a Markov chain to model this system of n parallel lightbulbs with replacement and use the model to find the expected time until all n lightbulbs die.

b. Find numerical values for the expected lifetimes without and with replacement, assuming a failure rate of 0.001/hr and a replacement rate of 10/hr.

4.8 Weather in the Land of Oz

The Land of Oz is said to have a peculiar weather pattern. It never has two nice days in a row. The day following a nice day is equally likely to be rainy or snowy. When it snows or rains, there is an even chance that next day's weather is the same. If there is a change from snow or rain, only half the time the change involves having a nice day.

a. Represent the weather in the Land of Oz as a 3-state Markov chain.

b. Show that if the transition matrix M for the Markov model of part a is raised to the nth power, the (i, j) entry of the resulting matrix M^n represents the probability of having type-j weather n days after having type-i weather.

c. Show that in the long run, weather in the Land of Oz will be totally independent of today's weather.

4.9 Moving balls between urns

We have two urns, together holding 4 balls. At each step, one of the 4 balls is chosen at random and moved from its current place to the other urn.

a. Taking the number of balls in urn 1 as the system state, represent this process as a Markov chain.

b. Assuming that we begin with no ball in urn 1, plot the variation of the expected number of balls in urn 1 as a function of steps.

c. Repeat part b, this time assuming that urn 1 has 2 balls at the outset.

4.10 The stepping-stone model

The stepping-stone model is used in the study of genetics. We have a square array of side n, with each of the n^2 cells having one of k different colors. Each cell has 8 nearest neighbors, even the cells on the edge of the array, which are assumed to be neighbors with cells at the other end of the row or column. In each step, a cell is chosen at random and is given the same color as a randomly chosen nearest neighbor.

a. How many states are there in the Markov model of this process?

b. Is it feasible to solve the Markov model for $k = 2$ colors in an array of side length $n = 10$?

c. Write a program to experiment with and observe state changes in the example of part b.

d. Prove that eventually, all cells will assume the same color.

e. Prove that the probability that one of two initial colors prevails is equal to the proportion of cells that are of that color at the start.

4.11 Making bail

A man in jail has \$300 but need \$800 to get out on bail. A prison guard agrees to make a series of bets with him. If the prisoner bets \$$x$, he wins \$$x$ with probability 0.4 and loses \$$x$ with probability 0.6.

a. Find the probability that the prisoner accumulates \$800 before losing all of his money, assuming he bets \$100 each time?

b. Repeat part a, this time assuming that the prisoner bets as much as possible, but not more than the amount he needs to reach \$800.

c. Which strategy, the one in part a or that of part b, gives the prisoner a better chance of making bail?

4.12 Absorbing Markov chains

An *absorbing state* in a Markov chain is one that has no outgoing transition. In an *absorbing Markov chain*, some absorbing state can be reached from any *transient state*. The transition matrix M of an absorbing Markov chain with r absorbing and $n - r$ transient states has the canonical form

$$M = \begin{pmatrix} Q & R \\ 0 & I_r \end{pmatrix}$$

where Q is a square matrix of size $n - r$ and I_r is the identity matrix of size r.

a. Provide an interpretation for the value of the (i, j) entry of the matrix Q^k.

b. Prove the identity $N = \sum_{j=0}^{\infty} Q^j = (I_{n-r} - Q)^{-1}$.

c. Show how the *fundamental matrix N* of part b can be used to find the expected number of steps before absorption, when starting from an arbitrary transient state.

d. Model the process of flipping a fair coin until the sequence HTH is observed as an absorbing Markov chain and find the expected number of flips before the desired pattern occurs.

4.13 The game of tennis

The game of tennis may enter a tie state called "Deuce," which requires one player to make two straight points in order to win. If player A makes a point, the game goes from "Deuce" to the "Advantage A" state, from which it either goes to the "A wins" state or back to the "Deuce" state, depending on who makes the next point. The "Advantage B" and "B wins" states are defined analogously. Suppose that player A has a probability p of winning any given point, regardless of the previous history of the game.

a. Set up a Markov chain to model the game of tennis, beginning from the "Deuce" state.

b. Find the probabilities of reaching the two absorbing states "A wins" and "B wins."

c. Starting at "Deuce," what is the expected number of points played before the game ends?

4.14 Stochastic sequential machines

Consider the stochastic sequential machine defined in Example 4.1.

a. Derive the machine's final state vector, given an initial state vector and a very long sequence of 0s as input.

b. Repeat part a for a very long sequence of 1s as input.

c. Repeat part a for a very long input sequence beginning with 0 and containing alternate 0s and 1s.

4.15 Modeling a 3-processor fail-soft system

Solve the simplified Markov model of Fig. 4.2b, as defined in Example 4.2, to find the steady-state probabilities for the 4 states, system availability (which you define), and system performability.

4.16 Quintuplicated system with repair

Perform an analysis similar to that of Example 4.8 for a system with 5-way voting on the outputs of identical modules, in which a disagreeing module and one of the good modules are removed from service and the system switches to TMR operation while the bad module is being repaired. If before the repair of the first failed module has been completed, one of the three operating modules fails, the good module that was taken out of service is switched back in and the system continues operating in TMR mode. Assume perfect failure detection and switching.

4.17 Fail-soft systems with imperfect coverage

We solved the Markov model of Example 4.9 in the special case of $\lambda_2 = 2\lambda_1$ and $\mu_2 = \mu_1$. Derive a solution for the general case, without the latter assumptions.

4.18 Fail-soft system with imperfect coverage

As suggested at the end of Example , the notion of coverage can be applied to repair transitions as well, so as to model imperfect repair or the inability of the system to reincorporate a repaired module into its operation. Show how this might be done and solve the resulting Markov model.

4.19 Birth-and-death processes

The $M/M/1$ queue is a special case of the $M/M/s$ queue discussed in Example 4.13. The $M/M/1/b$ queue assumes a limited buffer size b, so that arrivals or births are not accepted when the buffer is full. Solve the Markov model corresponding to this birth-and-death process with limited buffer size.

4.20 Little's theorem or law

Consider a bank or similar system in which customer arrivals and departures are modeled by a birth-and-death process with arrival rate λ and service rate μ. The average time W a customer spends in the system is the sum of the average time spent in the queue and the average service time $1/\mu$. Prove Little's theorem or law that states $L = \lambda W$, where L is the averge queue length.

4.21 Partial fraction expansion

To expand a fraction $N(s)/(s - r)^k$, which has the root r repeated k times, we write $N(s)/(s - r)^k = a_1/(s - r) + a_2/(s - r)^2 + \ldots + a_k/(s - r)^k$, where the a_j are constants to be determined. Use this method to find the partial fraction expansion of $(s + 3)/[s(s + 2)^2(s + 5)]$.

4.22 Convergence of Markov chains

In Problem 4.8, weather in the Land of Oz was modeled by a 3×3 Markov matrix and you were asked to show that, in the long run, weather is totally independent of today's weather. Another way of saying this in the general case of an $n \times n$ Markov matrix is that for k large enough, M^k converges to a matrix with identical rows $(x_0 \quad x_1 \quad \ldots \quad x_{n-2} \quad 1-x_0-x_1- \ldots -x_{n-2})$.

a. Does such a convergence occur for any Markov matrix M? If yes, explain your reasoning in full; if not, provide a counterexample.

b. Assuming that convergence does occur, outline an efficient procedure for finding the fixed long-term distribution $(x_0 \quad x_1 \quad \ldots \quad x_{n-2} \quad 1-x_0-x_1- \ldots -x_{n-2})$ from M, without having to compute many powers of M to empirically verify convergence.

4.23 Modeling of system behavior

An e-commerce Web site has the following simplified discrete-time 4-state model for customer behavior. Any customer starts in state 0. After one unit of time, the customer goes to the browse state 1. Over the next time unit, the customer goes from the browse state to the purchase state 2 with probability 0.1 and to the exit state 3 with probability 0.2. From the purchase state, the customer goes back to the browse state with probability 0.2 and to the exit state with probability 0.7. There are no transitions out of the exit state.

a. Draw a diagram for the state-space model and write down the corresponding Markov matrix M.

b. Derive the probability that a user in start state 0 at time step t will be in the exit state 3 at time step $t + 4$.

c. Derive the probability that any given customer makes a purchase before exiting.

d. Compute M^2, M^3, and M^4 and discuss their meanings. What is the limit of M^k as k tends to infinity?

e. Derive the probability that a customer makes more than one purchase (visits state 2 at lease twice).

References and Further Readings

[Ashb03] Ashbrook, D. and T. Starner, “Using GPS to Learn Significant Locations and Predict Movements Across Multiple Users,” *Personal and Ubiquitous Computing*, Vol. 7, No. 5, pp. 275-286, 2003.

[Behr00] Behrends, E., *Introduction to Markov Chains, with Special Emphasis on Rapid Mixing*, Vieweg, 2000.

[Bhad04] Bhaduri, D., “Tools and Techniques for Evaluating Reliability Trade-offs for Nano-Architectures,” MS thesis, Virginia Tech, 2004. [On-line version accessed on January 9, 2015.]

[Bolc06] Bolch, G., S. Greiner, H. de Meer, and K. S. Trivedi, *Quueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Wiley, 2006.

[Buza70] Buzacott, J. A., “Markov Approach to Finding the Failrue Times of Repairable Systems,” *IEEE Trans. Reliability*, Vol. 19, No. 4, pp. 128-134, November 1970.

[Funk07] Funk, A., J. R. Gilbert, D. Mizell, and V. Shah, “Modeling Programmer Workflows with Timed Markov Models,” http://gauss.cs.ucsb.edu/publication/ctwatch-tmm.pdf

[Ghah01] Ghahramani, Z., “An Introduction to Hidden Markov Models and Bayesian Networks,” *Int’l J. Pattern Recognition and Artificial Intelligence*, Vol. 15, No. 1, pp. 9-42, 2001.

[Grin06] Grinstead, C. M. and J. L. Snell, *Introduction to Probability*, on-line version of Chapter 11, entitled “Markov Chains,” pp. 405-470, accessed on January 9, 2015. http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter11.pdf

[IASU15] Iowa State Univeristy, “HIMAP Reliability Analysis Software,” accessed on January 9, 2015. http://ecpe.ece.iastate.edu/dcnl/Tools/tools_HIMAP.htm

[Kuo03] Kuo, W. and M. J. Zuo, *Optimal Reliability Modeling: Principles and Applications*, Wiley, 2003.

[Malh94] Malhotra, M. and K. S. Trivedi, “Power-Hierarchy of Dependability Model Types,” *IEEE Trans. Reliability*, Vol. 43, No. 3, pp. 493-502, September 1994.

[PTCW20] PTC Windchill (formerly Relex), https://www.ptc.com/en/products/windchill

[Puki98] Pukite, J. and P. Pukite, *Modeling for Reliability Analysis: Markov Modeling for Reliability, Maintainability, Safety, and Supportability Analyses of Complex Computer Systems*, IEEE Press, 1998.

[Reib91] Reibman, A. L. and M. Veeraraghavan, “Reliability Modeling: An Overview for System Designers,” *IEEE Computer*, Vol. 24, No. 4, pp. 49-57, April 1991.

[Reli20] ReliaSoft, “Reliability Analysis and Management Software,” Company Web site: https://www.reliasoft.com/products

[Sahn96] Sahner, R. A., K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*, Kluwer, 1996.

[Shoo02] Shooman, M. L., *Reliability of Computer Systems and Networks*, Wiley, 2002.

[UVA20] University of Virginia, “Galileo Reliability Analysis Software,” http://www.cs.virginia.edu/~ftree/

[Wiki20] Wikipedia, “Laplace Transform,” http://en.wikipedia.org/wiki/Laplace_transform

[Wiki20a] Wikipedia, “Partial Fraction Decomposition,” http://en.wikipedia.org/wiki/Partial_fraction