An Introduction to Markov Modeling: Concepts and Uses

Mark A. Boyd
NASA Ames Research Center
Mail Stop 269-4
Moffett Field, CA 94035

email: mboyd@mail.arc.nasa.gov

Summary and Purpose

Markov models are useful for modeling the complex behavior associated with fault tolerant systems. This tutorial will adopt an intuitive approach to understanding Markov models (allowing the attendee to understand the underlying assumptions and implications of the Markov modeling technique) without highlighting the mathematical foundations of stochastic processes or the numerical analysis issues involved in the solution of Markov models. This introduction to Markov modeling stresses the following topics: an intuitive conceptual understanding of how system behavior can be represented with a set of states and inter-state transitions, the characteristics and limitations of Markov models, and when use of a Markov model is and is not preferable to another type of modeling technique. Homogeneous, non-homogeneous and semi-Markov models will be discussed with examples. Use of Markov models for various comparatively sophisticated modeling situations that are commonly found in state-of-the-art fault-tolerant computing systems will also be discussed (specifically: repair, standby spares, sequence dependent behavior, transient and intermittent faults, imperfect fault coverage, and complex fault/error handling) with simple examples to illustrate each modeling situation covered.

This tutorial will be aimed at systems engineers/project leads/managers who need to include reliability or availability considerations in their design decisions, and who consequently would benefit from an intuitive description of what Markov modeling could do for them (in terms of what types of system behaviors can be captured and why they might want to use Markov modeling rather than some other modeling technique) to aid in designing/evaluating their systems. It will be assumed that the audience will have a background in undergraduate mathematics (calculus and elementary probability); previous exposure to Markov processes and elementary reliability/availability modeling will be helpful but not essential, and will not be assumed.

Mark A. Boyd

Mark A. Boyd is a research scientist in the Information Sciences Directorate at NASA Ames Research Center. He was awarded a BA in Chemistry from Duke University in 1979, an MA in Computer Science from Duke University in 1986, and a Ph.D. in Computer Science from Duke University in 1991. His research interests include mathematical modeling of fault tolerant computing systems and the development of reliability modeling tools. He is a member of the IEEE and the ACM.

Table of Contents

Introduction

Markov modeling is a modeling technique that is widely useful for dependability analysis of complex fault tolerant systems. It is very flexible in the type of systems and system behavior it can model. It is not, however, the most appropriate modeling technique for every modeling situation. The first task in obtaining a reliability or availability estimate for a system is selecting which modeling technique is most appropriate to the situation at hand. A person performing a dependability analysis must confront the question: is Markov modeling most appropriate to the system under consideration, or should another technique be used instead? The need to answer this gives rise to other more basic questions regarding Markov modeling: what are the capabilities and limitations of Markov modeling as a modeling technique? How does it relate to other modeling techniques? What kind of system behavior can it model? What kinds of software tools are available for performing dependability analyses with Markov modeling techniques? These questions and others will be addressed in this tutorial.

1

Intended Audience

- Engineers, managers, students, etc., with an interest in modeling systems for reliability
- Light or no background in modeling, reliability, or probability theory
- Could benefit from an intuitive presentation of Markov modeling:
  - How Markov models represent system behavior
  - Types of system behavior that can be represented
  - Why use Markov models rather than some other type of model?
  - Differences between the 3 types of Markov models

Slide 1

Slide 1: Intended Audience

The purpose of this tutorial is to provide a gentle introduction to Markov modeling for dependability (i.e. reliability and/or availability) prediction for fault tolerant systems. The intended audience are those persons who are more application oriented than theory oriented and who have an interest in learning the capabilities and limitations of Markov modeling as a dependability analysis technique. This includes engineers responsible for system design, managers responsible for overseeing a design project and for ensuring that dependability requirements are met, students studying engineering or dependability analysis, and others who have a need or interest to be familiar with the use of Markov models for dependability analysis. The audience will be assumed to familiar with calculus and elementary concepts of probability at no more than an undergraduate level. Beyond that, little or no background in modeling, dependability, or probability theory will be assumed on the part of the audience. In short, this tutorial is intended for anyone who could benefit from an intuitive presentation of the basics of Markov models and their application for dependability analysis.

Outline

2

Introduction

- Role of reliability/availability modeling in system design and validation
- Place of Markov models in the spectrum of modeling methods

Basics of Markov Models

- How Markov models represent system behavior:
  - states
  - transitions
- 3 types of Markov models:
  - Homogeneous
  - Non-homogeneous
  - Semi-Markov
- Example model: Hypercube Multiprocessor

  How different modeling assumptions give rise to different types of Markov models

Slide 2

Outline (cont)

3

Uses of Markov Models for Dependability Analysis

- Major advantages and disadvantages of Markov modeling
- How Selected System Behaviors can be Modeled with Markov Models:
  - Complex Repair
  - Standby Spares (Hot, Warm, Cold)
  - System has Sequence Dependent Behavior
  - System is subject to Transient/Intermittent Faults
  - System has complex Imperfect Coverage of Faults
  - System has complex Fault/Error Handling and Recovery

Additional Issues

- Model generation and validation
- Stiffness
- State space size - state reduction techniques

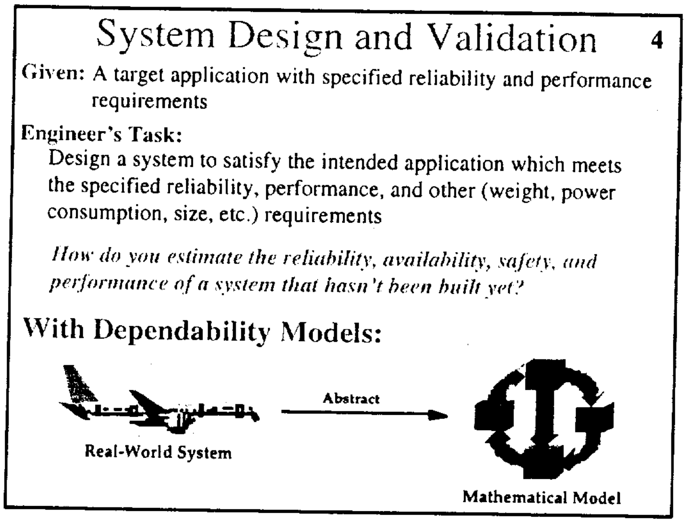
Selected Software Tools for Markov Modeling

Summary and Conclusion

Slide 3

Slides 2 & 3: Outline of Tutorial

This tutorial will be organized in the following way: we will begin with a discussion of the role that reliability modeling in general plays in system design and validation and the place that Markov modeling in particular occupies within the spectrum of the various modeling techniques that are widely used. We will then offer an intuitive description of generic Markov models and show how they can represent system behavior through appropriate use of states and inter-state transitions. Three types of Markov models of increasing complexity are then introduced: homogeneous, non-homogeneous, and semi-Markov models. An example, consisting of a fault-tolerant hypercube multiprocessor system, is then offered to show how different assumptions regarding system characteristics (such as component failure rates and standby spare policy) translate into different types of Markov models. This is followed by a discussion of the advantages and disadvantages that Markov modeling offers over other types of modeling methods, and the consequent factors that would indicate to an analyst when and when not to select Markov modeling over the other modeling methods. Next, a series of slides is presented showing how selected specific system behaviors can be modeled with Markov models. We then discuss some additional issues arising from the use of Markov modeling which must be considered. These include options for generating and validating Markov models, the difficulties presented by stiffness in Markov models and methods for overcoming them, and the problems caused by excessive model size (i.e. too many states) and ways to reduce the number of states in a model. Finally, we provide an overview of some selected software tools for Markov modeling that have been developed in recent years, some of which are available for general use.



System Design and Validation 4

Given: A target application with specified reliability and performance requirements

Engineer's Task:
Design a system to satisfy the intended application which meets the specified reliability, performance, and other (weight, power consumption, size, etc.) requirements

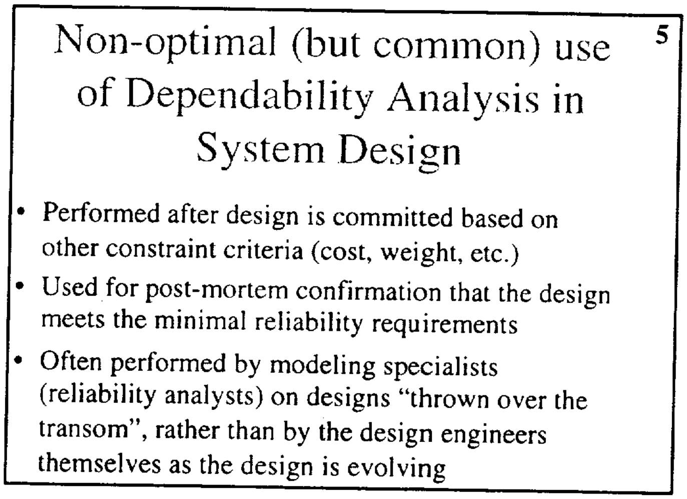
How do you estimate the reliability, availability, safety, and performance of a system that hasn't been built yet?

With Dependability Models:

Slide 4

Slide 4: Role of Dependability Modeling in System Design and Validation

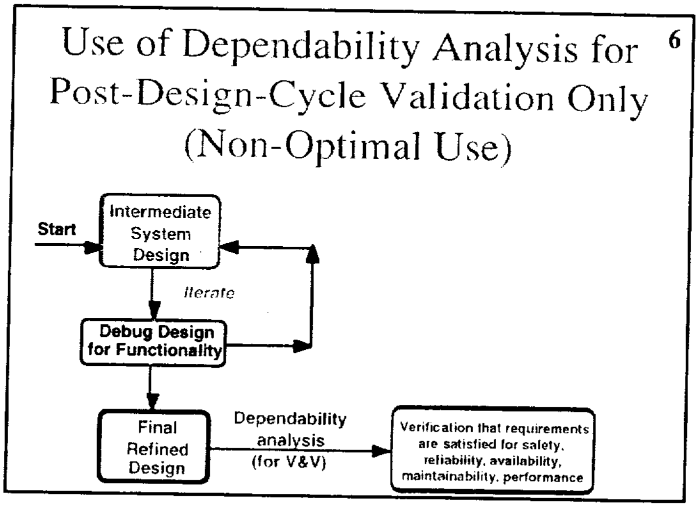
The process of designing and building a system often begins when a team of design engineers is presented with a target application by an outside agency (for example, NASA, the DoD, or a commercial customer) or by their management. This target application may have specified dependability and performance requirements, particularly if the application is a safety-critical system (***dependability*** is an umbrella term which encompasses reliability, availability, safety, etc.[1]). The engineers' task then is to design a system (or subsystem) which satisfies the requirements of the application (including function, performance, and dependability) while simultaneously adhering to other constraints such as limits on weight, power consumption, physical size, etc. The required function may be able to be satisfied by any one of a number of different designs, each of which may have different characteristics. Typically it is desirable to maximize performance and dependability while minimizing cost, weight, size, and power. Characteristics like cost, weight, and power are relatively easy to predict for a given design because they tend to be straightforward functions of the numbers and properties of the individual components used to construct the overall system. Performance and dependability are more difficult to predict because they depend heavily on the configuration in which the components are arranged. They may also depend on the work load imposed on the system and the environment in which the system operates. Short of actually building each proposed design and observing the performance and dependability from real-life experience (an option which is impractical), the system designers need tools with which to predict the performance and dependability of their candidate designs and assist them in selecting which design to actually build.



Non-optimal (but common) use of Dependability Analysis in System Design 5

- Performed after design is committed based on other constraint criteria (cost, weight, etc.)
- Used for post-mortem confirmation that the design meets the minimal reliability requirements
- Often performed by modeling specialists (reliability analysts) on designs "thrown over the transom", rather than by the design engineers themselves as the design is evolving

Slide 5



Use of Dependability Analysis for Post-Design-Cycle Validation Only (Non-Optimal Use) 6

Start
Intermediate System Design
Iterate
Debug Design for Functionality
Final Refined Design
Dependability analysis (for V&V)
Verification that requirements are satisfied for safety, reliability, availability, maintainability, performance

Slide 6

Slides 5 & 6: Non-Optimal (Post-Design-Phase Only) Use of Dependability Modeling for System Design and Validation

Mathematical modeling (of which Markov modeling is one method) provides such tools that can assist in providing the needed performance and dependability predictions. Often the design process is evolutionary, proceeding in a series of iterative refinements which may give rise to a sequence of decision points for component/subsystem configuration arrangements. Subsequent decision points may depend on earlier ones. Ideally, the system designers should be able to use dependability modeling throughout the entire design process to provide the dependability predictions required to make the best configuration selections at all decision points at all levels of system refinement. Having dependability modeling tools continuously available for use on a "what-if" basis by the system designers is important because of the exploratory nature that tends to characterize human creative work.

However, in practice the use of dependability modeling in the design of systems often falls short of this ideal. Instead of playing a role as an integral part of the design process, it may be used, after the design has been selected and committed, simply as a method for providing officially recognized evidence that the design meets contractually mandated dependability requirements. In this capacity, it is often performed by modeling specialists (i.e., reliability analysts) rather than by the design engineers.

This strategy for using dependability modeling has several disadvantages. The system designers are not given the benefit of the insight into the candidate designs that dependability modeling could provide while the design is still in its formative stage. The result may be that an ***acceptable*** design might be produced which meets the specified dependability requirements, but it is less likely that the ***best*** design will be produced than if dependability modeling was used throughout the design process. The use of dependability modeling during design rather than afterward can improve the quality of the system design that is produced.

Another disadvantage arises when the dependability analysis is performed by modeling specialists rather than by the design engineers. Modeling a system requires intimate knowledge of the system and how it operates. The design engineers have this more than anyone else. For a modeling specialist to model the system, the engineers must essentially teach the modeling specialist the technical subtleties of the system. Often these fall in a technical field that is outside the expertise of the modeling specialist. The engineers may not know exactly what information is important to give to the specialist, and the specialist may not know enough to ask for all the appropriate information. The result can be that important details may be omitted from the system model, and the reliability prediction obtained from the model may not be completely accurate or appropriate. Even if the information transfer from the designers to the modeling specialist is eventually adequate, there may be delays from false starts and errors (caught and corrected) that arise during the communication and are due to the unfamiliarity of each professional with the field of the other.

Slides 7 & 8: Optimal Use of Dependability Modeling for System Design and Validation: as an Integral Part of the Systems Engineering Design Cycle

For these reasons, it is generally preferable for the design engineers themselves to do as much as possible of the initial modeling (particularly the "what-if" modeling) of their system rather than to pass the full modeling job to a modeling specialist. The engineer may consult the modeling specialist if questions arise about the modeling process. The advent of sophisticated general-use reliability modeling computer programs, which insulate the user from much of the mathematical details and subtleties involved in modeling, has helped quite a bit to grant design engineers this kind of independence to do their own modeling.

It should be noted, however, that dependability modeling is still quite a bit of an art and can involve some subtle aspects that can be overlooked or misunderstood by inexperienced modelers. This is particularly true when Markov modeling techniques are used, and is especially true when performing the validation analysis on the final design. Even the most recent reliability modeling programs do not yet have robust capabilities for guarding against inadvertent application of inappropriate modeling techniques. For this reason it is wise for a design engineer to have a modeling specialist review any dependability model upon which important design decisions depend. Hopefully this double-checking will be less important as dependability modeling computer programs develop more sophisticated checking capabilities. But the current state of the art for these programs makes it still prudent to include a modeling specialist in the loop in this verification role.

Optimal Use of Dependability Analysis in System Design

7

Dependability Modeling should be an *integral* part of the System Design Process:

- Used throughout the design process at all levels of system evolution and refinement
- Used on a "what-if" basis by the Design Engineers to compare competing design options

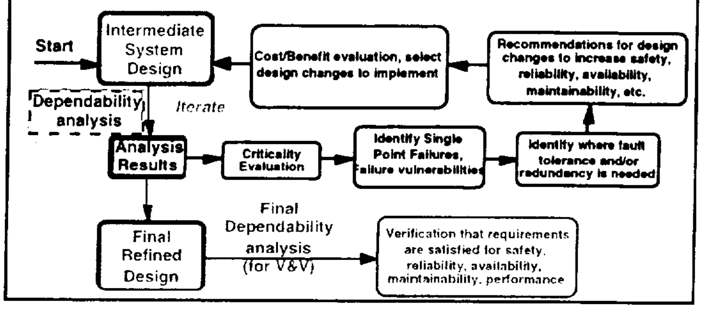
Benefits:

- When modeling is done by the Design Engineers as much as possible:
  - Reduces delays and errors due to communication problems between the Design Engineers and the Modeling Specialists
  - Can help the Design Engineers gain new insights into the system and understand it better
- Can help produce not just a minimally acceptable design, but the *best* design possible

Slide 7

8

Integration of Dependability Analysis into the Systems Engineering Design Process



Start

Intermediate System Design

Cost/Benefit evaluation, select design changes to implement

Recommendations for design changes to increase safety, reliability, availability, maintainability, etc.

Dependability analysis

Iterate

Analysis Results

Criticality Evaluation

Identify Single Point Failures, Failure vulnerabilities

Identify where fault tolerance and/or redundancy is needed

Final Refined Design

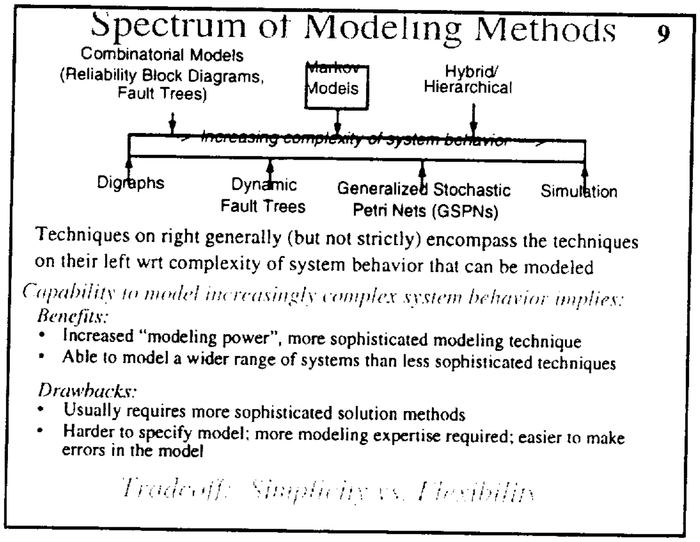
Final Dependability analysis (for V&V)

Verification that requirements are satisfied for safety, reliability, availability, maintainability, performance

Slide 8

Slide 9: The Place of Markov Modeling in the Spectrum of Modeling Methods

The range and capabilities of available methods for mathematical modeling have increased greatly over the last several decades. A dependability analyst has a full spectrum of methods from which to choose. Generally, the best strategy is to match the modeling method to the characteristics and required level of detail in the behavior of the system that must be modeled. It is important to select the simplest modeling method that will suffice. For this reason, it is helpful to have a knowledge of the characteristics, capabilities, and limitations of all modeling methods in the spectrum. While obtaining a thorough knowledge of all of this would be very time-consuming, it is possible to make a good selection with only a general working familiarity with the various modeling methods. The spectrum (depicted in the slide) extends from the simplest types of models on the left up through the most complex on the right. The more complex techniques on the right generally encompass[1] the simpler techniques on the left with respect to the range of system behavior that can be modeled. The further to the right a modeling technique appears in the diagram, the more sophisticated it tends to be relative to those to its left, and the wider the range of system behavior it can model. This capability is not without cost, however. The more sophisticated a method is, the more sophisticated the evaluation technique required to solve the model usually is. This occasionally means that solving the model will require more execution time than that needed to solve a simpler model of comparable size. Also, the more complex the modeling technique the harder it usually is for the user to specify the model, and the easier it is to make errors during model construction. In addition, it is generally more difficult to use the more sophisticated modeling techniques correctly, requiring greater modeling knowledge and experience on the part of the analyst. In summary, the decision of which modeling technique to use to model a system involves ***a tradeoff of simplicity vs. flexibility***. This fact provides the motivation to use the simplest modeling technique that suffices to model the system for the required level of detail.



Spectrum of Modeling Methods 9

Combinatorial Models (Reliability Block Diagrams, Fault Trees)

Markov Models

Hybrid/ Hierarchical

Increasing complexity of system behavior

Digraphs

Dynamic Fault Trees

Generalized Stochastic Petri Nets (GSPNs)

Simulation

Techniques on right generally (but not strictly) encompass the techniques on their left wrt complexity of system behavior that can be modeled

Capability to model increasingly complex system behavior implies:

Benefits:

- Increased "modeling power", more sophisticated modeling technique
- Able to model a wider range of systems than less sophisticated techniques

Drawbacks:

- Usually requires more sophisticated solution methods
- Harder to specify model; more modeling expertise required; easier to make errors in the model

Tradeoff: Simplicity vs. Flexibility

Slide 9

The leftmost modeling techniques appearing in the spectrum shown in the slide are the combinatorial modeling techniques, *digraphs* and *fault trees*[2] (included with fault trees are similar techniques like reliability block diagrams). These techniques model the system by expressing system behavior in terms of the *combinations* of individual events (for example, component failures) which cause the system to fail (for failure space models) or to operate correctly (for success space models). Models of these types are usually the easiest to construct and solve compared to the other more complex techniques. However, they are relatively limited in the types of system behavior they can model compared to the other techniques. More complex are *dynamic fault trees*[3, 4], which are a generalization of traditional fault trees that allow sequence dependent system behavior to be included in the model (sequence dependent behavior is behavior that depends in some way on the order in which events occur). Next on the scale are the Markov modeling techniques which are the topic of this tutorial. In addition to being able to model much of the combinatorial and sequence dependent behavior that the previous model types can, they can model a wide range of behavior that arises from many techniques used in present state-of-the-art fault tolerant systems, including the use of complex repair strategies, dynamic reconfiguration using spares, and complex fault/error recovery procedures that are not always perfectly effective. Next are hybrid and hierarchical modeling techniques. These essentially provide methods for combining models of the types already mentioned together into larger models. At the top of the scale is simulation. Simulation provides the ability to capture the most detailed system behavior of all the other modeling techniques, but at a cost of greater relative difficulty in constructing and validating the model, and also much greater execution time required to obtain highly accurate evaluations of the model.

The reader may note that Markov modeling techniques are approximately midway along the complexity spectrum of modeling techniques, and this indicates their place relative to the other modeling techniques. However, the reader should be cautioned that the spectrum in the slide is not to scale with respect to an absolute measure of modeling complexity and sophistication, and moreover the reference to Markov models itself represents several modeling techniques which cover a range of system behavior. These Markov modeling techniques will be discussed in the remainder of this tutorial.

Basics of Markov Models

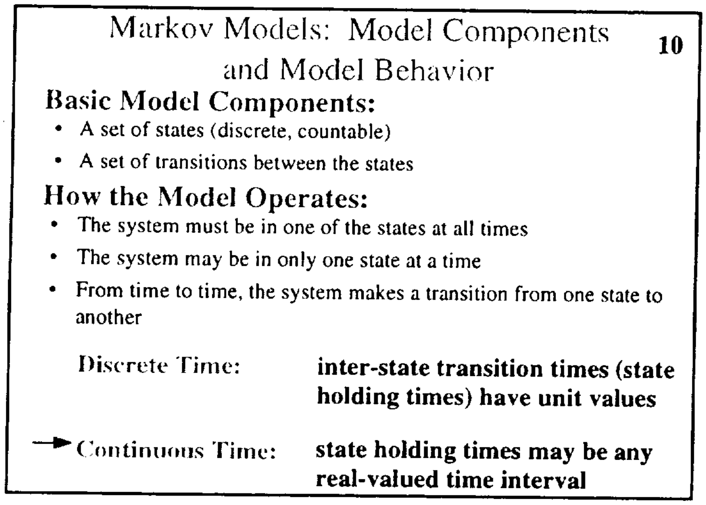
A discussion of Markov modeling begins with the basic components of Markov models: states and transitions. Also to be considered are the topics of how the states and transitions are used to express system behavior, what ``solving" a Markov model involves, and how reliability/availability estimates may be obtained from the solution of a Markov model. In addition, it is important to know the advantages and disadvantages of Markov modeling compared to other modeling techniques, and when Markov modeling is and is not preferred over other modeling techniques.

Slide 10: Markov Models - Basic Model Components and Behavior

There are two basic components common to all of the Markov models discussed in this tutorial: a set of ***states***, and a set of ***transitions*** between the states. The models considered here are limited to those having a countable number

[1] This is not to say that the more complex techniques on the right strictly encompass the simpler techniques on the left; it is only a general tendency. There are cases where a technique can model a certain type of system behavior that a technique farther to the right cannot, or can model only awkwardly. For example, digraphs are the only modeling technique in the spectrum that can model fault propagation elegantly. As a further example, combinatorial models can model system behavior which is combinatorial in nature but for which component lifetimes have a general (i.e. non-exponential) distribution; Markov models cannot model this type of system behavior at all.

(possibly infinite) of states. The model operates in the following way: the system is envisioned as being in one of the states at all times throughout the time period of interest. The system can be in only one state at a time, and from time to time it makes a transition from one state to another state by following one of the set of inter-state transitions. There are two types of models that can be considered at this point, depending on how the transitions are permitted to occur in the time domain. If the transitions are restricted to occur only at fixed, unit time intervals with a transition required at each interval, then the model is called a *Discrete Time Markov Chain (DTMC)*. If, however, this restriction is relaxed and the transitions are permitted to occur at any real-valued time interval, the model is called a *Continuous Time Markov Chain (CTMC)*. The time between transitions is called the *state holding time*. This tutorial will be concerned only with the latter type, i.e. CTMCs.



Markov Models: Model Components and Model Behavior

10

Basic Model Components:

- A set of states (discrete, countable)
- A set of transitions between the states

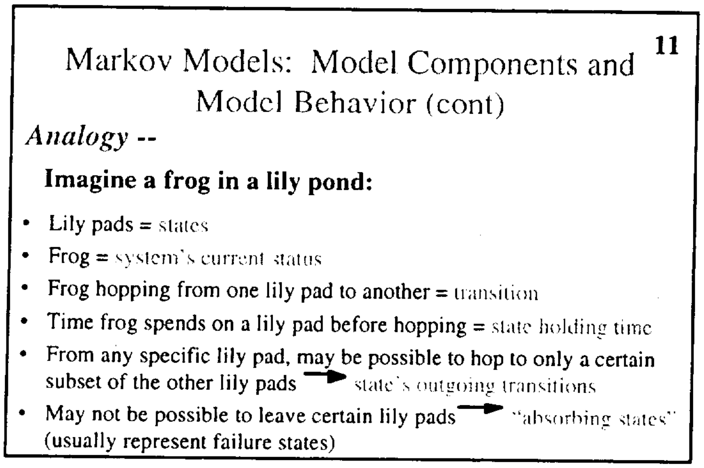
How the Model Operates:

- The system must be in one of the states at all times
- The system may be in only one state at a time
- From time to time, the system makes a transition from one state to another

Discrete Time: **inter-state transition times (state holding times) have unit values**

→ Continuous Time: **state holding times may be any real-valued time interval**

Slide 10



Markov Models: Model Components and Model Behavior (cont)

11

Analogy --

Imagine a frog in a lily pond:

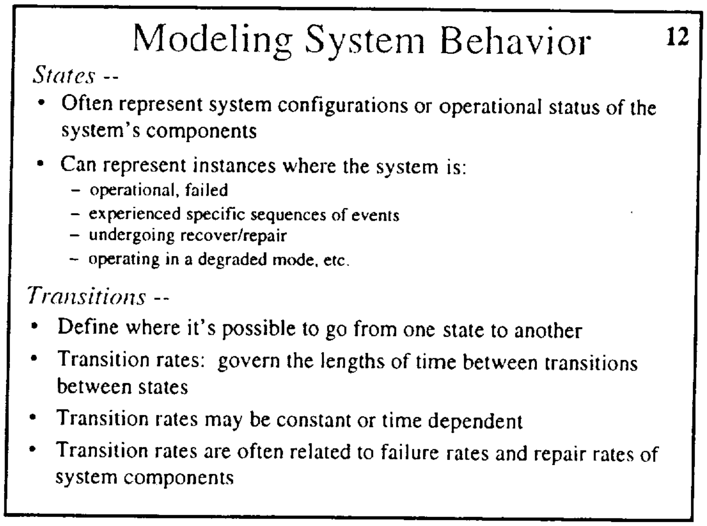
- Lily pads = states
- Frog = system's current status
- Frog hopping from one lily pad to another = transition
- Time frog spends on a lily pad before hopping = state holding time
- From any specific lily pad, may be possible to hop to only a certain subset of the other lily pads → state's outgoing transitions
- May not be possible to leave certain lily pads → "absorbing states" (usually represent failure states)

Slide 11

Slide 11: Markov Models - A Simple Analogy

An analogy may help with envisioning how the Markov model works: imagine a frog in a lily pond where he is free to hop among the lily pads in the pond, and with the further provision that he never falls into the water[5]. The lily pads in the pond correspond to states in a Markov model. The frog corresponds to the system's current status or state of being.

The frog hopping from one lily pad to another corresponds to the system making a transition from one state to another in the Markov model. The time that the frog spends sitting on a lily pad before making a hop corresponds to the state holding time. From any specific lily pad, the frog may be able to hop to only a specific subset of the other lily pads in the pond (some may be too far away, some may have a log or other obstacle barring the way). The lily pads to which hopping is possible correspond to the set of outgoing transitions each state has that specify which other states are directly reachable from the given state. In the pond there may be some lily pads from which the frog cannot leave once he hops there. These correspond to what are called ***absorbing states*** in a Markov model. These states usually correspond to system failure states in a Markov model of a system.



Modeling System Behavior

12

States --

- Often represent system configurations or operational status of the system's components
- Can represent instances where the system is:
  - operational, failed
  - experienced specific sequences of events
  - undergoing recover/repair
  - operating in a degraded mode, etc.

Transitions --

- Define where it's possible to go from one state to another
- Transition rates: govern the lengths of time between transitions between states
- Transition rates may be constant or time dependent
- Transition rates are often related to failure rates and repair rates of system components

Slide 12

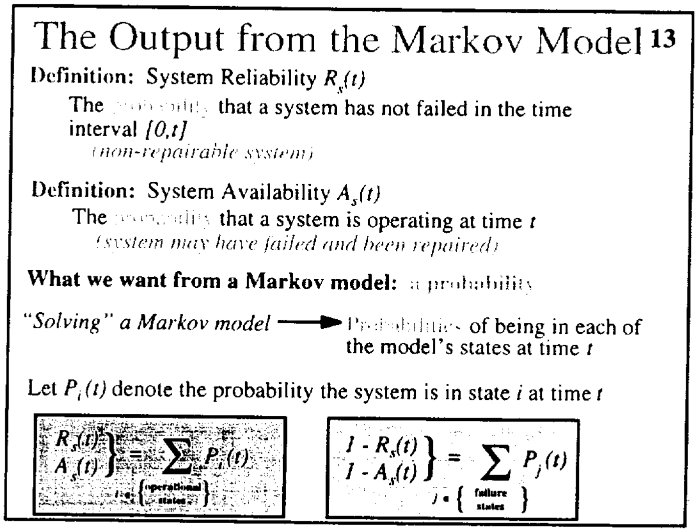
Slide 12: Markov Models - Modeling System Behavior

When Markov models are used as dependability models of systems, the states frequently represent some configuration or functional status of the system. They can actually represent almost anything, but usually they represent something like an enumeration of the components that are working and/or failed in the system. Often the construction of a Markov model begins with a simple representation for the states like this, and then additional criteria or characteristics that need to be represented are added to the state meanings. A state can represent situations such as instances where the system is operational, failed, undergoing recovery or repair, operating in a degraded mode, having experienced some specific sequence of events, etc.

The transitions define where it's possible to go directly from one state to another. The transitions are labeled in various ways depending on the type of model and the convention being used. A common practice used for reliability modeling is to label each transition with a ***transition rate*** which governs the length of the time that elapses before the system moves from the originating state to the target state of the transition (the state holding time). These transition rates may be either constant or functions of time, and they often are related to the collective influence of failure and repair rates of individual components on the transition between states.

Slide 13: The Output From the Markov Model

The *reliability*, $R_s(t)$ of a system after an elapsed mission time t may be defined as *the probability that the system has not failed at any time* from the start of the mission at time $t = 0$ until time t. Reliability is usually the measure of interest for non-repairable systems because failures for such systems are permanent for the remainder of the mission. Markov models for such systems have no cycles (i.e. are *acyclic*). For systems for which repair of failed components or subsystems is possible, the measure that is most frequently of interest is the system *availability* at time t, $A_s(t)$. System availability may be defined as the probability that the system is operating at time t. Note that this definition admits the possibility that the system may have failed one or more times since the beginning of the mission and has been repaired. Repair is represented in the Markov model of such systems by the presence of cycles depicting the loss and then restoration of the functionality of a component or subsystem. The term *dependability* encompasses both reliability and availability, and a reference to dependability as a measure will be interpreted in this tutorial to mean whichever measure (reliability or availability) is appropriate to the context of the discussion.



The Output from the Markov Model 13

Definition: System Reliability $R_s(t)$

The probability that a system has not failed in the time interval $[0,t]$

(non-repairable system)

Definition: System Availability $A_s(t)$

The probability that a system is operating at time t

(system may have failed and been repaired)

What we want from a Markov model: a probability

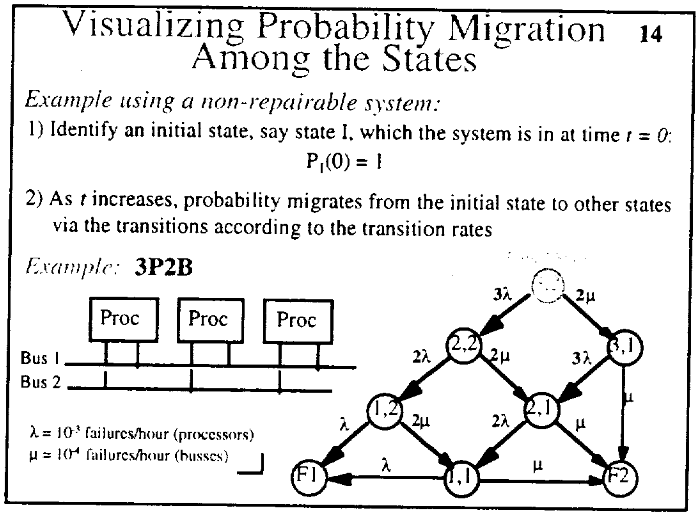
"Solving" a Markov model ⟶ Probabilities of being in each of the model's states at time t

Let $P_i(t)$ denote the probability the system is in state i at time t

$$\left.\begin{matrix} R_s(t) \\ A_s(t) \end{matrix}\right\} = \sum_{i \in \left\{\substack{\text{operational}\\\text{states}}\right\}} P_i(t) \qquad \left.\begin{matrix} 1 - R_s(t) \\ 1 - A_s(t) \end{matrix}\right\} = \sum_{j \in \left\{\substack{\text{failure}\\\text{states}}\right\}} P_j(t)$$

Slide 13

These definitions indicate that, whatever the measure of interest, the desired output of an evaluation of a Markov dependability model is a numeric value which is a probability. It happens that the process of "solving" a Markov model produces as output the probabilities of being in each of the states of the model at time t (for transient analysis). Since the events of being in each state of the Markov model are mutually exclusive (the system cannot be in more than one state at a time) and collectively exhaustive (the system always must be in at least one of the states), it follows that the sum of the probabilities of being in any subset of the Markov model's states is also a valid probability. The states of any Markov model that models a system may be partitioned into two sets: one set containing states that represent situations where the system is operating correctly (either with full functionality or in some type of degraded mode), and the other set containing states that represent situations where the operation of the system has degraded so much that the system must be considered failed. The reliability/availability of the system may then be taken to be the sum of the probabilities of being in one of the operational states at time t, and the complement (unreliability or unavailability) is the sum of the probabilities of being in one of the failure states at time t.



Visualizing Probability Migration Among the States 14

Example using a non-repairable system:

1) Identify an initial state, say state I, which the system is in at time $t = 0$:

$$P_I(0) = 1$$

2) As t increases, probability migrates from the initial state to other states via the transitions according to the transition rates

Example: **3P2B**

$\lambda = 10^{-3}$ failures/hour (processors)

$\mu = 10^{-4}$ failures/hour (busses)

Slide 14

Slide 14: Visualizing Probability Migration Among the States

As a mission progresses, the system's dependability behavior is reflected in the probabilities of being in each of the states in the Markov model. The probabilities of being in the states change over time and reflect the expected behavior of the system over a very large number of missions. A useful device to aid in visualizing the changing of the state probabilities over time is to imagine the probability as a limited quantity of a fluid material such as a gas, the states as receptacles, and the transitions as unidirectional pipes through which the gas can diffuse. Often a Markov model of a system will contain a single initial state which represents a fully operational system. At the start of the mission all the probability (gas) is contained in the initial state. As time progresses, the probability migrates from this initial state to other states in the model, as a gas might diffuse, through the transitions at a rate determined by the transition rates that label the transitions. This analogy is not exact, since the gas diffusion model does not take into account some of the stochastic properties of the Markov model (i.e. the Markov property, etc.). However, the analogy is useful for visualizing a picture of what is happening to the state probabilities over time at a conceptual level.

The example shown in the slide serves to illustrate the probability migration process. Consider a system consisting of three redundant processors which communicate with each other and other components over two redundant busses. In order for the system to be operational, at least one processor must be able to communicate correctly over at least one bus. Assume also that repair of failed components is not possible during a mission. A Markov model for this system appears to the right of the system diagram in the slide. Initially, all processors and both busses are assumed to be working correctly. The initial state is labeled *{3,2}* to denote three working processors and two working busses. If a processor fails, the system moves to state *{2,2}* which denotes two working processors and two working busses. If instead a bus fails, the system moves to state *{3,1}* which denotes three working processors and one working bus. Subsequent failures cause further transitions as indicated in the Markov chain. As time progresses, probability density migrates from the initial state *{3,2}* down to the other states in the model. Since this is a non-repairable system, as $t \rightarrow \infty$ the system must eventually fail. This is represented in the model by the system eventually reaching one of the two failure states (labeled *{F1}* and *{F2}*). The relative rates of probability density migration will be consistent with the transition rates that label the transitions. For example, since the failure rate for the processors (λ) is ten times greater than the failure rate for the busses (μ), the system generally will migrate to the states on the left of the Markov chain more quickly than to the states on the right.

"Solving" the Markov Model 15

Focus on the change in probability for individual states:

change in probability for state *i* = incoming probability from all other states - outgoing probability to all other states

- System of *n* simultaneous differential equations (one for each state)
- Usually solved numerically by computer
- Solved model gives probability of the system being in state *i* at time *t*

Slide 15

Slide 15: "Solving" the Markov Model

If a dependability analyst is familiar with the stochastic properties and underlying assumptions of a Markovian modeling technique, then a thorough knowledge of the numerical methods needed for solving that type of Markovian model is generally unnecessary in order to use the modeling technique for dependability analysis in an effective way provided that the analyst has access to an appropriate software tool that can evaluate the model. For this reason, a detailed discussion of the methods for solving Markov models is beyond the scope of this tutorial. It is useful, however, to be aware of how certain limitations inherent in the solution techniques may affect the construction of the model and influence the type of system behavior that can feasibly be represented in the model. We will touch on this topic later in the tutorial. For now, it is helpful to give, in very general terms, a brief description of what is done to "solve" a Markov model.

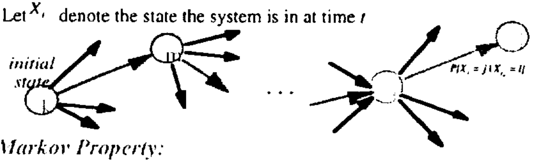
The previous slide showed how probability density migrates among the states in the Markov model over time. The key element in finding a procedure for determining the probability of the individual states at a particular point in time is to focus on the change in probability with respect to time for each state *i*. Intuitively, the change in the probability for a given state is simply the difference between the amount of probability coming into the state from all other states and the amount of probability going out of the state to other states in the model. This is expressed in terms of a differential equation which includes terms consisting of products of transition rates with state probabilities. The result is an $n \times n$ system of simultaneous differential equations (one differential equation for each state). The solution of this system of differential equations is a vector of state probabilities at the specified time *t*. The solution of the differential equations is usually done numerically with a computer.

Slide 16: The Markov Property: Blessing and Curse

A fundamental property that is shared in one form or another by all the Markovian models discussed in this tutorial is the "Markov property". This property is really a simplifying assumption. In the most general discrete-state stochastic process, the probability of arriving in a state *j* by a certain time *t* depends on conditional probabilities which are associated with sequences of states (paths) through which the stochastic process passes on its way to state *j*. It also depends on the times $t_0 < t_1 < \ldots < t_n < t$ at which the process arrives at those intermediate states. A complete accounting for all possible paths and all possible combinations of times would be very complex and usually is not feasible. The problem of evaluating all of the state probabilities in the resulting stochastic process generally is not tractable. The Markov property allows a dramatic simplification both in the defining of the stochastic process (i.e. the specification of the conditional probabilities) and in the evaluation of the state probabilities. It does this by allowing one to assume that the probability of arriving in a state *j* by a time *t* is dependent only on the conditional probabilities of the transitions into state *j* from states immediately preceding state *j* on the transition paths and not on all the transitions along each entire path to state *j*. Another way of saying this is that the future behavior of the simplified stochastic process (i.e. Markov model) is dependent only on the present state and not on how or when the process arrived in the present state.

The Markov Property: Blessing and Curse 16

Let X_t denote the state the system is in at time *t*



Markov Property:

For all times $t_0 < t_1 < \ldots < t_n < t$

$$P[X_t = j \mid X_{t_0} = k, X_{t_1} = m, \ldots, X_{t_n} = i] = P[X_t = j \mid X_{t_n} = i]$$

Transition to any next state *j depends only on the present state i* - NOT on the previous transition path that arrived at the present state i

Advantage:

- *Simplifying assumption:* dramatically simplifies both the job of specifying the transition probabilities ~~and~~ the mathematics of evaluating the Markov model
- Helps make evaluation of the Markov model tractable

Drawback:

- Assumption is very restrictive and may not be valid for many real-world systems - *the analyst must take care!*
- If the assumption is not reasonably valid for a system, can't model the system with Markov models (won't get a meaningful result), and another modeling technique needs to be used

Slide 16

The great benefit of the Markov property is that it helps make the evaluation of Markovian models tractable. It is something of a mixed blessing, however, in that it is a very restrictive assumption that is not always consistent with the reality of real-world system behavior. Real systems do tend to have their future behavior depend in various ways on what they have experienced in the past. There are common situations where the Markov property is even counter-intuitive. As an example, consider a situation where a component in a system breaks down and is repaired on the fly during some mission. If this is modeled with a Markov model for which the failure rates of the components are assumed to be constant, the underlying assumption derived from the Markov property is that the repaired component must end up being "as good as new" after the repair and from then on behaves just like a brand new component, regardless of how long the component had been in service before it failed and regardless of how much environmental stress it experienced. There are many situations where this is just not an accurate representation of reality. A dependability analyst using Markov models must be aware of the implications of the Markov property and always keep in mind the limitations it places on the system behavior that can be modeled with Markov models. As with any modeling technique that relies on underlying assumptions, if the assumptions are too inconsistent with the characteristics of the real system, then any dependability estimates for the system obtained from the model are not meaningful and cannot be used to represent or predict the behavior of the real system.

Three Types of Markov Models

We now introduce three types of Markov models that will be described in this tutorial.

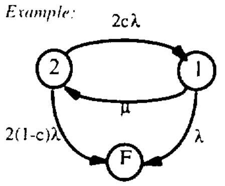
Slides 17 & 18: 3 Types of Markov Models

The simplest and most commonly used Markov model type is the ***homogeneous Continuous Time Markov Chain (CTMC)***. For this type of Markov model, the "Markov" property holds at all times. Recall that, intuitively, the Markov property states that the selection of the transition to the next state, and indeed all future behavior of the model, depends only on the present state the system is in and not on the previous transition path that led the system to be in the present state. In terms of the frog-hopping analogy, this can be described by stating that the lily pad that the frog next decides to hop to depends solely on which lily pad he is presently sitting on and not on the sequence of lily pads he visited before arriving on the present one, nor even on whether he has ever visited the present lily pad before. A second property is that the state holding times in a homogeneous CTMC are exponentially distributed and do not depend on previous or future transitions. To say that a state holding time is exponentially distributed means that, if the system is in state i at time τ, the probability that the next transition (leading out of state i) will occur at or before a time t units in the future (say at time $\tau + t$) is given by $1 - e^{-\lambda_i t}$ (or, conversely, the probability that the next transition will occur at or after time $\tau + t$ is given by $e^{-\lambda_i t}$, where λ_i is the sum of all the rates of the transitions going out from state i). The second property says that this is true for <u>*all*</u> states in the CTMC. In terms of the frog analogy, the second property says that the length of time the frog sits on a lily pad is exponentially distributed. Furthermore, the length of time the frog sits on the lily pad is the same regardless of: the sequence of lily pads the frog followed in order to arrive at the present one, the amount of time it took to get to the current lily pad, and which lily pad he hops to next. A third property, which is a consequence of the exponentially distributed holding times, is that interstate transition rates are all constant[6]. A fourth property, which is also a consequence of the exponentially distributed state holding times, is that the time to the next transition is not influenced by the time already spent in the state. This means that, regardless of whether the system has just entered state i or has been in state i for some time already, the probability that the next transition will occur at or before some time t units into the future remains the same (for the frog, this means that regardless of whether he has just landed on the present lily pad or has been sitting on the lily pad for some time already, the probability that he will hop to a new lily pad at or before some time t units into the future remains the same). This property is a consequence of a property of the exponential distribution, called the "memoryless property"[6].

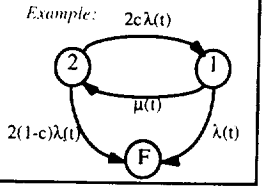
3 Types of Markov Models

17

- Homogeneous CTMCs
  - Simplest, most commonly used
  - Markov property always holds
  - Transition rates are constant
  - State holding times are exponentially distributed
  - "Memoryless Property" - Time to next transition is not influenced by the time already spent in the state



- Non-homogeneous CTMCs
  - more complex
  - Markov property always holds
  - transition rates are generalized to be functions of time - dependent on a "global clock"

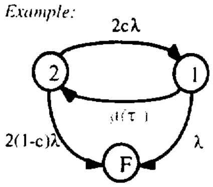


Slide 17

3 Type of Markov Models (cont)

18

- Semi-Markov Models
  - Most complex
  - Markov property only holds at certain times (i.e. only when transitions occur)
  - Transition rates can be functions of state-specific (i.e. local) clocks, not the mission time (global) clock
  - State holding times have distributions that:
    - can be general (i.e. non-exponential)
    - can depend on the next state
  - Semi-Markov models can result when detailed fault/error handling is included in a system model



Slide 18

The simple example model shown in the slide will be used to illustrate the differences between the three different Markov model types. This example is convenient because, despite having only three states, it can be used to demonstrate repair, imperfect fault coverage, and all three types of Markov models (imperfect fault coverage and repair will be discussed shortly). The example model works like this: imagine a control system consisting of two identical active processors. In the event of the failure of one of the active processors, the failure must be detected, and the faulty processor must be identified and switched off-line. The process of switching out the faulty processor is not perfectly reliable. This is represented by a probability (denoted by *c*) that the switching out reconfiguration process succeeds, and another probability (denoted by *1 - c*) that the reconfiguration process will not succeed and leads to an immediate failure of the system. Upon the failure of one of the processors, one repair person is available to fix the failed processor and return it to service. If one of the processors is being repaired and the second processor fails before the repair of the first is complete, the system will fail immediately. The diagrams at the right of the slide shows the Markov model in terms of the states and interstate transitions. The state labeled *{2}* denotes the state in which both processors are functioning correctly, the state labeled *{1}* denotes the state in which one of the two processors has failed and is undergoing repair while the other processor continues to function correctly, and the state labeled *{F}* denotes the state in which both processors have failed, causing the system itself to fail. The system begins its mission in state *{2}* with both processors functioning correctly. If one of the processors fails during the mission and the remaining processor is able to continue performing the control functions of the system successfully, the system moves to state *{1}* and repair begins on the failed processor. If the repair of the failed processor is completed, the system returns to state *{2}* with two fully operational processors. However, if the second processor fails before the repair of the first failed processor is successfully completed, the system will fail immediately and move to state *{F}*. If a processor fails while the system is in state *{2}* (both processors functioning correctly) and the remaining processor is unable to continue performing the control functions of the system (reconfiguration unsuccessful), the system also will fail immediately and move to state *{F}*.

The characteristics of the homogeneous CTMC model type may be illustrated using the example control system as follows. Let λ be the (constant) rate of failure of a processor, μ be the (constant) rate at which the repair person can repair a failed processor, and *c* be the probability that the system response to a processor failure permits it to continue operating (i.e. reconfigures successfully if necessary). Since there are two processors functioning when the system is in state *{2}*, the total rate at which failures occur and cause the system to leave state *{2}* is 2λ. When such a failure occurs, with probability *c* the system successfully reconfigures and moves to state *{1}*, so the rate that the system moves to state *{1}* is given by $2c\lambda$. Conversely, with probability *1 - c* the system is unable to reconfigure, so the rate that the system moves from state *{2}* directly to state *{F}* is given by $2(1 - c)\lambda$. Once the system arrives in state *{1}*, a subsequent failure of the other processor (which occurs at rate λ) causes the system to fail and move to state *{F}*. On the other hand, the repair person fixes the failed processor at rate μ, and if the repair is successfully completed before the other processor fails the system will return to state *{2}*. Note that when the system is in state *{2}* its behavior is the same whether it has experienced one or more failures or none at all, i.e. whether it has made a round trip to state *{1}* and back does not affect the future behavior of the system at all. It is as if the system, having experienced a processor failure and had the failed processor repaired, promptly "forgets" that the processor had ever failed. This is a consequence of the Markov property.

A ***non-homogeneous CTMC*** is obtained when a homogeneous CTMC is generalized to permit the transition rates to be functions of time as measured by a "global clock", such as the elapsed mission time, rather than requiring them to be constant. The Markov property still holds at all times for non-homogeneous CTMCs: the selection of the transition to the next state depends only on the present state the system is in and not on the previous transition path that lead the system to be in the present state. The state holding times also do not depend on previous or future transitions, as was the case for homogeneous CTMCs. In general, the transition rates may be any function of global time.

In terms of the frog analogy, the frog's situation remains the same as before except that the rates at which the frog hops may now change with time. The rate at which the frog hops may decrease the longer he spends in the pond (perhaps he is getting tired); alternatively, it may increase the longer he spends in the pond (perhaps the sun is setting and he is becoming more active as night approaches).

The example control system model may again be used to illustrate the differences between non-homogeneous CTMCs and homogeneous CTMCs in terms of the state-transition diagrams. Let λ and μ again denote the rates of processor failure and repair, respectively, except that now they are functions of the mission time. The non-homogeneous CTMC shown in the slide is the same as the one for the homogeneous CTMC, except that the transition rates are now all functions of the mission time.

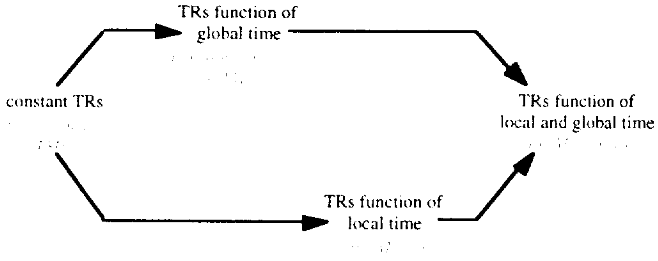
The final model type to be considered is the ***semi-Markov model***. It is the most complex of the three. It is called a *semi*-Markov model because the Markov property does not hold at all times. Rather, it holds only at the times when transitions occur. The behavior of the semi-Markov model is the same as the others in that the selection of the transition to the next state does not depend on the previous transition path that brought the system to the present state. However, it differs from the others in that the state holding times can have distributions that can be completely general (non-exponential) and also can depend on the next state. In terms of the frog analogy, the behavior of the semi-Markov model can be described as follows: the frog hops between lily pads in the pond as before. However, as soon as he lands on a new lily pad he selects the next lily pad to which he plans to hop according to the Markovian transition probabilities for that lily pad's outgoing transitions. Then, before hopping again, he waits a period of time that has a distribution that depends on which lily pad he has selected as his next one[5]. This waiting time need not be exponentially distributed ... it can have any general distribution. As a consequence of the generally distributed state holding times, the inter-state transition rates can be a function of time as measured by "local clocks". A "local clock" in this context would be a timer that begins counting the passage of time at the moment the state is entered. This is in contrast to a "global clock", which would be a timer that begins counting the passage of time at the moment that the mission begins and is independent of the time spent in any one state.

The control system example may again be used to illustrate the difference between a semi-Markov model and the previous two types of Markov models. Assume that the failure rate of a processor is again constant. Now, however, assume that the repair duration is a function of the time (τ_1) that the processor has been under repair. The state-transition diagram for the resulting semi-Markov model is shown in the slide. It is identical to that for the homogeneous CTMC case except that the repair transition rate is a function of the time τ_1 that the processor has been under repair (i.e. the time that the system has been in state *{1}*).

Semi-Markov models require the most computational effort of all the Markov model types to solve. They are often produced when detailed fault/error handling is included in a Markov model. This is the case because non-constant transitions between states that model fault handling often depend on the time elapsed since the fault occurred and handling/recovery commenced rather than on the elapsed mission time.

Relative Modeling Power of the Different Markov Model Types 19

Partial Order with respect to "Modeling Power":



TRs function of global time
constant TRs
TRs function of local and global time
TRs function of local time

--> Increasing Model Complexity and "Modeling Power" -->

Slide 19

Slide 19: Relative Modeling Power of the Different Markov Model Types

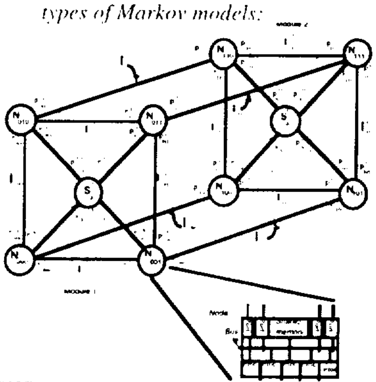
The diagram in the slide gives a pictorial image of the relationship between the various Markov model types with respect to model type complexity and modeling "power". This can be considered to be a type of partial order, with homogeneous CTMCs being the simplest and lowest on the scale of modeling power because of the requirement for constant interstate transition rates. To model more complex behavior than can be accommodated by homogeneous CTMCs, the interstate transition rates may be permitted to be nonconstant by allowing them either to be functions of global time (non-homogeneous CTMCs), or functions of state local time (semi-Markov models). Semi-Markov models can model behavior that is in some senses more complex than that which can be modeled by non-homogeneous CTMCs, and so semi-Markov models can be considered to be more sophisticated than non-homogeneous CTMCs. However, there are things that non-homogeneous CTMCs can model which semi-Markov models cannot, so semi-Markov models are not an encompassing generalization of non-homogeneous CTMCs. However, an example of a model type that *does* encompass both non-homogeneous CTMCs and semi-Markov models is one which has inter-state transition rates that are functions of global and local time both within the same model. Such a model is *non-Markovian.* Models of this type are very difficult to solve analytically (numerically) and often require more flexible evaluation techniques like simulation.

Slide 20: An Example: A Fault Tolerant Hypercube

To illustrate how differing assumptions about the characteristics of a system such as component failure rates and reconfiguration processes can translate into different Markov model types, consider the example system shown in the slide. Specifically, consider a three dimensional fault-tolerant hypercube multiprocessor whose processing nodes are themselves fault tolerant multiprocessors consisting of four active processors and a spare processor as shown in the slide[3, 7]. If the processors in all processing nodes are assumed to have constant failure rates, the resulting Markov model of the system will be a homogeneous CTMC regardless of whether the spare processors in the processing nodes are hot or cold. However, if the processors are all assumed to have either increasing or decreasing failure rates, then the resulting Markov model will be non-homogeneous *provided* the spare processors in the processing nodes are hot spares. If they are cold spares instead of hot spares, then the resulting model is no longer a Markov model. It is instead a non-Markovian model because the failure rates of the originally active processors are functions of the mission time (global clock), whereas the failure rates of any of the initially cold spare processors are functions of time measured since the processor was activated rather than the global mission time. Such a model may require simulation to evaluate[8].

Fault Tolerant Hypercube Example 20

How different modeling assumptions give rise to different types of Markov models:



Various Assumptions:

Processors have constant FRs, hot & cold spares:
homogeneous CTMCs

Processors have IFRs/DFRs hot spares:
non-homogeneous CTMCs

Processors have IFRs/DFR cold spares:
non-Markovian model

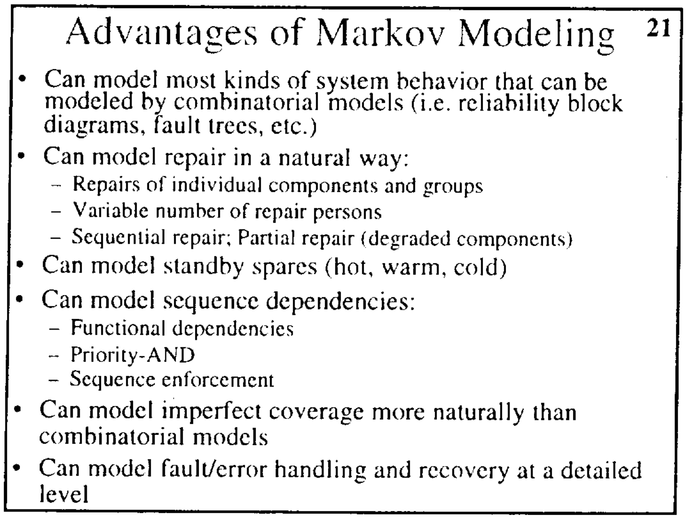
Slide 20

Use of Markov Models for Dependability Analysis

We now consider the topic when Markov modeling is an appropriate modeling technique of choice for dependability analysis, including: the advantages and disadvantages of using Markov models for dependability analysis, the types of system behavior that Markov models are well-suited to model, and when Markov modeling is <u>not</u> preferred.

Slide 21: Advantages of Markov Modeling

Compared to other modeling methods, Markov modeling offers certain advantages and disadvantages. The primary advantage lies in its great flexibility in expressing dynamic system behavior. Markov models can model most kinds of system behavior that combinatorial models can (with the exception that, because they are limited by the Markov property assumption and assumptions on the distributions of component lifetimes, they cannot model situations which can be modeled by combinatorial models with generally distributed (non-exponential) component lifetimes). In addition, Markov models can model in a natural way types of behavior which traditional combinatorial models can express only awkwardly or not at all[2]. These types of behavior include:



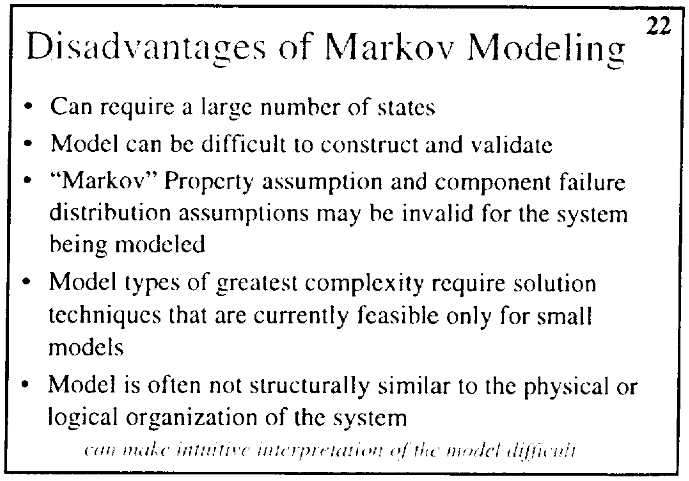
Advantages of Markov Modeling 21

- Can model most kinds of system behavior that can be modeled by combinatorial models (i.e. reliability block diagrams, fault trees, etc.)
- Can model repair in a natural way:
  - Repairs of individual components and groups
  - Variable number of repair persons
  - Sequential repair; Partial repair (degraded components)
- Can model standby spares (hot, warm, cold)
- Can model sequence dependencies:
  - Functional dependencies
  - Priority-AND
  - Sequence enforcement
- Can model imperfect coverage more naturally than combinatorial models
- Can model fault/error handling and recovery at a detailed level

Slide 21

- **Behavior involving complex repair:** This includes situations consisting of repairs of either individual components or groups of components, the presence of any number of repair persons assigned in any arbitrary way to repair activities, repair procedures that must follow a specific sequence, and any degree of partial repair (possibly resulting in subsystems or components with degraded performance).
- **The use of standby spares:** This includes hot, warm, and cold spares. Hot spares are spare units that are powered up throughout the mission and are immediately available to take over from a failed active unit, but which are also subject to failure at the same rate as the active unit. Warm spares are units which are powered up throughout the mission, but which fail at a lower rate than an active unit until called upon to take over for a failed active unit. Cold spares are units that are powered off until activated to take over for a failed active unit. Cold spares are assumed not to fail while they are powered down, but once activated can fail at the same rate as an active unit.
- **Sequence dependent behavior:** Sequence dependent behavior is behavior that depends on the sequence in which certain events occur. Examples include: *functional dependencies*, where the failure of one component may render other components unavailable for further use by the system; *Priority-AND*, where behavior differs depending on whether one event happens before or after another; and *sequenceenforcement*, where it is simply not possible for certain events to occur before certain other events have occurred.
- **Imperfect fault coverage:** Imperfect fault coverage arises when a dynamic reconfiguration process that is invoked in response to a fault or component failure has a chance of not completing successfully, leading to a single point failure of the system despite the presence of redundancy intended to survive failures of the type that has occurred. When this can happen, the fault is said to be *imperfectly covered*, and the probabilities that the system reconfiguration is successful or not are called *coverage factors*.

If detailed representation of fault/error handling is required in the model, Markov modeling techniques can easily represent such behavior also. Care should be used with this, however, because under some circumstances the inclusion of fault/error handling in the Markov model can cause numerical difficulties to arise during the solution of the model (for example, *stiffness*, which will be discussed later in this tutorial).



Disadvantages of Markov Modeling 22

- Can require a large number of states
- Model can be difficult to construct and validate
- "Markov" Property assumption and component failure distribution assumptions may be invalid for the system being modeled
- Model types of greatest complexity require solution techniques that are currently feasible only for small models
- Model is often not structurally similar to the physical or logical organization of the system

can make intuitive interpretation of the model difficult

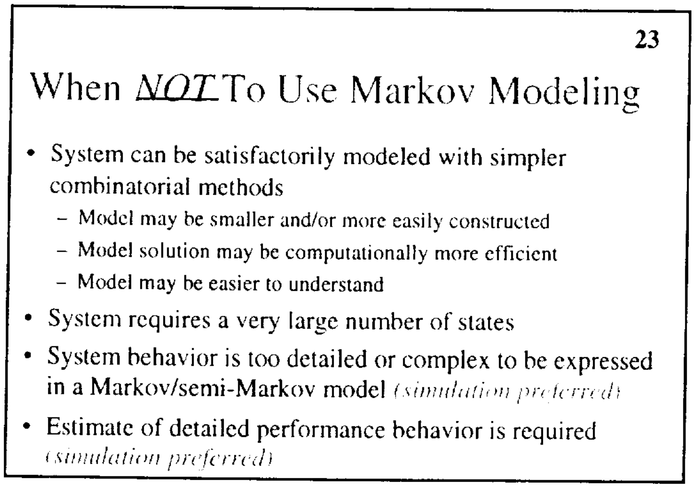
Slide 22

Slide 22: Disadvantages of Markov Modeling

Markov modeling techniques do have some disadvantages which make them not appropriate for some modeling situations. The two most important disadvantages involve state space size and model construction. Realistic models of state-of-the-art systems can require a very large number of states (for example, on the order of thousands to hundreds of thousands). Solving models with so many states can challenge the computational resources of memory and execution time offered by computers that are currently widely available. Also, the problem of correctly specifying states and inter-state transitions is generally difficult and awkward. This is especially so if the model is very large. It may be very difficult for the analyst to construct a model of a large system and verify that it is correct. Recall that the "Markov" property assumption is restrictive and may not be appropriate for many systems. If this is the case for an individual system, then Markov modeling is

[2] Recent research in fault tree modeling[3, 4, 9-12] has lead to advances that enable sequence dependency behavior, standby spares, and imperfect fault coverage to be modeled conveniently in fault trees, thereby eliminating many of the advantages that Markov modeling techniques formerly had over combinatorial models in these areas. A tutorial presented at this conference in 1996 and 1997, "New Results in Fault Trees"[13, 14] gives an overview of this work.

not an appropriate modeling technique for that system because any dependability estimate obtained from evaluating the model will not be meaningful. Of less importance, but still significant problems, are issues involving solution of the more complex types of Markov models and the form of the Markov model itself. The more sophisticated Markov model types can express much more complex system behavior than the simplest type. However, they require more complex solution techniques that require much more execution time to solve than the simplest Markov model type requires. Consequently, it is currently feasible to solve only relatively small Markov models of the more complex types. Finally, the form of the Markov model (states and transitions) often does not have much of a physical correspondence with the system's physical or logical organization. This may make it comparatively difficult for an analyst to obtain a quick intuitive visual interpretation of a model's evaluation in the same way as may be done with, for example, a digraph.



23

When *NOT* To Use Markov Modeling

- System can be satisfactorily modeled with simpler combinatorial methods
  - Model may be smaller and/or more easily constructed
  - Model solution may be computationally more efficient
  - Model may be easier to understand
- System requires a very large number of states
- System behavior is too detailed or complex to be expressed in a Markov/semi-Markov model *(simulation preferred)*
- Estimate of detailed performance behavior is required *(simulation preferred)*

Slide 23

Slide 23: When NOT to Select Markov Modeling for Dependability Analysis

It is important to know when to select Markov modeling as the modeling method of preference. It is equally important to know when ***not*** to select Markov modeling and to select a different modeling method instead. In general, Markov modeling is not the preferred modeling method whenever one of the following conditions arise:

- If the system can be satisfactorily modeled with a simpler combinatorial method, then use of Markov modeling may be overkill and the simpler method should be used instead. There are several motivations for this: a combinatorial model may be smaller and may be more easily constructed than a Markov model of the system. It may be more computationally efficient to solve a combinatorial model than a Markov model. Also, the combinatorial model may be easier for the analyst to understand, especially if the analyst is not a specialist in modeling.
- If the system requires a very large Markov model, then the effort required to generate or solve the Markov model may be excessive and an alternate modeling method should be considered. This is especially the case if the model is one of the more sophisticated types of Markov models which require comparatively large amounts of execution time to solve. Use of hierarchical or hybrid modeling techniques may help subdivide the model and alleviate problems caused by too many states in the model.
- If the system behavior to be modeled is too complex or detailed to be expressed in a Markov type model, then an alternate method capable of representing the behavior of interest should be used instead of Markov modeling. Here, "too complex" includes system behavior that can not be modeled because of limitations due to the Markov property or assumptions about transition rates. Although sometimes hierarchical/hybrid methods are sufficient when Markov modeling cannot be used, often simulation is needed to capture behavior that is too complex for Markov models. This also holds true when detailed performance behavior must be modeled instead of or in addition to dependability. Markov models can capture performance behavior through what are called *Markov reward models*[15], but these are more limited in flexibility and range of performance behavior that can be represented than simulation. With simulation, the level of detail in the performance or dependability behavior that can be expressed is limited only by the level of detail of the model, which itself is limited only by the abilities and patience of the analyst who builds, validates, and evaluates the model.

How Selected System Behaviors Can Be Modeled

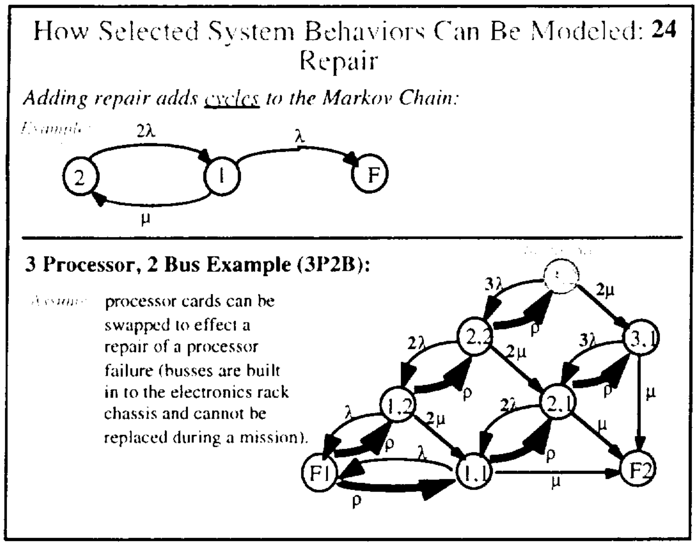
We next present several examples that demonstrate how some selected system behaviors can be modeled with Markov models.

Slide 24: Repair

Markov modeling is very well suited to modeling repair situations. In general, the occurrence of failures causes loss of function and/or redundancy. Repair involves the restoration of functionality and/or redundancy that has been lost. Restoration of full functionality is usually assumed[3], taking the system back to a state it had occupied earlier. For this reason, *modeling repair usually adds cycles to a Markov model.* The example 3-state model in the diagram at the top of the slide illustrates this concept. The Markov chain in the slide represents a system with two active redundant components. In the state labeled *{2}*, both components are functioning properly. A failure occurs in one or the other of the two components at a rate of 2λ, taking the system to state *{1}* where only one component remains functional. The occurrence of a second failure (at failure rate λ) takes the system to a failure state. If a repair person is available, then upon the first failure he/she can begin repairing the component that failed. This is represented in the diagram by the transition from state *{1}* to state *{2}* labeled with a rate of repair μ. Assuming the repair restores full functionality to the component that failed, upon completion of the repair the system will again have two fully functionality components, indicating that it will have returned to state *{2}*.

[3] Occasionally only partial restoration of functionality is achieved by the repair activity. This can be modeled in a Markov model by having the repair transition take the system to another state which represents a degraded functionality, rather than back to the state representing full functionality.

Note that the result of adding this repair activity to the model has resulted in the addition of a cycle between states *{2}* and *{1}*.



How Selected System Behaviors Can Be Modeled: **24**
Repair

Adding repair adds cycles to the Markov Chain:

Example:

3 Processor, 2 Bus Example (3P2B):

Assume: processor cards can be swapped to effect a repair of a processor failure (busses are built in to the electronics rack chassis and cannot be replaced during a mission).

Slide 24

This basic procedure for modeling repair in Markov models can be generalized to handle a wide range of variations in repair resources and procedures. This includes such variations as:

- several available repair persons instead of just one
- the requirement that some components must be repaired beforeothers
- the policy that a certain number of failures must occur before repair activities are initiated
- the policy that, once a system has reached a failure state, a specific number(s) of components of specific types must successfully be repaired before the system may return to being operational

As an example of the last bulleted case, suppose that, in the 3-state example discussed above, it is the policy that no repair will be performed until the system has reached the failure state, and that both components are typically repaired before the system will be declared operational. This criteria might apply to a low-safety-critical or non-time-critical application for which there is a significant cost involved in initiating the repair activity. An example from a household-oriented domain might be a house with two bathtubs/showers in which a "failure" would be a bathtub drain getting seriously clogged. Considering the time and expense of calling in a professional plumber to clear out the entire drain piping system, the household members might well opt to wait until both bathtubs become unusably clogged before calling a plumber to fix both drains in one service trip. In this case, the repair transition would go from the system failure state labeled *{F}* directly to the state labeled *{2}*, and the repair rate μ would represent the time required to clean out the entire drain piping system for the house.

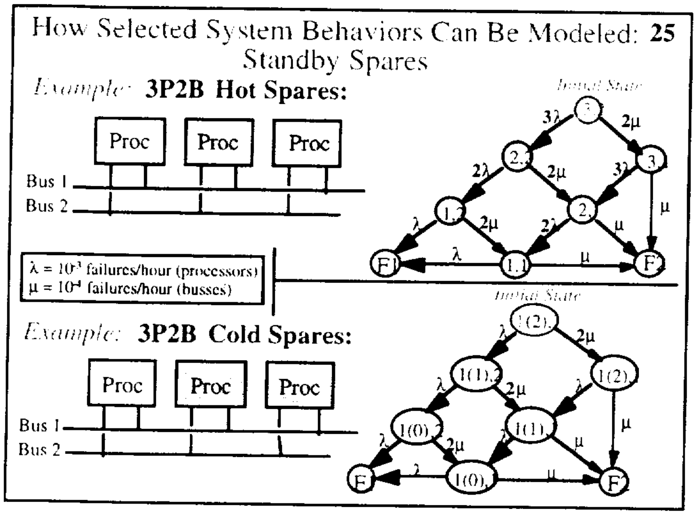
All of the above generalizations of the repair modeling process paragraph are possible in Markov models because of the great flexibility that the Markov modeling technique offers the analyst in: 1) specifying transitions between any arbitrary pairs of states, 2) labeling the transitions with any arbitrary combinations of repair (transition) rates, and 3) the interpretation of what each Markov model state represents.

All of these concepts for modeling repair situations can be generalized from the simple 3-state Markov model to larger and more complex Markov models. An example is shown in the Markov model shown at the bottom of the slide. Here repair has been added to the basic model of failure events for the 3P2B system that was first introduced in *Slide 14*. Assume that the processors reside on individual electronics cards (with one processor to a card) that are mounted in a chassis in an electronics rack, and that the chassis has the two redundant busses built in to it. The overall 3P2B system would then consist of three processor cards mounted in the chassis. With this physical configuration, repairing a failed processor is relatively easy: it is as simple as swapping out a bad card for a new spare card and can be performed at a rate ρ. Repairing a failed bus, however, is a much more complex and difficult procedure (the entire chassis must be replaced and reconfigured), and is not considered feasible during a mission (this would be the case if the 3P2B system is, for example, part of a control system for an aircraft). The model shown in the diagram at the bottom of the slide shows the repair arcs , labeled with the repair rate ρ, that model the repair activity (i.e. the swapping of a spare processor board for a failed one) that brings the system back from each degraded state to the state with one more functional processor. This introduces cycles into the Markov model between each pair of states connected by a transition representing a processor failure. Because repair of the busses is not considered feasible, there are no corresponding repair transitions (and no resulting cycles) between pairs of states connected by a transition representing a bus failure. Any generalizations of the repair policy for processors (such as those that were listed earlier) would be included in the 3P2B Markov model in a same manner, and may result in different repair transition rates and/or repair transitions going to different states in the Markov model than has been produced by the basic repair policy illustrated in the slide.

Slides 25 and 26: Standby Spares

Markov models also are well suited to modeling standby spares. As in the case of modeling repair, this capability also may be traced to the flexibility that Markov modeling affords the analyst both for specifying the transition rates on individual transitions and for interpreting what the states represent in the model. The diagrams in these two slides illustrate how Markov models can be used to model all three types of standby spares: hot, warm, and cold.

In general, a ***standby spare*** is a component (similar or identical in construction and/or functionality and performance to a ***primary active component***) which is held in reserve to take over for a primary active component should the primary component experience a failure. As soon as a failure occurs in a primary active component for which spares remain available, a spare unit is switched in to take over for the failed primary unit, and the system moves to a state that represents the presence of one less redundant component of that type (or a failure state, if all redundancy for that component type has been exhausted). The failure rate at which such a transition occurs is the sum of the failure rates for all active components of that type.



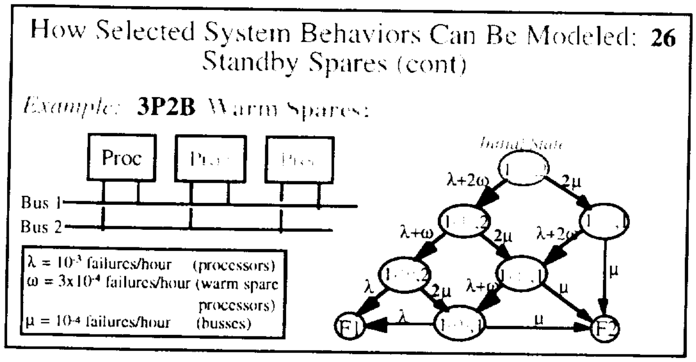
How Selected System Behaviors Can Be Modeled: **25**
Standby Spares

Example: **3P2B Hot Spares:**

λ = 10^{-3} failures/hour (processors)
μ = 10^{-4} failures/hour (busses)

Example: **3P2B Cold Spares:**

Slide 25



How Selected System Behaviors Can Be Modeled: **26**
Standby Spares (cont)

Example: **3P2B** Warm Spares:

λ = 10^{-3} failures/hour (processors)
ω = $3x10^{-4}$ failures/hour (warm spare processors)
μ = 10^{-4} failures/hour (busses)

Slide 26

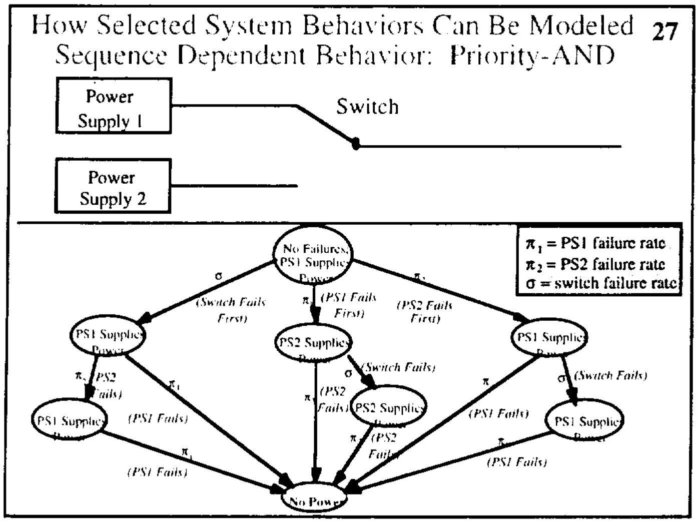
The modeling of hot standby spares is essentially what has been used in both major examples discussed in this tutorial so far (i.e. the 3P2B example and the 3-state model(s)). A ***hot standby spare*** is a component that remains in a fully powered, operations-ready state (possibly even shadowing the operations of the primary active unit) during normal system operation so that it can take over as quickly as possible should the primary active unit experience a failure. As a consequence, it is assumed to be vulnerable to failures at the same failure rate as if it were a primary active unit. The Markov model in the diagram at the top of *Slide 25* shows how hot standby spares are modeled. Since any of the hot spares or the primary active unit could fail at any time, the failure rate at which a failure occurs is the sum of the failure rates for all active components of that type (i.e. all the hot spares and the primary active unit). For example, in the Markov model in the diagram at the top of *Slide 25*, the transition from the initial state that represents a processor failure has a transition (failure) rate of 3λ, since there is one primary active unit and 2 hot spare units, making a total of three components vulnerable to failure, each of which have a failure rate of λ. Note that, since the use of standby spares involves a detection-and-reconfiguration process which may not be perfectly reliable, a more detailed modeling of the use of standby spares than shown in these slides would include *coverage probabilities*. An example of this for the 3P2B system for hot spares is shown in *Slide 29* and is discussed in more detail below in the commentary for that slide.

A ***cold standby spare*** is a component that is powered down until it is needed to take over for a failed primary unit. At that time, it is powered up, initialized (if necessary), and takes over the operations for which the failed primary unit formerly was responsible. The usual assumption is that the cold spare is not vulnerable to failure at all while it is powered down, but that after it has been activated it can fail at any time at the failure rate that characterizes its fully active state.

The Markov model in the diagram at the bottom of *Slide 25* shows how the model would change if the spare processors are cold spares rather than hot spares. This Markov model uses a slightly different state labeling method in order to be able to track the status of the standby spare processors. Specifically, the part of the state label representing the processors is expanded to include both the number of active primary units (which in this case will always be one for all operational states) and the number of available spares remaining (shown in the parentheses). For example, the label of the initial state, *{1(2),2}*, indicates that there is one active primary processor functioning, two cold standby spares available, and two functioning busses. A transition resulting from a processor failure necessarily implies that it is the active primary unit that has failed (since none of the unpowered spare units are allowed to fail), and that, when the state at the destination of the transition is reached, one of the unpowered spare units will have been activated and will have taken over for the failed former primary unit. This will necessarily cause the count of available standby spares to decrease by one. For example, the label of the state at the end of the transition from the initial state,*{1(2),2}*, to the state representing recovery from a processor failure, *{1(1),2}*, indicates one fewer spare processor is available than before the failure. The transition rate for this transition is λ (rather than 3λ as in the case for hot spares) because only the active primary unit can fail (the two unpowered cold spare processors cannot fail as long as they remain unpowered). For similar reasons, all transitions representing processor failures in the model have a transition rate of λ, regardless of how many spare processors remain available.

Finally, a ***warm standby spare*** is a component that remains powered during normal operation of the primary active unit, however it is assumed to operate in a manner that subjects it to less environmental and/or operational stress than a fully active primary unit until it is needed to take over for a failed primary unit. As a consequence, the usual assumption is that the warm spare is vulnerable to failure at a lesser failure rate than when it is in its fully active state, but that after it has been activated it can fail at any time at the failure rate that characterizes its fully active state. Since the warm spares are active and therefore vulnerable to failure, the failure rate(s) of the warm spares contribute to the transition rate of a transition representing the failure of a particular component type. More specifically, the transition (failure) rate of a transition representing the failure of a particular component type is the sum of the failure rates of all active components, which includes the warm spares as well as the fully active units.

The Markov model in the diagram of *Slide 26* shows how the model would change if the spare processors are warm spares rather than hot spares. The same state labeling method that was used for the cold spare Markov model is also used here in the warm spare Markov model. The initial state represents one primary processor that is fully active and two warm spare processors that are "partially active" from a failure perspective. The failure rate for the transition representing a processor failure is therefore the sum of the failure rate of the fully active processor, λ, and the failure rates of all the warm spare processors, 2ω, giving a total failure rate of λ + 2ω as shown in the diagram in the slide. The resulting state after the transition occurs represents a situation where the system is operating with one fully active processor and one remaining warm spare. Note that this is the case regardless of whether it was the primary unit processor that failed or one of the warm spares.



How Selected System Behaviors Can Be Modeled 27
Sequence Dependent Behavior: Priority-AND

π_1 = PS1 failure rate
π_2 = PS2 failure rate
σ = switch failure rate

Slide 27

Slide 27: Sequence Dependent Behavior: Priority-AND

Because Markov models fundamentally are comprised of sequences of states connected by transitions, they are a natural vehicle for modeling sequence dependent behavior. ***Sequence dependent behavior*** is behavior that depends in some way on the sequence in which events occur. Examples include:

- Situations where certain events cannot take place until other events have occurred. A special case of this is the cold spare, where (because of the assumption that the cold spare cannot fail while it is powered down) the failure of an initially cold spare component cannot occur until the component has been activated to take over for a failed primary unit.
- Situations where certain events cause certain other events to occur, or preclude certain other events from occurring. This has been called ***functional dependency***[3, 4]. It is easily modeled with Markov models because of the flexibility in specifying which pairs of states are connected by transitions, and what the transition rates are for individual transitions.
- Situations where future behavior differs depending on the order in which two or more certain events occur.

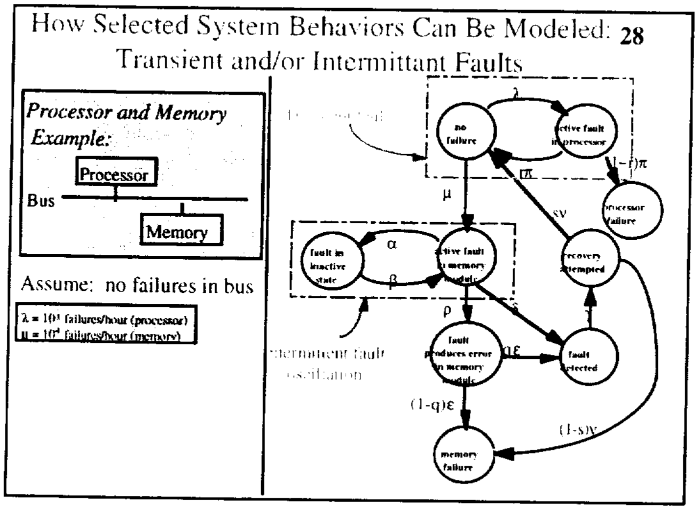
The situations described in the last bullet have long been modeled in fault trees using what are called Priority-AND gates[16]. In a fault tree, a Priority-AND gate is an AND gate in which the output event of the gate occurs only if all input events occur *and* they occur in a specific order. If all input events occur but in an incorrect order, the output event of the gate does not occur. In terms of a Markov model, a Priority-AND type of situation is one which requires the specific order of event occurrences to be included in what (at least some of) the states represent. This is easily done because of the flexibility that Markov modeling offers in assigning interpretations (meanings) to individual states in the model.

The slide shows an example of how a Priority-AND type of sequence dependency (i.e. one that requires that sequences of events be "remembered" in the interpretations of the state meanings) can be modeled with a Markov model. Suppose a power subsystem consists of two redundant power supplies connected to the rest of the system by a simple toggle switch. Initially, Power Supply 1 is supplying power to the system, and Power Supply 2 is a hot spare backup. If Power Supply 1 fails, the system is supposed to automatically reconfigure itself to switch to Power Supply 2 so that no loss of power is experienced as a result of the failure of Power Supply 1. Hence, the system would experience a loss of power only after Power Supply 2 failed. However, different outcomes may occur depending on the sequence in which the three components (the two power supplies and the switch) fail.

The Markov model in the diagram at the bottom of the slide depicts the sequence dependent alternatives and shows how they can be modeled. The initial state represents the situation where all components are working properly and Power Supply 1 is supplying power to the rest of the system. The following situations may arise, depending on the order in which the components fail:

- If the switch fails first (the leftmost transition out of the initial state), there is no immediate effect on the operation of the system - Power Supply 1 continues to supply power to the rest of the system. However, the redundancy protection offered by Power Supply 2 is lost, because the system can no longer switch over to Power Supply 2 if a failure occurs in Power Supply 1. The system will now lose power as soon as Power Supply 1 fails (the failure of Power Supply 2 will have no effect on the operation of the system).
- If Power Supply 1 fails first (the center transition out of the initial state), the switch would reconfigure the power subsystem so that Power Supply 2 would supply power to the system instead (this reconfiguration process could be modeled in more detail by considering the probability of success or failure of the reconfiguration using a coverage probability (see *Slide 29*)). Since this is a non-repairable system in this example, after the reconfiguration to use Power Supply 2 occurs, the failure of the switch no longer has an effect on the operation of the system. The system would lose power as soon as Power Supply 2 fails, whether the switch fails or not.
- If Power Supply 2 fails first (the rightmost transition out of the initial state), there is no immediate effect on the operation of the system - Power Supply 1 continues to supply power to the rest of the system. However, the redundancy protection offered by Power Supply 2 is lost, because the system can no longer switch over to Power Supply 2 if a failure occurs in Power Supply 1. The system will now lose power as soon as Power Supply 1 fails (the failure of the switch will have no effect on the operation of the system).

Note that, even though there are four states labeled "PS1 Supplies Power" in the Markov model, these four states are distinct states and are <u>**NOT**</u> the same state. This is because each of these states represents not only the situation that Power Supply 1 is supplying power, but they also implicitly represent the sequence of component failures that occurred in order to reach the state (this could be more explicitly indicated in the state label than it has been in the state labeling policy used for this example). This example is simple enough that the exact sequence of component failures does not have great importance; however, the reader should be able to recognize that other examples may be constructed in which the order in which the components have failed could be of critical importance. The bottom line is: Markov models can model such situations by allowing such "memory" of event sequences be part of the interpretation of the meaning of each state.



How Selected System Behaviors Can Be Modeled: 28
Transient and/or Intermittant Faults

Processor and Memory Example:

Assume: no failures in bus

$\lambda = 10^{-4}$ failures/hour (processor)
$\mu = 10^{-4}$ failures/hour (memory)

Slide 28

Slide 28: Transient and/or Intermittent Faults

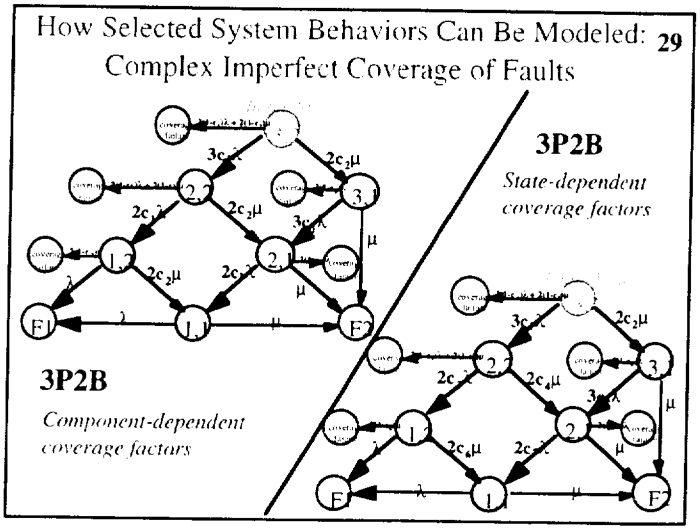
Markov models are also adept at modeling situations involving transient and/or intermittent faults. A ***transient fault*** is a fault that enters and remains in an active state (and is thus capable of causing a malfunction) for a finite time *t*, after which it spontaneously and permanently disappears or enters a benign state (in which it can no longer cause a malfunction)[17]. An example might be a transient in a power line, or a stray gamma ray that causes a bit to flip in a memory storage location. An ***intermittent fault*** is a fault that randomly oscillates between active and benign states[17]. An example might be a loose wire connection.

The Markov model in the diagram in the slide shows an example that illustrates how both transient and intermittent faults can be modeled (it is based on a modified version of the coverage model used in the CARE III reliability prediction program[6, 17]). Suppose a subsystem consists of a processor and a memory communicating with each other and other components in the system over a bus. For the sake of simplicity, we will model failures in the processor and memory only (this is equivalent to assuming that the bus does not fail). The processor fails at rate λ. There is simple error recovery implemented for it sufficient to recover from transient failures (for example, retry of instruction executions and/or I/O requests that fail a parity check). If the recovery procedure (i.e. retry) succeeds (with probability r), then the system moves back to the initial ***{no failure}*** state at rate rπ. If the recovery procedure was unsuccessful (with probability 1-r, indicating a permanent fault), then the system goes to the state labeled ***{processor failure}*** at rate (1-r)π. This illustrates an example of modeling of a transient fault (shown in the dotted box labeled "Transient fault").

A more complex error recovery procedure is implemented for memory failures. Suppose that the memory module experiences faults at a rate μ. This will cause the system to move from the ***{no failures}*** state to the ***{active fault in memory module}*** state. If the fault is an intermittent fault, the system may oscillate between the active and inactive states as shown in the slide in the box labeled "Intermittent fault oscillation", moving from the active state to the inactive state at rate α, and back again from the inactive state to the active state at rate β. The remainder of the fault handling procedure for the memory unit is implemented as shown by the remaining states in the slide.

Slide 29: Complex Imperfect Coverage of Faults

If a reconfiguration process (invoked in response to a failure of an active component) can itself fail to be completed successfully, then the fault that caused the reconfiguration to be initiated is called an ***imperfectly covered fault***, and the probabilities that reconfiguration is or is not successful are called ***coverageprobabilities***. Imperfect fault coverage is expressed in Markov models through the use of two outgoing transitions for each imperfectly covered fault that can occur while the system is in a particular operational state. One of these transitions represents the successful completion of reconfiguration. This transition leads to a state in which the system is operating after reconfiguration has been achieved, and its transition rate is the product of the probability of successful reconfiguration (say, *c*) and the rate of occurrence of the imperfectly covered fault. The second transition represents an unsuccessful reconfiguration attempt. This second transition leads to a state in which the system has failed due to an uncovered fault, and its transition rate is the product of the probability the reconfiguration does not succeed (*1-c*) and the rate of occurrence of the imperfectly covered fault.



How Selected System Behaviors Can Be Modeled: 29
Complex Imperfect Coverage of Faults

Slide 29

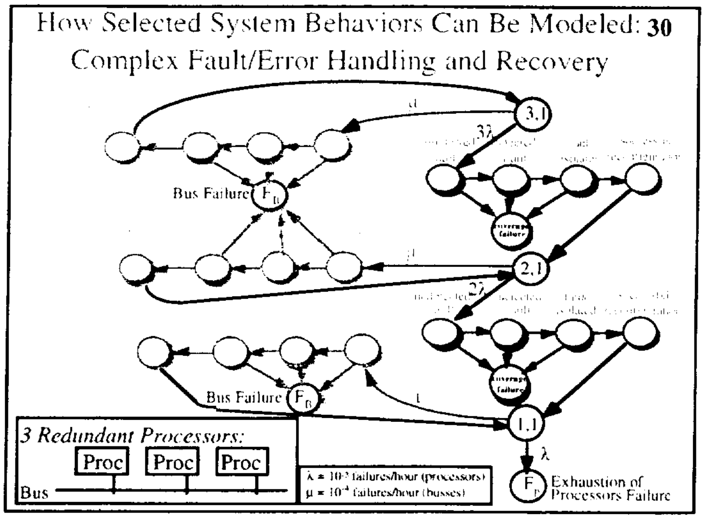
The Markov models in the slide illustrate how imperfect coverage of faults can be added to the Markov model for the 3P2B example system introduced in *Slide 14*. For example, the transition from the initial state for an imperfectly covered processor fault is separated into two transitions: one representing the successful reconfiguration which goes to state *{2,2}* at rate $3c_1\lambda$ (where c_1 is the probability of a successful processorreconfiguration*)*, and one representing a failed reconfiguration attempt that goes to a coverage failure state at rate $3(1-c_1)\lambda$. Likewise, the transition from the initial state for an imperfectly covered bus fault is separated into two transitions: one representing the successful reconfiguration which goes to state *{3,1}* at rate $2c_2\mu$ (where c_2 is the probability of a successful bus reconfiguration*)*, and one representing a failed reconfiguration attempt that goes to a coverage failure state at rate $2(1-c_2)\mu$. (Note that the two coverage failure states for the individual processor and bus components can be merged together into one failure state, with a transition rate that is the sum of the transition rates to the two formerly independent coverage failure states (i.e. with a new transition rate of $3(1-c_1)\lambda + 2(1-c_2)\mu$).

There is more than one way to add coverage to a Markov model, depending on what assumptions are made about the reconfiguration process. For example, the two Markov models in the slide show two different ways to assign values to coverage probabilities. The simpler of the two methods is to assume that each component type has a specific probability that a reconfiguration will succeed. This results in the Markov model on the left hand side of the slide. If the reconfiguration process is modeled in more detail, however, one will find that coverage probabilities (even for the same component type) actually tend to vary from state to state[4] (see *Slide 35*). The Markov model on the right hand side of the slide shows how this more general situation can be modeled. The bottom line is that, because of the flexibility in specifying transition rates for transitions, Markov models are capable of modeling imperfect fault coverage to any level of complexity achievable with coverage probabilities.

Slide 30: Complex Fault/Error Handling and Recovery

The flexibility in specifying the interpretations for individual states makes Markov models well suited for modeling complex fault/error handling behavior. The Markov model in the slide provides an example. Consider a subsystem consisting of three redundant processors which communicate with each other and the rest of the system over a common bus. Assuming that the system implements error recovery procedures for both the processors and the bus, the Markov model shows how such error recovery procedures can be modeled.

If a fault occurs in a processor, it is initially undetected and could potentially cause a system crash before it is detected. The first step in recovery is the detection of the fault (which, if not successful, could cause a system crash), followed by the isolation of the fault to determine which processor has experienced the fault and to switch it off-line (an unsuccessful outcome of this step could also cause a system crash). Once the failing processor has been identified and switched off-line, the system will have achieved a successful reconfiguration and can continue operating in a state of degraded operation. Each of these steps are denoted by shadowed states on the right hand side of the slide, and transitions between these states will take place at specific rates (which are not shown in the slide for the sake of simplifying the diagram).


How Selected System Behaviors Can Be Modeled: **30**

Complex Fault/Error Handling and Recovery

3 Redundant Processors:

$\lambda = 10^{-3}$ failures/hour (processors)

$\mu = 10^{-4}$ failures/hour (busses)

Slide 30

If a fault occurs in the bus during an I/O operation, it is initially undetected and could potentially cause a system crash before it is detected. The first step in recovery is the detection of the fault (which, if not successful, could cause a system crash). Since there is no redundancy for busses in this example, recovery from a bus fault is limited to attempting retries of the I/O operation in order to recover from transient and intermittent faults. Hence, the next step is to retry the I/O operation (which can cause a system crash if not successful). If the retry is successful, the system may continue operation in the state it was in before the bus fault occurred. Each of these steps are denoted by shadowed states on the left hand side of the slide, and transitions between these states will take place at specific rates (which are not shown in the slide for the sake of simplifying the diagram).

Typically the transition rates between states for fault/error handling are much faster (often by orders of magnitude) than the transition rates for transitions that represent the occurrence of failures (e.g. λ and μ in the case of this example). Consequently, adding fault/error handling to a Markov model in the fashion has the potential of adding *stiffness* to the model (see *Slide 32* and *Slide 35*, so the analyst should take care in using this modeling technique. *Slide 35* discusses one method for mitigating this problem.

Additional Issues

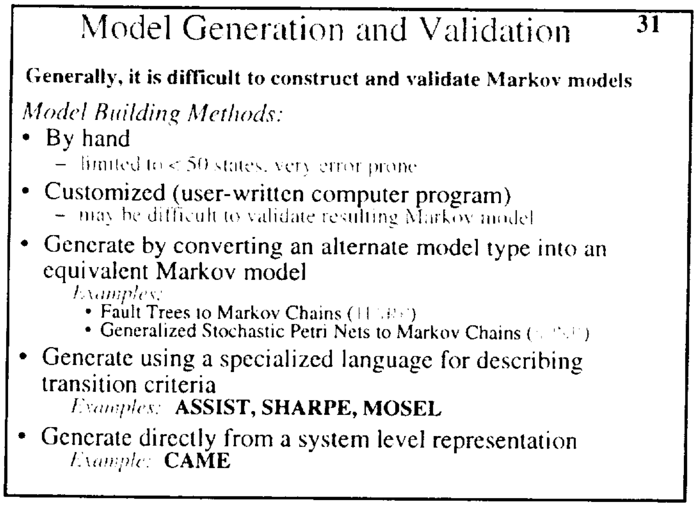
There are some additional issues that must be considered by a dependability analyst who is intending to use Markov modeling to predict the dependability of a system. They are important because they can impose limitations on the applicability of Markov modeling as a modeling technique. These issues are discussed in the next several slides.

Slide 31: Model Generation and Validation

One of the most troublesome aspects of using Markov modeling for dependability analysis is the fact that it is generally

[4] The details of the reason behind this fact are beyond the scope of this tutorial, but the essence of the reason for this is that the coverage probabilities depend partially on the number of components vulnerable to failure, which of course varies from state to state.

difficult to construct and validate Markov models. This is especially true for large models. An analyst has several options for building a Markov model of a system. The most basic is to draw the model by hand. This is a very error prone method and usually is practical only for very small models - those with 50 states or less. The next best option is to write a customized computer program to generate the Markov model from information about the system that can be coded in a standard programming language like FORTRAN or C. This may also be a troublesome method to use because of the difficulty in debugging the program and ensuring that a correct Markov model is generated. Again, this is particularly true if the model is large (has many states). However, before the advent of generalized analysis programs designed for Markov modeling and analysis, this method was often the only one available to a dependability analyst. The recent past has seen the development of several generalized dependability analysis programs designed to implement sophisticated Markov modeling techniques. The generation of the Markov model has been a common obstacle for all of the developers of such programs. Consequently several of these programs have included features for automatically generating the Markov models from alternate specifications as an integral part of the program. Three different approaches taken by several important modeling programs will be described here.



Model Generation and Validation 31

Generally, it is difficult to construct and validate Markov models

Model Building Methods:

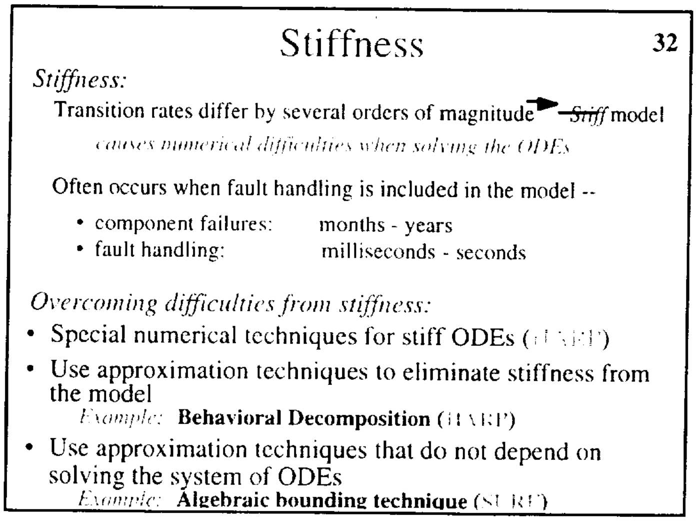
- By hand
  - limited to < 50 states, very error prone
- Customized (user-written computer program)
  - may be difficult to validate resulting Markov model
- Generate by converting an alternate model type into an equivalent Markov model
  - *Examples:*
    - Fault Trees to Markov Chains (HARP)
    - Generalized Stochastic Petri Nets to Markov Chains (SPNP)
- Generate using a specialized language for describing transition criteria
  - *Examples:* **ASSIST, SHARPE, MOSEL**
- Generate directly from a system level representation
  - *Example:* **CAME**

Slide 31

One method for automatically generating a Markov model is to automatically convert a model of a different type into an equivalent Markov model. This approach has been taken by the Hybrid Automated Reliability Predictor (HARP) program[18] (now part of the HiRel package of reliability analysis programs[19]) and the Stochastic Petri Net Package (SPNP)[20], both of which were developed at Duke University. HARP converts a dynamic fault tree model into an equivalent Markov model[21]; SPNP converts a generalized stochastic Petri net (GSPN) into an equivalent Markov model. This approach provides an advantage if the alternate model type offers a more concise representation of the system or one which is more familiar to the analyst than Markov models (as is often the case for fault trees), or if the alternate model type is able to more easily represent certain system behavior of interest than Markov models (as is the case with Petri Nets with respect to, for example, representing concurrent events). A second method for automatically generating a Markov model is to use a specialized computer programming language for describing transition criteria. This approach is used by the Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST) program developed at NASA Langley Research Center[22]. This approach offers an advantage to those analysts who are more comfortable with specifying system behavior in a computer programming language format rather than formats offered by other modeling methods. A third method that has been developed generates a Markov model directly from a system level description. This technique is used by the Computer Aided Markov Evaluator program (CAME)[23] developed at C. S. Draper Laboratories.

Slide 32: Stiffness

Another difficulty that arises when Markov models are used to analyze fault tolerant systems is a characteristic called ***stiffness***. Stiffness appears in a Markov model which has transition rates that differ by several orders of magnitude. Stiffness is a problem because it causes numerical difficulties during the solution of the ordinary differential equations (ODEs) that arise from the Markov model. Stiffness often appears when fault/error handling behavior is included in the Markov model. Fault/error handling causes large differences in transition rates within the model by virtue of the difference in the time scales associated with failure processes and fault handling processes. Failures of components typically occur in a time frame ranging from months to years between failures. Conversely, once a fault occurs it needs to be handled rapidly to avoid a system failure, so fault handling typically occurs in a time frame ranging from milliseconds to seconds. Hence the transition rates that represent component failures are orders of magnitude slower than transition rates that represent the response to a fault, and it is this which is the source of the stiffness.



Stiffness 32

Stiffness:

Transition rates differ by several orders of magnitude → *Stiff* model

causes numerical difficulties when solving the ODEs

Often occurs when fault handling is included in the model --

- component failures: months - years
- fault handling: milliseconds - seconds

Overcoming difficulties from stiffness:

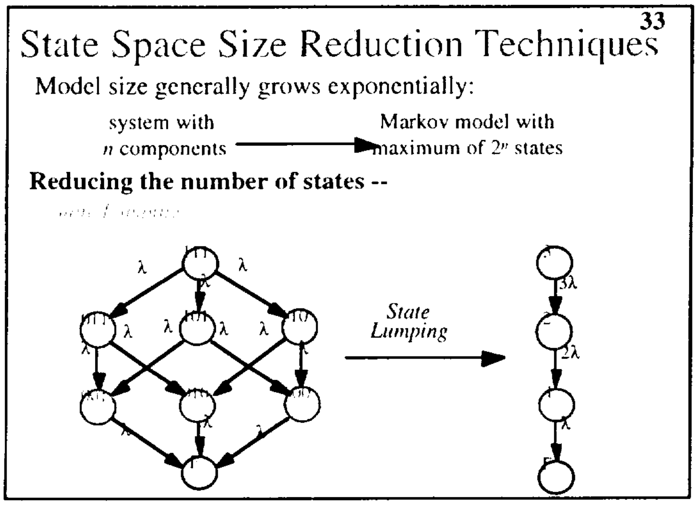
- Special numerical techniques for stiff ODEs (HARP)
- Use approximation techniques to eliminate stiffness from the model
  - *Example:* **Behavioral Decomposition** (HARP)
- Use approximation techniques that do not depend on solving the system of ODEs
  - *Example:* **Algebraic bounding technique** (SURE)

Slide 32

There are a number of ways to attempt to overcome the difficulties presented by stiffness in the Markov model. Special numerical techniques do exist for solving stiff ODEs[24]. As an alternative, it is possible to use certain approximation techniques which can eliminate stiffness from the model before the ODEs are solved. An example of such an approximation method is ***behavioral decomposition***, which will be described shortly. This method is used by the HARP reliability analysis program[18, 19]. Yet another alternative is to use a different type of approximation technique that does not rely on solving the ODEs from the Markov model. An example of this approach is the algebraic bounding technique used by the Semi-Markov Unreliability Range Evaluator (SURE) program developed at NASA Langley Research Center[25, 26].

Slide 33: State Space Size Reduction Techniques: State Lumping

Next to the difficulty in generating and validating Markov models of realistic systems, the problem posed by excessive numbers of states is the second most serious obstacle to effective use of Markov models for dependability analysis. A system composed of *n* components in theory can require a Markov model with a maximum of 2^n states. Usually the actual number of states in a model is much less than 2^n because typically once a critical combination of events or component failures causes system failure, further subsequent component failures will not cause the system to become operational again (that is, failure states in the Markov model are absorbing - they do not have outgoing transitions to operational states). Even so, a system with many components may still require a very large number of states to enumerate all the operational states of the system. The next several slides will describe some techniques that can be used to reduce the number of states in a model in order to address this problem.



33

State Space Size Reduction Techniques

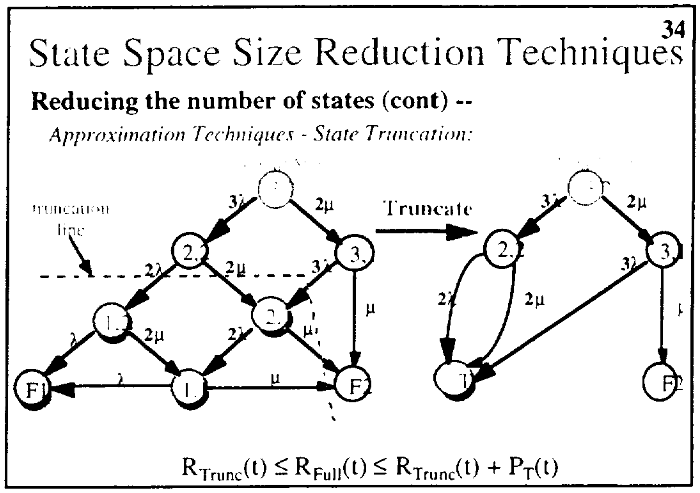
Model size generally grows exponentially:

system with *n* components → Markov model with maximum of 2^n states

Reducing the number of states --

Slide 33

Under some circumstances, it may be possible to combine groups of states in the model together into composite states. This process is called <u>*state lumping*</u> [27] and has the potential to reduce the numbers of states required depending on the form of the model, the meanings of the states, and system behavior of interest that must be represented in the model. An example of the lumping together of states is shown in the slide for a system consisting of three redundant components. If what is important to the analyst is only the number of operating components rather than a detailed accounting of each operational configuration, then the Markov model on the left containing 8 states may be transformed into the Markov model on the right containing 4 states by grouping together the three states for which two components are operating (one component failed) and the three states for which only one component is operating (two components failed). States to be lumped together in this way must meet certain requirements (see [27]).



34

State Space Size Reduction Techniques

Reducing the number of states (cont) --

Approximation Techniques - State Truncation:

$R_{Trunc}(t) \leq R_{Full}(t) \leq R_{Trunc}(t) + P_T(t)$

Slide 34

Slide 34: State Space Size Reduction Techniques: State Truncation

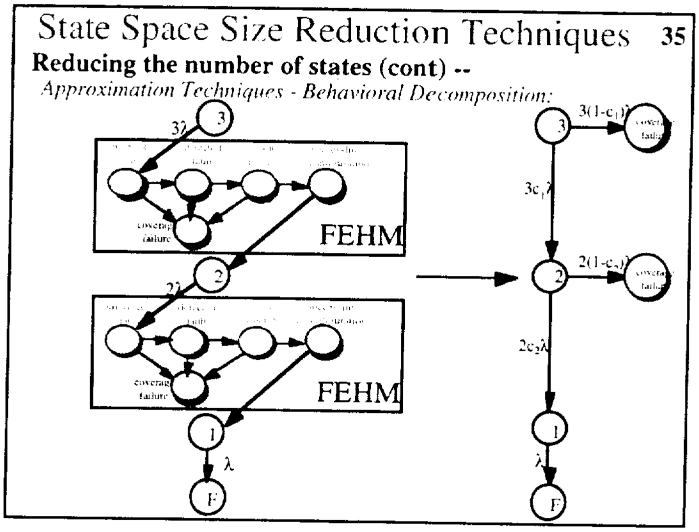
Another approximation technique capable of reducing the number of states in the Markov model is called <u>*state truncation*</u>. It involves constructing the model to include only those states representing a limited number of component (covered) failures. States which represent configurations with a larger number of component failures than the truncation limit are combined together into an aggregate state. In general, the aggregate state will contain both failure states and operational states. This fact allows a bounded interval for the system reliability to be obtained by assuming that the aggregate state represents in turn: 1) only failure states, and then 2) only operational states. Assuming that the aggregate state represents only failure states underestimates that actual system reliability (because some of the states within the aggregate state were actually operational states that were assumed to be failure states), whereas assuming the aggregate state represents only operational states overestimates the actual system reliability (because some of the states within the aggregate state were actually failure states that were assumed to be operational states).

An illustration of this technique is shown in the slide. On the left is a Markov model of the 3-processor, 2-bus example system that was introduced in Slide 14. Suppose that the computer to be used to solve this model has a ridiculously small memory and the entire Markov model cannot be generated (this example may stretch the imagination a bit, but the situation would become more realistic if the system for which the Markov model is to be generated were to contain 100 components or more). Suppose that, as a consequence of the memory limitations of the computer, it is decided to include in the generated Markov model only those states that represent one or fewer covered component failures. This means that after states with one component failure are generated, all further states in the model (representing two or more component failures) are aggregated together into one state. The effect of this process is that states *{1,2}*, *{2,1}*, *{1,1}*, and *{F1}* (the shadowed states below the truncation line in the Markov chain diagram on the left side of the slide) are all lumped together into a single aggregate state labeled *{T}* as shown in the truncated model on the right side of the slide. Note that the aggregate state *{T}* contains both operational states (i.e., states *{1,2}*, *{2,1}*, and *{1,1}*) and failure states (in this case, state *{F1}* is the only failure state). The reliability of the full model on the left side of the slide is simply the sum of the probabilities for the three operational states above the truncation line (states *{3,2}*, *{2,2}*, and *{3,1}*) and the three operational states below the truncation line (states *{1,2}*, *{2,1}*, and *{1,1}*). If the aggregate state *{T}* in the truncated model on the right side of the slide is first considered to be a failure state, then the reliability of the truncated model is the sum of probabilities of only the three operational states above the truncation line. This is less than the actual reliability of the full model and so serves as a lower bound on the actual system reliability. If the aggregate state *{T}* is next assumed to be an operational state, then the reliability of the truncated model is the sum of the probabilities of the six operational states *and also* the probability of state *{F1}*. This is greater than the actual reliability of the full model (because the failure state *{F1}* is counted as an operational state, when in reality it is not) and so serves as an upper bound for the actual system reliability. Hence a bounded interval for the actual system reliability may be obtained by solving a Markov model of only five states instead of a Markov model of eight states.

The savings obtained for this small example may not seem significant, but the savings may be considerably more impressive if the truncated Markov model contains several thousand states and the states below the truncation line number in the hundreds of thousands. The reader may note that the width of the bounded interval in which the actual system reliability lies is equal to the probability of the aggregate state *{T}*. This indicates that the interval will be small (and the approximation most effective) when the probabilities of the states below the truncation line are very small compared to the probabilities of the states above the truncation line. Since the probability migration among the states moves from states above the line to states below the line as a mission progresses, this implies that state truncation is most effective for models of systems for which the mission time is relatively short and failure rates of components are very small (i.e. inter-state transition rates are very slow). Under these conditions, most of the probability density will likely remain above the truncation line for the time period of interest, and so state truncation will be an effective approximation technique.

Slide 35: State Space Size Reduction Techniques: Behavioral Decomposition

Another state reduction technique may be applicable when some states of the model are used to model fault/error handling behavior and the fault/error handling transitions are several orders of magnitude faster than fault occurrence transitions. If the fault/error handling states can be arranged in groups (sometimes called Fault/Error Handling Models, or FEHMs) such that the fast transitions occur only between states within a group, then an approximation technique called ***behavioral decomposition*** can be employed to produce a simplified model, resulting in a reduction in the number of states in the simplified model.



State Space Size Reduction Techniques 35

Reducing the number of states (cont) --

Approximation Techniques - Behavioral Decomposition:

Slide 35

The mathematical details of the behavioral decomposition approximation technique are beyond the scope of this tutorial, however intuitively the process involves reducing the states within the FEHM to a probabilistic branch point, which then replaces them in the original model to produce the simplified model. This may be done because the fault/error handling transitions within the FEHMs are so much faster than the fault occurrence transitions outside of the FEHMs that, to the other (non-FEHM) states in the model, it appears that a transition into a FEHM exits again nearly instantaneously. The approximation, then, makes the assumption that the exit from the FEHM is actually instantaneous rather than only nearly instantaneous. The greater the difference in magnitude between the fast FEHM transitions and the slow non-FEHM transitions, the faster the actual exit from the FEHM will be in the original model in relative terms, so the closer the approximation assumption will be to reality, and the closer the approximate answer obtained by evaluating the simplified model will'be to the actual answer obtained by evaluating the original model. To apply the approximation, the FEHM is solved by itself, in isolation from the rest of the overall model, to find the probabilities of reaching the individual exiting transitions leading out of the FEHM (i.e. to find the probability of reaching each FEHM absorbing state at $t = \infty$). The resulting FEHM exit probabilities (which are now coverage factors) are substituted into the original model in place of the states that were in the FEHM. This has the effect of not only reducing the number of states in the model which must be solved, but also eliminating all the fast transitions from the model as well (i.e., removing stiffness from the model).

It must be emphasized that this procedure is an *approximation* technique, that the evaluation result of the simplified model definitely *will be different* than that of the original model, and that the closeness of the evaluation of the simplified model to that of the original model depends on the validity of the assumption that the fast FEHM transitions and the slow non-FEHM transitions differ greatly in magnitude. If the fast and slow transitions are sufficiently close in magnitude, then the approximation will not be very good, and the evaluation of the simplified model will not be very close to the evaluation of the original model. There have been a number of efforts aimed at establishing bounds for the accuracy of the behavioral decomposition approximation[28-30].

An example of the application of behavioral decomposition is illustrated in the slide. On the left is the original Markov model of three redundant components, in which two groups of states model the response of the system to the occurrences of faults. The states in these groups appear inside boxes labeled as FEHMs. The system initially resides in the state labeled *{3}*. After a time one of the three components may experience a fault, causing the system to move into the top FEHM. Once a fault occurs it must first be detected, then the detected fault must be isolated (i.e., the system must decide which of the three components is faulty), and a reconfiguration must occur to switch the faulty component out of operation, replacing it with one of the spares. If any of these steps is unsuccessful, it will cause the system to fail and move to the FEHM absorbing state labeled ***{coverage failure}***. If the reconfiguration is successful, the system reaches the FEHM absorbing state labeled ***{successful reconfiguration}***, from which it exits the FEHM and moves to the state labeled *{2}* where it continues normal operation. To anyone looking at the Markov model within the time frame of the fault occurrences (i.e., the time scale of the holding times of states *{3}*, *{2}*, and *{1}*), it will seem that once a transition finally occurs out of state *{3}* into the corresponding FEHM, a sequence of transitions within the FEHM occurs so rapidly that the system almost immediately ends up either in state *{2}* or the FEHM state labeled ***{coverage failure}***. If the system ends up in state *{2}*, the whole process is repeated when a second component experiences a fault which causes the system to move into the bottom FEHM.

On the right side of the slide is the simplified Markov model that results from applying the behavioral decomposition approximation. The states of the FEHM between state *{3}* and state *{2}* in the original model are replaced by a probabilistic branch point and removed from the simplified model. The probabilistic branch point has two paths: a branch leading to state *{2}* that may be taken with probability c_1 (where c_1 is determined by solving the Markov model comprised of the states in the FEHM in isolation from the rest of the overall original model to determine the probability of reaching the state labeled ***{successful reconfiguration}*** in the steady state, i.e. at $t = \infty$), and a branch leading to a state representing a coverage failure that may be taken with probability $1 - c_1$ (where $1 - c_1$ must necessarily be the probability of reaching the FEHM state labeled ***{coverage failure}*** in the steady state). These coverage probabilities are then incorporated into the simplified model as shown in the slide. If the system arrives in state *{2}* and subsequently experiences a second component failure, the process is repeated using the bottom FEHM. Note that the different transition rates leading into the top and bottom FEHMs will cause the exit probabilities to differ between them, i.e. the coverage probability for exiting the top FEHM by a successful reconfiguration (c_1) in general will be different than the corresponding successful reconfiguration exit probability for the bottom FEHM (c_2). This is shown in the simplified model in the slide.

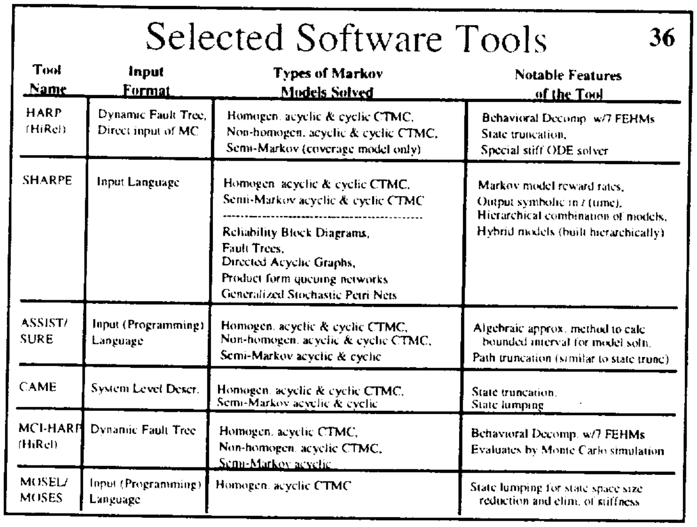
To summarize, it is not crucial for the reader to fully understand all of the details of behavioral decomposition as described here. However, the important facts to remember are that behavioral decomposition, if applicable, can reduce the number of states in the Markov model <u>*and*</u> eliminate stiffness from the model at the same time.

Selected Software Tools for Markov Modeling

The last several years has seen the development of several software tools for performing dependability analysis that incorporate the results of recent research in state-of-the-art methods in Markov modeling. Many of these programs address the topics that have been discussed in the past several slides. The next slide summarizes the key characteristics and features of several such software tools. Included in the summary are the form of the input (model type(s), etc.), which of the types of Markov models covered in this tutorial can be solved by the software tool, and distinguishing features (if any) that help to differentiate it from the other tools. It should be noted that each of these tools was designed to be most efficient with greatest utility in certain specific modeling areas, and that no one program currently exists that will satisfy all uses with the same degree of efficiency, utility, and ease of use. There is some overlap in capability for most of these programs, but some do a much better job than others for specific applications. The majority of these tools were developed at universities or under the sponsorship of the federal government (NASA) and so are available to the general public for use.

Slides 36 and 37: Summary of Selected Software Tools for Markov Model-based Dependability Analysis

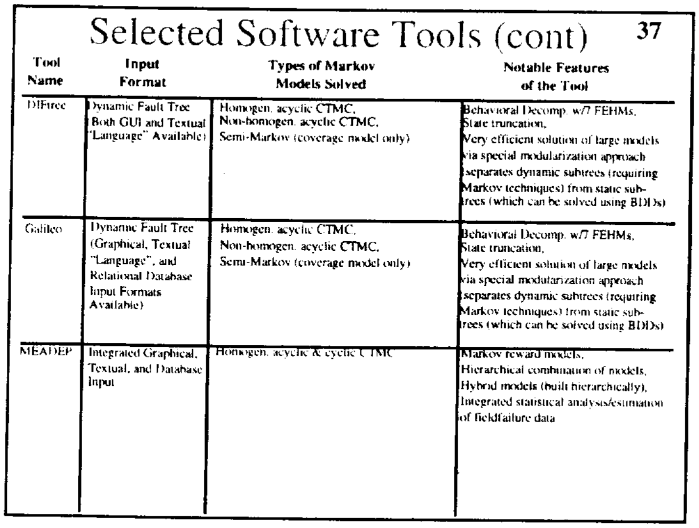
The ***Hybrid Automated Reliability Predictor (HARP)*** program[18] is a descendent of an earlier reliability analysis program called CARE III[17] and was developed to address some of the limitations in the CARE III program. Input of a model may be in one of two forms: either directly in the form of a Markov model (i.e. listing of states and inter-state transitions), or in the form of a *dynamic fault tree*[3, 4]. If a dynamic fault tree is used, it is converted automatically to an acyclic Markov model before being solved[21]. If the model is specified directly in the form of a Markov chain instead of as a dynamic fault tree, then the model solved by HARP can be either cyclic or acyclic. The Markov model can be either homogeneous or non-homogeneous regardless of which form of input is used. In addition to the Markov chain or dynamic fault tree, the user must also provide fault/error handling information (parameters for FEHMs) as input to the program if behavioral decomposition is to be used. Whereas CARE III provided only two types of FEHM coverage models that could be used when employing behavioral decomposition, HARP allows the user to select from seven different FEHM coverage models (which can be mixed and matched). If the input is in the form of a dynamic fault tree, the user has the option of using state truncation to limit the number of states in the generated Markov model. If the Markov model that is solved in the final step of the HARP analysis contains stiffness, special ODE solution routines that are designed for solving stiff ODEs are automatically invoked instead of the usual ODE solution routines. A graphical user interface (GUI) is available for model input and graphical output analysis on Sun workstations and PC clone systems[31]. An older textual user interface is also available. HARP was developed at Duke University under the sponsorship of NASA Langley Research Center. HARP has been combined together with several related reliability analysis programs into a reliability analysis package called HiRel[19] which is available for general use. Persons interested in obtaining a copy of HiRel may order it through NASA's COSMIC software distribution organization at the following addresses: COSMIC, University of Georgia, 382 E. Broad St., Athens, GA 30602-4272; phone: (706) 524-3265; email: service@cosmic.uga.edu.



Selected Software Tools 36

Tool Name	Input Format	Types of Markov Models Solved	Notable Features of the Tool
HARP (HiRel)	Dynamic Fault Tree, Direct input of MC	Homogen. acyclic & cyclic CTMC, Non-homogen. acyclic & cyclic CTMC, Semi-Markov (coverage model only)	Behavioral Decomp w/7 FEHMs State truncation, Special stiff ODE solver
SHARPE	Input Language	Homogen. acyclic & cyclic CTMC, Semi-Markov acyclic & cyclic CTMC -------------------------------- Reliability Block Diagrams, Fault Trees, Directed Acyclic Graphs, Product form queuing networks Generalized Stochastic Petri Nets	Markov model reward rates, Output symbolic in *t* (time), Hierarchical combination of models, Hybrid models (built hierarchically)
ASSIST/ SURE	Input (Programming) Language	Homogen. acyclic & cyclic CTMC, Non-homogen. acyclic & cyclic CTMC, Semi-Markov acyclic & cyclic	Algebraic approx. method to calc bounded interval for model soln. Path truncation (similar to state trunc)
CAME	System Level Descr.	Homogen. acyclic & cyclic CTMC, Semi-Markov acyclic & cyclic	State truncation, State lumping
MCI-HARP (HiRel)	Dynamic Fault Tree	Homogen. acyclic CTMC, Non-homogen. acyclic CTMC, Semi-Markov acyclic	Behavioral Decomp. w/7 FEHMs Evaluates by Monte Carlo simulation
MOSEL/ MOSES	Input (Programming) Language	Homogen. acyclic CTMC	State lumping for state space size reduction and elim. of stiffness

Slide 36



Selected Software Tools (cont) 37

Tool Name	Input Format	Types of Markov Models Solved	Notable Features of the Tool
DIFtree	Dynamic Fault Tree Both GUI and Textual "Language" Available)	Homogen. acyclic CTMC, Non-homogen. acyclic CTMC, Semi-Markov (coverage model only)	Behavioral Decomp. w/7 FEHMs, State truncation, Very efficient solution of large models via special modularization approach separates dynamic subtrees (requiring Markov techniques) from static subtrees (which can be solved using BDDs)
Galileo	Dynamic Fault Tree (Graphical, Textual "Language", and Relational Database Input Formats Available)	Homogen. acyclic CTMC, Non-homogen. acyclic CTMC, Semi-Markov (coverage model only)	Behavioral Decomp. w/7 FEHMs, State truncation, Very efficient solution of large models via special modularization approach separates dynamic subtrees (requiring Markov techniques) from static subtrees (which can be solved using BDDs)
MEADEP	Integrated Graphical, Textual, and Database Input	Homogen. acyclic & cyclic CTMC	Markov reward models, Hierarchical combination of models, Hybrid models (built hierarchically), Integrated statistical analysis/estimation of fieldfailure data

Slide 37

The ***Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE)*** program [32-34] is an integrated tool that allows many different model types to be solved either individually or combined hierarchically into (hybrid) models. The input to the program is in the form of a generalized model description language. SHARPE can solve homogeneous cyclic and acyclic CTMCs and cyclic and acyclic semi-Markov models. In addition, SHARPE can also solve other types of models, including: reliability block diagrams, fault trees, directed acyclic graphs, product form single chain queuing networks, and Generalized Stochastic Petri Nets (GSPNs). SHARPE has several features that distinguish it from the majority of the other tools discussed here. SHARPE provides the capability to assign *reward rates* to states of a Markov model, producing a *Markov reward model* which can then be used for performance analysis. A unique feature of SHARPE is its ability to produce symbolic output in which the desired answer (for example, the probability of a particular state of a Markov model at a time t) is given symbolically as a function of t (such as, for example, $P(1) = 1 - e^{-2t}$). As a consequence of this symbolic output capability, individual models of independent subsystems (independent with respect to subsystem events, i.e. failure and other events) may be combined together in a hierarchical manner to produce larger composite models. Even models of different types may be combined in this way to produce hybrid composite models. This gives the analyst a great deal of flexibility in building a system model. A limitation of SHARPE is that the model types it solves are fairly basic and do not include some of the enhancement features like behavioral decomposition, state truncation, and automated generation of Markov models that are found in the other tools. However, if these features are not needed, the benefits of symbolic output and hierarchical/hybrid model capability outweigh the lack of model enhancement features. SHARPE was developed at Duke University. Persons interested in obtaining more technical and/or licensing information about SHARPE should contact Dr. Kishor S. Trivedi, 1713 Tisdale St., Durham, NC 27705, phone: (919) 493-6563, internet: kst@egr.duke.edu.

The ***Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST)*** program[22] and the ***Semi-Markov Unreliability Range Evaluator (SURE)*** program[25, 26] are a coordinated pair of programs designed to work together. SURE is a program for evaluating semi-Markov models, whereas ASSIST (which implements an specialized programming language for describing Markov and semi-Markov models) generates a semi-Markov model specification suitable for use as input to the SURE program. The input for ASSIST/SURE is a user-written program in the ASSIST language which ASSIST uses to generate the semi-Markov model. Model types that can be evaluated by SURE include cyclic and acyclic models of all three types (homogeneous, non-homogeneous and semi-Markov). The unique feature that distinguishes ASSIST/SURE from the other tools discussed here is its use of an algebraic method to calculate a bounded interval value for the model solution. This approach allows SURE to avoid having to solve a system of simultaneous ODEs. This approach also differs from the others in that it evaluates probabilities of transition paths through the model rather than probabilities of states. SURE implements a path truncation feature, which is similar to the state truncation technique discussed earlier in this tutorial. Both SURE and ASSIST were developed at NASA Langley Research Center. Persons interested in obtaining a copy of ASSIST/SURE may order it through NASA's COSMIC software distribution organization at the following addresses: COSMIC, University of Georgia, 382 E. Broad St., Athens, GA 30602-4272; phone: (706) 524-3265; email: service@cosmic.uga.edu.

The ***Computer Aided Markov Evaluator (CAME)*** program[23] is a tool whose distinguishing feature is its automatic generation of the Markov model from a system level description entered by the analyst. The system level description includes information about the system architecture, the performance requirements (failure criteria), and information about reconfiguration procedures. From this information, the program automatically constructs an appropriate Markov model. The analyst can monitor and control the model construction process. Homogeneous cyclic and acyclic CTMCs can be generated and solved. Modifications to the original CAME program permit semi-Markov models to be generated using the input format required by the SURE program, permitting cyclic and acyclic semi-Markov models to be generated and solved by a coordinated use of CAME and SURE. CAME also implements state truncation and state aggregation (lumping). CAME was developed at C. S. Draper Laboratories and in the past has not been available for general use outside of Draper Labs. Persons interested in obtaining additional information about CAME should contact Dr. Philip Babcock, C. S. Draper Laboratories, 555 Technology Square, Cambridge, MA 02139.

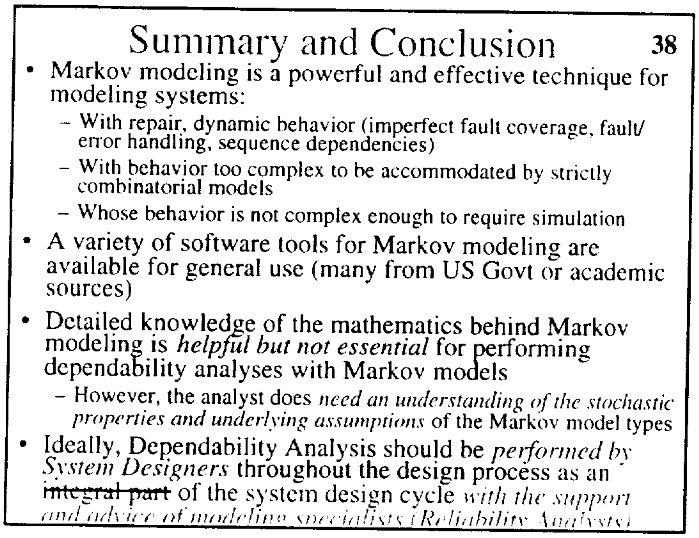
MCI-HARP is a relative of the HARP program mentioned earlier. Input to MCI-HARP is in the form of a dynamic fault tree, and MCI-HARP uses the same input files as the HARP program for dynamic fault trees. Since the input is in the form of a dynamic fault tree, the underlying Markov model that is evaluated is acyclic (no repair). MCI-HARP can evaluate homogeneous CTMCs, non-homogeneous CTMCs, semi-Markov models, and also non-Markovian models (such as those with inter-state transition rates which are functions of global and local time both in the same model). MCI-HARP differs from HARP in that the underlying Markov or non-Markovian model is evaluated using simulation rather than by numerical (analytic) solution techniques. This permits much larger and more complex models to be solved than can be accommodated by HARP (for example, use of component IFR/DFRs and cold spares within the same model [8]), although at a cost of large execution time requirements if the results must be highly accurate[35]. Because it is fully compatible with HARP, MCI-HARP implements behavioral decomposition for modeling imperfect fault coverage exactly as HARP does. The compatibility between the two programs also allows them to be used together in a coordinated way, permitting the analyst to select the appropriate program (analysis method) to analyze the model or partial models depending on model size and characteristics. On models that can be solved with both programs, the analyst has the option of comparing the outputs obtained from the two programs to verify results. MCI-HARP was developed at NASA Ames Research Center by modifying a precursor program, called MC-HARP, that was originally developed at Northwestern University[36]. HARP, MC-HARP, and MCI-HARP are all members of a package of related reliability analysis programs which collectively are called HiRel[19] and which are all available for general use. Persons interested in obtaining a copy of MCI-HARP or any other member program of the HiRel reliability modeling package may order it through NASA's COSMIC software distribution organization at the following addresses: COSMIC, University of Georgia, 382 E. Broad St., Athens, GA 30602-4272; phone: (706) 524-3265; email: service@cosmic.uga.edu.

The ***MOdeling Specification and Evaluation Language (MOSEL) and the MOdeling, Specification, and Evaluation System (MOSES)*** tools are a pair of tools designed to work together. They were developed at University of Erlangen, Germany. MOSEL is a programming-type language for specifying a Markov model, and MOSES is the solver engine that evaluates the Markov model described in MOSEL. The relationship between these two programs is very similar to the relationship between ASSURE and ASSIST programs that were discussed earlier. The MOSEL language seems tailored toward describing queuing systems, but it can also describe Markov models arising from other, more general origins. The MOSES evaluator program has some special evaluation features based on multi-grid methods that implement state aggregation, reduce stiffness, and allow very large models to be evaluated. The MOSEL/MOSES pair of programs are aimed at solving cyclic and acyclic homogeneous Markov models. Persons interested in obtaining more information about MOSEL/MOSES may contact Stefan Greiner by email at Stefan.Greiner@informatik.uni-erlangen.de.

DIFtree is a dependability modeling tool developed at the University of Virginia[11]. Although it is oriented toward building and solving fault trees, it employs a unique modularization technique[12] for identifying and separating out parts of a fault tree that are *dynamic* (i.e. contain sequence dependencies) from those parts that are *static* (i.e. are combinatorial only). The static parts of the fault tree are solved with efficient methods based on Binary Decision Diagrams (BDDs)[9, 10, 37]. The dynamic parts of the fault tree (which are subtrees that essentially are *dynamic fault trees* in their own right) require Markov modeling techniques to solve and use fundamentally the same methodology as that used by HARP[3, 4]. The result is a tool for solving dynamic fault trees that is (potentially) much more efficient than HARP for fault trees that have little or no dynamic behavior that they are modeling. DIFtree accepts as input a dynamic fault tree in either a graphical or a textual form. Because the dynamic fault tree-to-Markov model conversion feature does not provide for repair, the resulting Markov models that correspond to the dynamic sub-fault trees are acyclic. Both homogeneous and non-homogeneous CTMCs can be solved. Imperfect coverage can be accommodated in both the static[9] and the dynamic parts of the fault tree, and DIFtree has the same FEHM submodel handling and state truncation capabilities as HARP. Persons interested in obtaining more information about DIFtree may contact Joanne Bechta Dugan, Dept. of Electrical Engineering, Thornton Hall, University of Virginia, Charlottesville, VA 22903-2242; email: jbd@Virginia.edu. A copy of DIFtree (for Unix hosts) may downloaded by anonymous ftp at csisun15.ee.virginia.edu.

The ***Galileo Fault Tree Analysis Tool*** was also developed at University of Virginia. It runs on PC-based computers under Windows 95 or Windows NT and incorporates the *DIFtree* modeling tool (see above) as its fault tree/Markov model solving engine. It provides an integrated wrapper development environment around DIFtree (based on the standard COTS applications MS Word 97, MS Access 97, and Visio Technical 4.0 - 5.0) for specifying fault trees for solution by DIFtree. Because of its close tie with DIFtree, it can solve the same Markov model types and has the same features as DIFtree (see above). Persons interested in obtaining more information about Galileo may visit its Web site at the following URL: http://www.cs.virginia.edu/~ftree. An alpha version of Galileo can be downloaded from that Web site.

A new dependability modeling tool for Windows 95 and Windows NT named ***MEADEP (MEAsure DEPendability)*** will soon be available from SoHaR, Inc. Like SHARPE and DIFtree (see above), MEADEP also is not strictly a Markov model-based tool only, but Markov models are one of the important modeling options that it offers. MEADEP provides an integrated GUI-based development environment that provides any combination of the following three input options: a graphical-oriented drawing facility for creating Markov models (and Reliability Block diagrams), a textual input capability, and a feature for extracting and integrating field failure data from a Database. MEADEP appears to be oriented toward solving homogeneous cyclic and acyclic CMTCs. It allows a reward rate to be specified for individual states, thereby allowing the use of Markov reward models. A distinctive feature is the capability to hierarchically combine submodels into a composite system model, and the ability to use this feature to build hybrid models (in which submodels are of different types). Another feature is an integrated capability for statistical analysis and estimation from field failure data built in to the tool. Persons interested in obtaining more information about MEADEP may visit SoHaR's MEADEP Web site at the following URL:
http://www.sohar.com/meadep/index.html.



Summary and Conclusion 38

- Markov modeling is a powerful and effective technique for modeling systems:
  - With repair, dynamic behavior (imperfect fault coverage, fault/error handling, sequence dependencies)
  - With behavior too complex to be accommodated by strictly combinatorial models
  - Whose behavior is not complex enough to require simulation
- A variety of software tools for Markov modeling are available for general use (many from US Govt or academic sources)
- Detailed knowledge of the mathematics behind Markov modeling is *helpful but not essential* for performing dependability analyses with Markov models
  - However, the analyst does *need an understanding of the stochastic properties and underlying assumptions* of the Markov model types
- Ideally, Dependability Analysis should be *performed by System Designers* throughout the design process as an ~~integral part~~ of the system design cycle *with the support and advice of modeling specialists (Reliability Analysts)*

Slide 38

Slide 38: Summary and Conclusion

This tutorial has presented an introduction to the use of Markov modeling for dependability analysis for fault tolerant systems. The emphasis has been on giving an intuitive feeling for the capabilities and limitations of the three major types of Markov model types, and how they represent the behavior of a system. It was observed that Markov modeling is an effective technique for modeling systems that exhibit complex repair, dynamic behavior (such as imperfect fault coverage, fault/error handling, and sequence dependencies), and general behavior that is too complex to be accommodated by simpler combinatorial modeling methods but not so complex that simulation is required. It was seen that a number of software tools have been developed for Markov modeling, and that several of them are available for general use. It was noted that detailed knowledge on the part of the dependability analyst of the mathematics behind Markov modeling techniques is helpful but not essential to be able to perform dependability analyses with Markov models provided that appropriate software tools for Markov modeling are available. It is sufficient for the analyst to have an understanding of the stochastic properties and underlying assumptions of Markov modeling, and an understanding of their implications (limitations) for representing system failure behavior in order to be able to use Markov modeling effectively for most common dependability analysis needs.

In this tutorial it was also noted that, ideally, dependability analysis should be performed throughout the entire design process by system designers (whenever possible) instead of exclusively by modeling specialists because it is ultimately the system designers that are most qualified to perform the dependability analysis by virtue of their familiarity with the technical details of the system. The availability of software tools for modeling such as those described in this tutorial helps make this approach feasible. However, it should be emphasized that in such an approach there is still an important place for the modeling specialist (i.e. reliability analyst) in the role of assisting the system designers with understanding subtleties in the modeling process and verifying that a completed model does not use any modeling techniques inappropriately. This is needed because dependability modeling is still as much an art as it is a science, and there are limits to the effectiveness of the automation of the modeling process that these tools provide. This is especially true when relatively sophisticated modeling methods, such as Markov modeling, are used. In addition, the current state-of-the-art modeling tools generally do not yet have comprehensive safeguards to prevent an inexperienced user from inappropriate use of modeling techniques. In light of these facts, it is still wise to rely on experienced human expertise when finalizing any dependability model which has a major impact on the ultimate design of a complex fault tolerant system.

References

[1] J.-C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," presented at IEEE International Symposium on Fault-Tolerant Computing, FTCS-15, 1985.

[2] M. A. Boyd and D. L. Iverson, "Digraphs and Fault Trees: A Tale of Two Combinatorial Modeling Methods," presented at 1993 Reliability and Maintainability Symposium, Atlanta, GA, 1993.

[3] M. A. Boyd, "Dynamic Fault Tree Models: Techniques for Analysis of Advanced Fault Tolerant Computer Systems," in *Ph.D. Thesis, Department of Computer Science*. Durham: Duke University, 1990.

[4] J. B. Dugan, S. Bavuso, and M. A. Boyd, "Fault Trees and Sequence Dependencies," presented at 1990 Reliability and Maintainability Symposium, Los Angeles, CA, 1990.

[5] R. A. Howard, *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. New York: John Wiley and Sons, 1971.

[6] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[7] M. A. Boyd and J. O. Tuazon, "Fault Tree Models for Fault Tolerant Hypercube Multiprocessors," presented at 1991 Reliability and Maintainability Symposium, Orlando, FL, 1991.

[8] M. A. Boyd and S. J. Bavuso, "Modeling a Highly Reliable Fault-Tolerant Guidance, Navigation, and Control System for Long Duration Manned Spacecraft," presented at AIAA/IEEE Digital Avionics Systems Conference, Seattle, WA, 1992.

[9] S. A. Doyle, J. B. Dugan, and M. A. Boyd, "Combinatorial Models and Coverage: A Binary Decision Diagram (BDD) Approach," presented at 1995 Reliability and Maintainability Symposium, Washington, DC, 1995.

[10] S. A. Doyle, J. B. Dugan, and M. A. Boyd, "Combining Imperfect Coverage with Digraph Models," presented at 1995 Reliability and Maintainability Symposium, Washington, DC, 1995.

[11] J. B. Dugan, B. Venkataraman, and R. Gulati, "DIFtree: A Software Package for the Analysis for Dynamic Fault Tree Models," presented at 1997 Reliability and Maintainability Symposium, Philadelphia, PA, 1997.

[12] R. Gulati and J. B. Dugan, "A Modular Approach for Analyzing Static and Dynamic Fault Trees," presented at 1997 Reliability and Maintainability Symposium, Philadelphia, PA, 1997.

[13] J. B. Dugan and S. A. Doyle, "Tutorial: New Results in Fault Trees," presented at 1996 Reliability and Maintainability Symposium, Las Vegas, NV, 1996.

[14] J. B. Dugan and S. A. Doyle, "Tutorial: New Results in Fault Trees," presented at 1997 Reliability and Maintainability Symposium, Philadelphia, PA, 1997.

[15] A. L. Reibman, "Modeling the Effect of Reliability on Performance," *IEEE Transactions on Reliability*, vol. R-39, pp. 314-320, 1990.

[16] E. J. Henley and H. Kumamoto, *Probabilistic Risk Assessment: Reliability Engineering, Design, and Analysis.* New York: IEEE Press, 1992.

[17] J. J. Stiffler, "Computer-Aided Reliability Estimation," in *Fault-Tolerant Computing: Theory and Techniques*, vol. 2, D. K. Pradhan, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1986, pp. 633-657.

[18] J. B. Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist, "The Hybrid Automated Reliability Predictor," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 9, no. 3, pp. 319-331, 1986.

[19] S. J. Bavuso and J. B. Dugan, "HiRel: Reliability/Availability Integrated Workstation Tool," presented at 1992 Reliability and Maintainability Symposium, 1992.

[20] G. Ciardo, "Analysis of Large Stochastic Petri Net Models," in *Department of Computer Science.* Durham, NC: Duke University, 1989.

[21] M. A. Boyd, "Converting Fault Trees to Markov Chains for Reliability Prediction," in *Department of Computer Science.* Durham, NC: Duke University, 1986.

[22] S. C. Johnson, "ASSIST Users Manual," NASA Langley Research Center, Technical Memorandum 877835, August 1986.

[23] G. Rosch, M. A. Hutchins, and F. J. Leong, "The Inclusion of Semi-Markov Reconfiguration Transitions into the Computer-Aided Markov Evaluator (CAME) Program," presented at AIAA/IEEE Digital Avionics Systems Conference, San Jose, CA, 1988.

[24] A. L. Reibman and K. S. Trivedi, "Numerical Transient Analysis of Markov Models," *Computers and Operations Research*, vol. 15, no.1, pp. 19-36, 1988.

[25] R. W. Butler, "The Semi-Markov Unreliability Range Evaluator (SURE) Program," NASA Langley Research Center, Technical Report July 1984.

[26] R. W. Butler, "The SURE2 Reliability Analysis Program," NASA Langley Research Center, Technical Report 87593, January 1985.

[27] M. Aoki, "Control of Large-Scale Dynamic Systems by Aggregation," *IEEE Transactions on Automatic Control*, vol. AC-13, pp. 246-253, 1968.

[28] J. McGough, M. K. Smotherman, and K. S. Trivedi, "The Conservativeness of Reliability Estimates Based on Instantaneous Coverage," *IEEE Transactions on Computers*, vol. C-34, no. 7, pp. 602-609, 1985.

[29] J. McGough, R. M. Geist, and K. S. Trivedi, "Bounds on Behavioral Decomposition of Semi-Markov Reliability Models," presented at 21st Fault Tolerant Computer Symposium, 1991.

[30] A. L. White, "An Error Bound for Instantaneous Coverage," presented at 1991 Reliability and Maintainability Symposium, 1991.

[31] S. J. Bavuso and S. Howell, "A Graphical Language for Reliability Model Generation," presented at 1990 Reliability and Maintainability Symposium, Los Angeles, CA, 1990.

[32] R. A. Sahner, "Hybrid Combinatorial-Markov Methods for Solving Large Performance and Reliability Models," in *Department of Computer Science.* Durham, NC: Duke University, 1985.

[33] R. A. Sahner and K. S. Trivedi, "Reliability Modeling Using SHARPE," *IEEE Transactions on Reliability*, vol. R-36, pp. 186-193, 1987.

[34] R. A. Sahner and K. S. Trivedi, "A Software Tool for Learning About Stochastic Models," *IEEE Transactions on Education*, 1993.

[35] M. A. Boyd and S. J. Bavuso, "Simulation Modeling for Long Duration Spacecraft Control Systems," presented at 1993 Reliability and Maintainability Symposium, Atlanta, GA, 1993.

[36] M. E. Platt, E. E. Lewis, and F. Boehm, "General Monte Carlo Reliability Simulation Including Common Mode Failures and HARP Fault/Error Handling," The Technical Institute of Northwestern University, Technical report January 1991.

[37] S. A. Doyle, "Dependability Analysis of Fault Tolerant Systems: A New Lock at Combinatorial Modeling," in *Dept. of Computer Science.* Durham, NC: Duke University, 1995, pp. 279.