Fault Tolerant Computing
CS 530
Midterm Review

Yashwant K. Malaiya
Colorado State University

Fault Tolerant Computing: Midterm Review

Midterm:

- Sec 001 Th Mar 11, 3:30-4:45 PM
  - Also for Sec 801 Local students, not working full time
- Sec 801 distance students Mar 11 3:30-Mar 12 4:45PM
- Respondus Lockdown browser
  - Closed book/notes
  - Built-in Scientific calculator in browser
  - One blank sheet permitted
    - Show both side at the beginning and at the end
    - Destroy on camera
- Formula sheet? No.

How to prepare for the Midterm

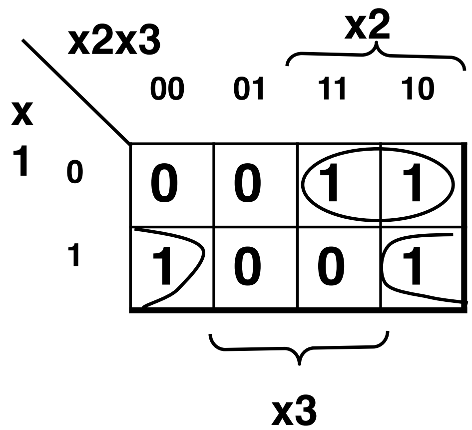
- **Attentively attend lectures**
  - **linking concepts and methods critically**
- **Quizzes. Find out why.**
- **PSA1: Ask why.**
- **You should be able**
  - **Solve similar and related problems.**
  - **Explain why.**
  - **Apply principles to solve new problems.**

Topics

- Terminology and ideas
- Digital Systems, Fault Modeling
- Combinational & Sequential Circuit Testing
- Probabilistic Methods
- Random Testing
- Reliability: combinatorial and time dependent
- Software Reliability:
  - Static modeling, Module size, SRGM

Combinational Example

x1	x2	x3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

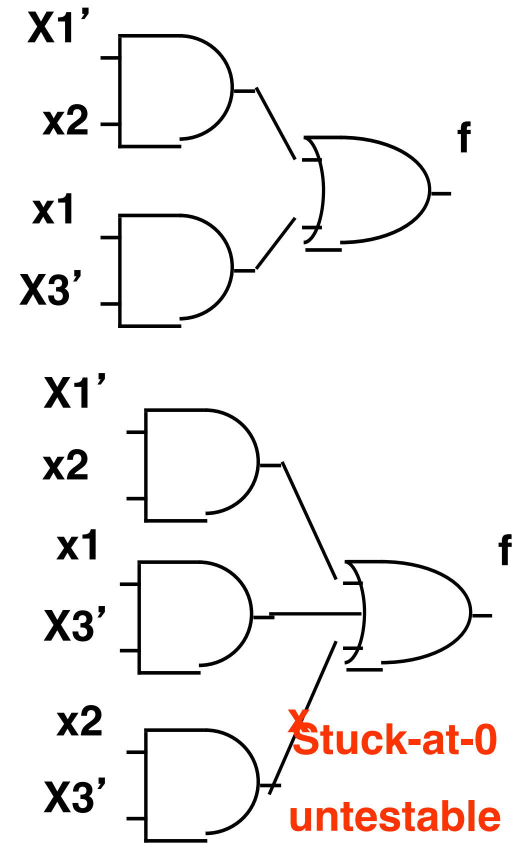


Implementation A

F=x1' x2 + x1x3'

Implementation B

F=x1' x2 + x1x3' + x2x3'



redundant

Note that the x2x3' term is redundant.

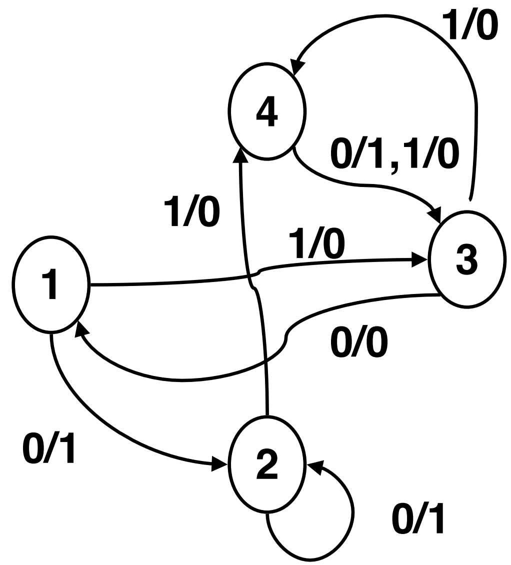
Here a prime indicates complement.

A stuck-at-0 fault makes the line always stay at 0 regardless of what it is supposed to be.

FSM Description

state \ Input x	0	1
1	2,1	3,0
2	2,1	4,0
3	1,0	4,0
4	3,1	3,0

Entries: N,Z



Format:
x/z

given

input	0	1	0
state	1	2	4
output	1	0	1

Based on FSM state table

(or state diagram)

Stuck-at 0/1 Model

- Classical model, well developed results/methods
  - Many opens and shorts result in a node getting stuck-at a 0 or 1.
- May not describe some defects in today's VLSI.
  - still a nice way of structural "probing". Covering all stuck-at 0/1 will result in covering a large fraction of all faults.
- Model: any one or more of these may be stuck at 0 or 1: *a gate input, a gate output, a primary input*.
- Justification: many lower level defects can be shown to have an equivalent effect.

Common abbreviations:

s-a-0, s-a-1

Single Fault Assumption

- Assumption: only one fault is present at a time.
- Significantly reduces complexity.
- Good for fault detection: *complete single stuck test set* will detect almost all multiple faults.
- Not good for fault location.
- A Multiple fault is a simultaneous presence of several single faults.
- How many *multiple faults in a unit*?
  - Assume k lines
  - 3 states per line: normal, s-a-0, s-a-1
  - Total 3^k-1 faulty situations! (For k=1000, total 1.3×10^{477})

One among 3^k situations is a normal unit.

Test coverage

- A single test typically covers (i.e. tests) for several potential faults.
- The coverage obtained by a test-set can be obtained using fault simulators for hardware.
- The test coverage achieved by a test-set is given by ratio:

$$\text{coverage} = \frac{\text{Number of faults covered}}{\text{Total number of possible faults}}$$

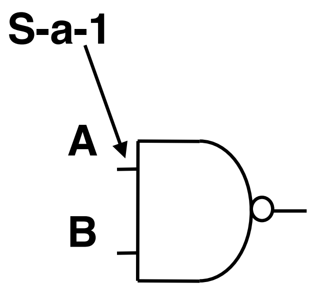
- By convention, coverage is evaluated for stuck-at 0/1 faults in hardware, often given in percentage.

Test generation: Some Basics (2)

- **All tests are contained in T, where T = $f \oplus f_\alpha$**

i.e. T is the set of vectors for which normal and faulty outputs are different.

Example:



A \ B	0	1
0	1	1
1	1	0

f = (AB)’

A \ B	0	1
0	1	0
1	1	0

f_α = B’

A \ B	0	1
0	0	1
1	0	0

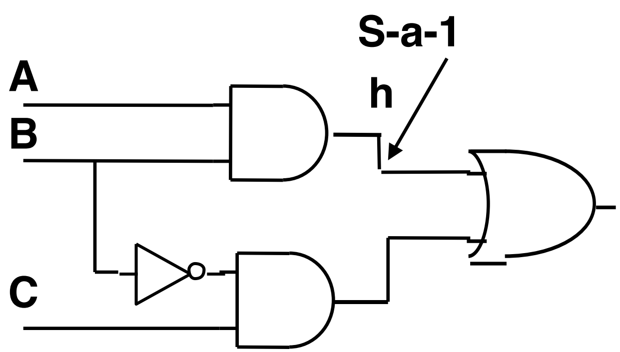
$f \oplus f_\alpha$

T = A’B (01) is a test. The only test.

$f \oplus f\alpha$ is 1 for combinations for which Karnaugh maps of f and $f\alpha$ are different.

Boolean Difference: Internal Nodes

- Consider an internal node h=h(X) s-a-1. Express the original function f(X) as $f_h(X,h)$. Tests for h s-a-1 are given by $\bar{h}(X)\, df_h(X,h)/dh$.



$f(A,B,C)=AB+\bar{B}C \quad h(A,B)=AB$

$f_h(B,C,h)=h+\bar{B}C$

$df_h/dh \;=\; f_h(0,B,C)\oplus f_h(1,B,C) = (\bar{B}C)\oplus 1$

$= \overline{\bar{B}C} \;=B+\bar{C}$

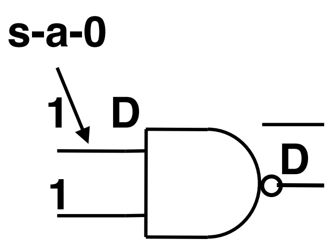
$T = \bar{h}\, df_h/dh = \overline{(AB)}(B+\bar{C}) = (\bar{A}+\bar{B})(B+\bar{C}) = \bar{A}B+\bar{A}\bar{C}+\bar{B}\bar{C}$

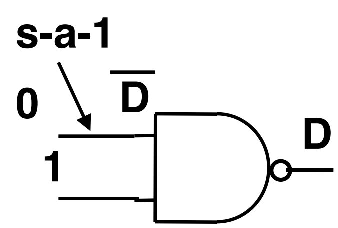
=010, 011, 000, 100 (four vectors!)

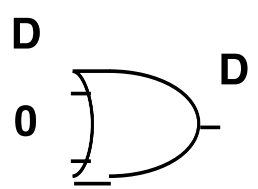
A \ BC	00	01	11	10
0	1	0	1	1
1	1	0	0	0

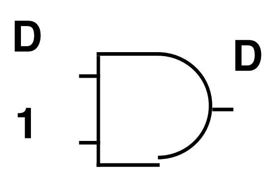
D-Notation

- Notation: Line has value D if it is 1 normally and 0 in presence of the fault. Line has value $\overline{D}$ if it is 0 normally and 1 in presence of the fault.





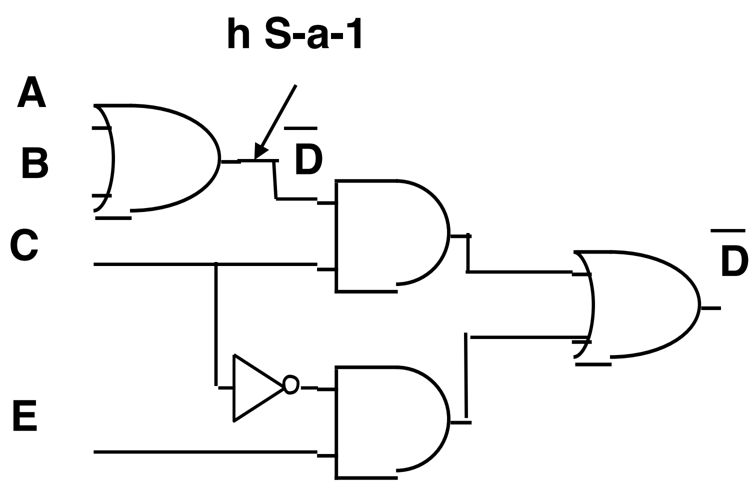




Rules of error propagation:

Gate	All other inputs
AND, NAND	1
OR, NOR	0
XOR	0, 1

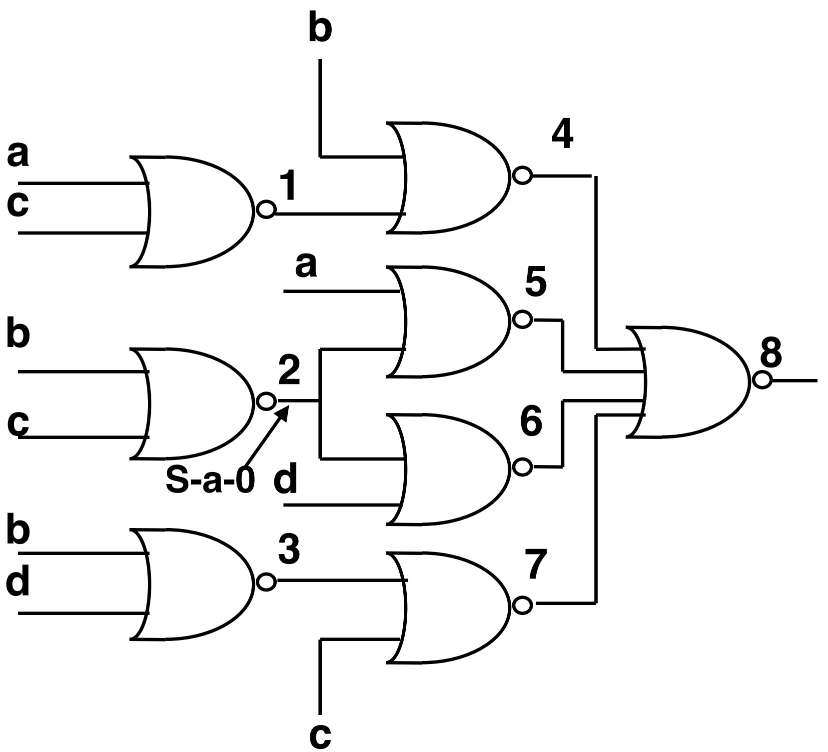
Single Path Propagation



Single path propagation attempts to propagate error using a single path from the fault site to an output.

- **Excitation:**
  - h=0 normally. Need A,B=0,0
- **Propagation:**
  - Other AND input:1
  - Other OR input: 0
- **Justification:**
  - C=1 already. E=x (*don't care*)
- **Test is (0,0,1,x)**

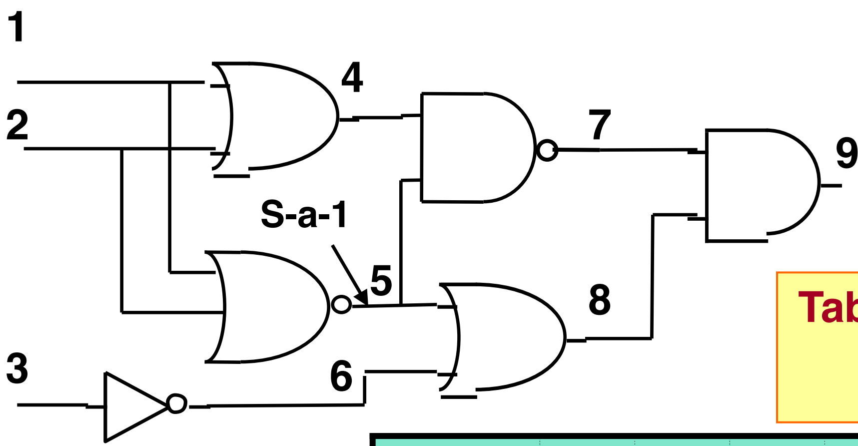
Schneider's Counterexample



- Multiple path propagation thru 5 and 6 works!
- b,c=0,0; a,d=0,0 Thus (0,0,0,0) is a test.

- Try single path 2-6-8
- Excitation: D at 2: b,c=0,0
- Forward trace:
  - $\overline{D}$ at 6: d=0
  - D at 8: 4,5,7=0,0,0
- Implication:
  - Since b=d=0, 3=1, 7=0
- Line Justification (backward trace):
  - For 5=0: a=1
  - Hence 1=0, 4=1 (!)
  - Inconsistency.
- Single path propagation fails.

D-Algorithm Ex (part 2)



Try: path 5-8-9

Table gives step-by-step values, until an inconsistency is observed

Step	1	2	3	4	5	6	7	8	9
Initial	1	0			$\bar{D}$				
5→8	1	0			$\bar{D}$	0		$\bar{D}$	
8→9	1	0			$\bar{D}$	0	1	$\bar{D}$	$\bar{D}$
4←7	1	0		0	$\bar{D}$	0	1	$\bar{D}$	$\bar{D}$
3←6	1	0	1	0	$\bar{D}$	0	1	$\bar{D}$	$\bar{D}$
1,2←4	ϕ	0	1	0	$\bar{D}$	0	1	$\bar{D}$	$\bar{D}$

D-drive

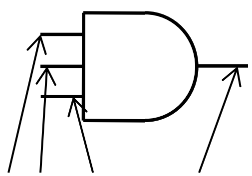
Justifi-cation

Inconsistency!

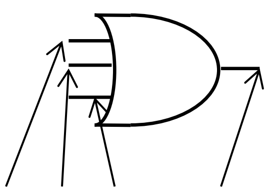
Need to Backtrack

Fault Collapsing (2)

- **Equivalence**: **Faults α and β are equivalent if $f_\alpha = f_\beta$. Then α and β affect the output in exactly the same way.**



All s-a-0 equivalent



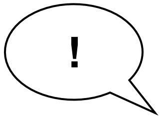
All s-a-1 equivalent

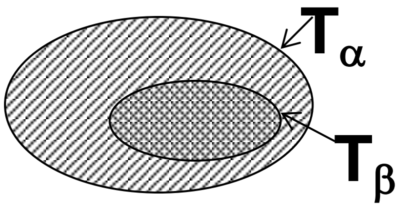
- **For an N-input gate only n+2 faults need to be considered**
- **Ex: NAND gate: we only need to consider**
  - **Any input s-a-0 or output s-a-1 (count as 1)**
  - **One input s-a-1 (total n such inputs)**
  - **Output s-a-0 (1)**
- **Termed *Equivalence fault collapsing***

Fault Collapsing (2)

- **Dominance**: A fault α dominates fault β if $T_\beta \subset T_\alpha$.

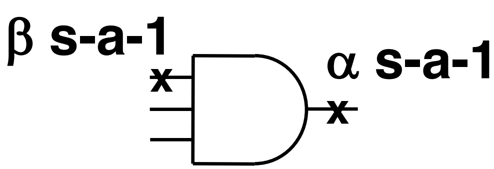
  For detection only fault β needs to be considered. For location, both need to be considered separately (if distinguishable)





Detection only attempts to identify that the unit under test is faulty.

Example:



T_α= 0xx, x0x, xx0

T_β= 011

$\therefore T_\beta \subset T_\alpha$

(0,1,1) will test for both α and β. No need to use other tests if only detection is needed.

Fault Collapsing: Check-points (2)

- **Theorem:** In a combinational circuit, any test set that detects all stuck faults on
  - all primary inputs and
  - All branches of fanout points

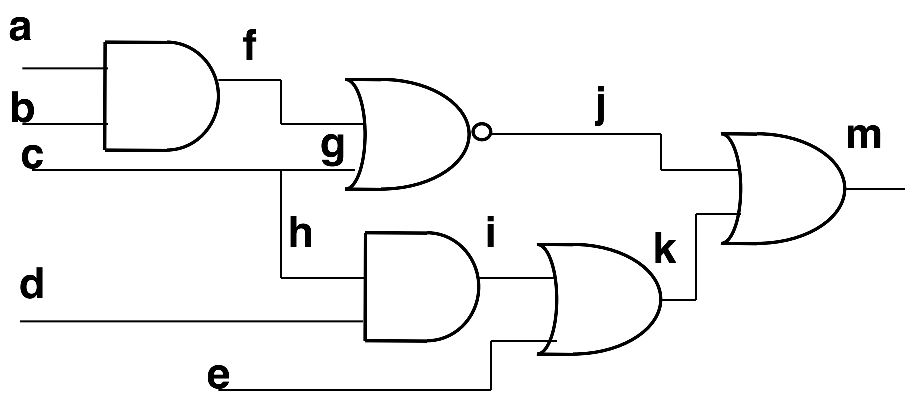
  will detect all stuck faults in the network.

These are appropriately called Checkpoints

Incidentally a check-point concept is also applicable for software testing

H. Yin, Z. Lebne-Dengel and Y. K. Malaiya, " Automatic Test Generation using Checkpoint Encoding and Antirandom Testing" Int. Symp. on Software Reliability Engineering, 1997, pp. 84-95.

Checkpoints: Example



- 12 nodes, two faults at each node (s-a-0, s-a-1) thus 24 faults before collapsing.
- Checkpoints are:
  - Primary inputs: a,b,c,d, e
  - All branches of fan-out points: g,h
  - Faults at checkpoints 7x2=14 faults
- Thus only 14 out of 24 need to be considered.

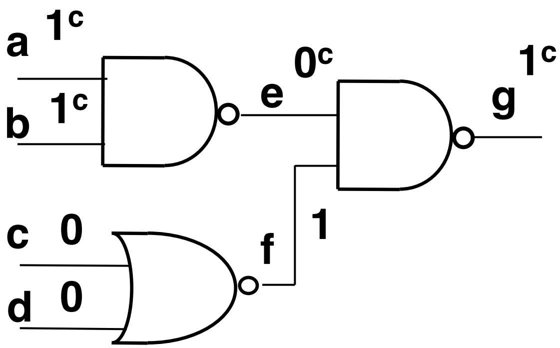
Why Test Set Reduction works

Generally one pattern tests for several faults, because

- With a given vector, several nodes will be critical.

A node is critical if a change in its logic value will change the output.

Example: Here the critical nodes are marked with a c. A node is critical only under a specific input vector, here (1,1,0,0).

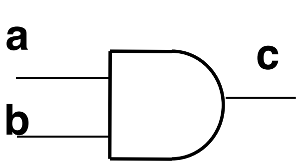


(1100) will detect a s-a-0, b s-a-0, e s-a-1 and g s-a-0

Test Set Compaction

- **Minimize the number of patterns.**

Example:



tests \ faults	a-0	a-1	b-0	b-1	c-0	c-1
00						√
01		√				√
10				√		√
11	√		√		√	

Minimum set

Answer: 01, 10,11 will test for all the faults. Thus no need to apply 00.

In practice heuristics are used, complete optimization is not needed.

Fault distinction

- **Preset test set:** no decision making during testing
- **Adaptive:** successive narrowing down

Problem: There is one fault. Is it f1, f2 or f3?

Fault	Test t_1	Test t_2	Test t_3
f_1	tests	doesn't	tests
f_2	tests	tests	doesn't
f_3	doesn't	tests	tests

- **Preset approach:**
  - **Get response to t_1, t_2, t_3**
  - **Then Identify.**
- **Adaptive: Apply t_1**

No detection **Detection**

$\therefore f_3$ **Apply t_3**

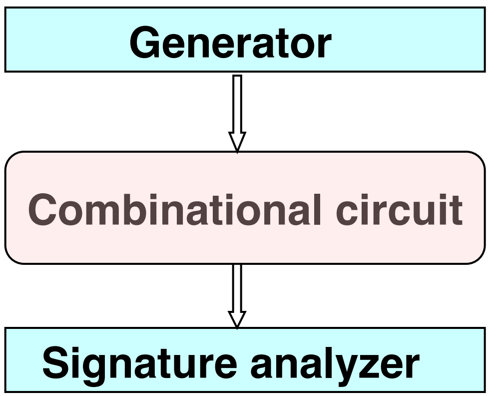
Detection **No det.**

f_1 **f2**

Assuming equal probability 1/3 for each fault, average number of tests

to identify the fault= 2x 1/3+2x1/3+1 x1/3 = 1.7 vectors!

BIST (Built-in self-test)



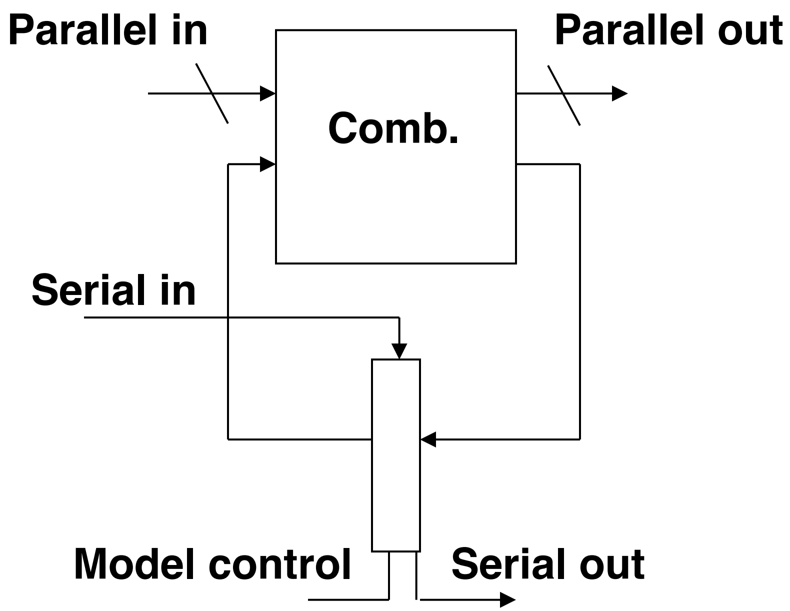
ALFSR: autonomous linear feedback shift register.

Better generators include our antirandom test generator.

- Generator generates pseudorandom vectors. Often an ALFSR.
- Signature analyzer compresses all successive responses into a signature. Usually an LFSR.
- Compared with known good signature.
- Aliasing probability: prob. that a bad circuit can result in good signature. Generally very small.

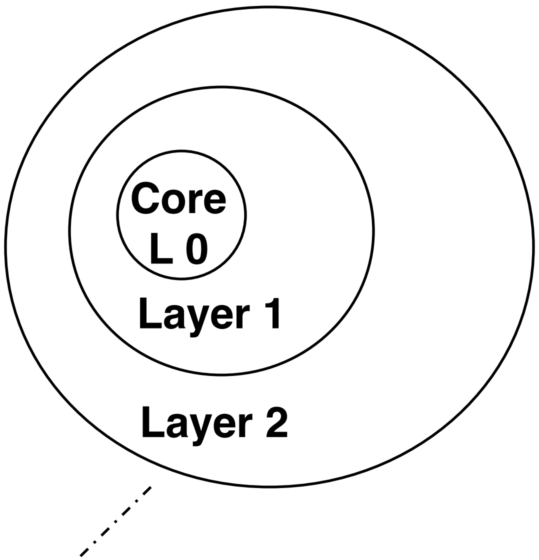
Sequential Circuits with feedback: Scan-chain approach

- **Design for testability**: feedback-less during testing. The flip-flops can be configured to form **scan-chains**.



- **Scan Design**: modes
  - **Normal mode**: parallel in/out
  - **Test mode**: serial in/out
- Sequence of operations
  - Scan a vector: test mode
  - Latch response: normal mode
  - Scan response out: test mode
- If scan-chain too long
  - Use Multiple chain
  - Use **Partial scan**

Incremental Testing Approach



D Brahme, JA Abraham, Functional Testing of Microprocessors, IEEE Trans Comp, Jun 1984, pp. 475- 485.

- Partition system into layers such that layer i can be exercised using only layers 0, .. i-1.
- Test components in each layer in the sequence L0, L1,..Ln.
- Layering may require
  - Assumptions
  - Disabling feedback during testing
- Proofs of complete coverage can be constructed.
- Fault isolation can be done.

Distributions, Binomial Dist.

- **Note that** $$\int_{x\min}^{x\max} f(x)dx = 1 \qquad \sum_{i\min}^{i\max} p(x_i) = 1$$
- **Major distributions:**
  - **Discrete: Bionomial, Poisson**
  - **Continuous: Gaussian, expomential**
- **Binomial distribution: outcome is either** success or failure
  - **Prob. of *r* successes in *n* trials, prob. of one success being *p***

$$f(r) = \binom{n}{r} p^r (1-p)^{n-r} \quad for \quad r = 0, \ldots, n$$

incidentally $$\binom{n}{r} = {}^nC_r = \frac{n!}{r!(n-r)!}$$

Distributions: Poisson

- **Poisson: also a discrete distribution, λ is a parameter.**

$$f(x) = \frac{\lambda^{x} e^{-\lambda}}{x!}$$

- **Example: μ = occurrence rate of something.**
  - **Probability of r occurrences in time t is given by**

$$f(r) = \frac{(\mu t)^{r} e^{-\mu t}}{r!}$$

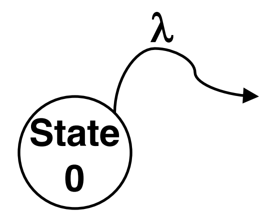
Often applied to fault arrivals in a system

Exponential & Weibull Dist.

Exponential Distribution: is a continuous distribution.

- Density function

  $f(t) = \lambda e^{-\lambda t} \qquad 0 < t \leq \infty$

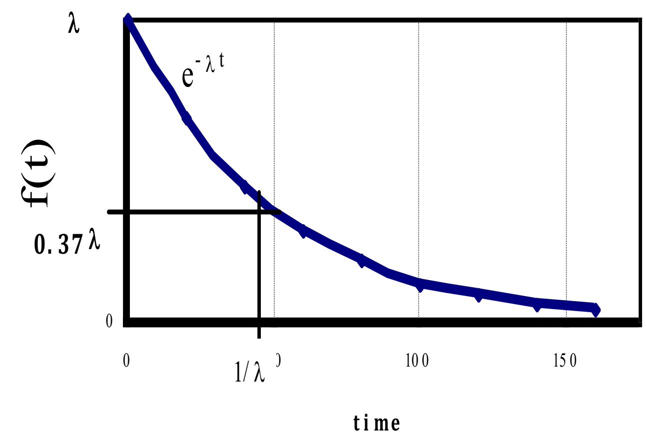


Example:

- λ: exit or **failure rate**.
- Pr{exit the good state during (t, t+dt)}

  $= e^{-\lambda t} \lambda \, dt$

- The **time T spent in good state has an exponential distribution**



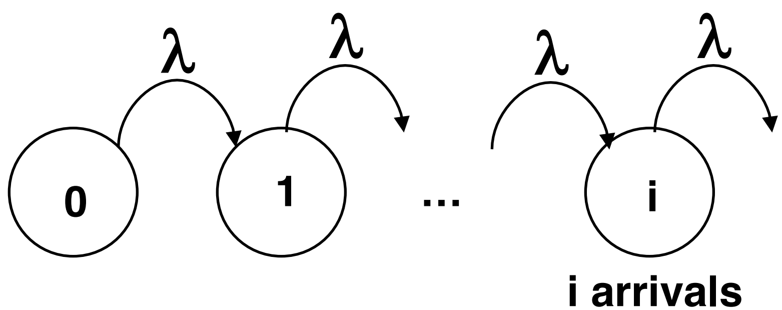
- **Weibull Distribution**: is a 2-parameter generalization of exponential distribution. Used when better fit is needed, but is more complex.

Poisson Process: properties

- **Poisson process: ** A Markov counting process N(t), $t \geq 0$, N(t) is the number of arrivals up to time t.
- Properties of a Poisson process:
  - **N(0) = 0**
  - **P{an arrival in time Δt} = $\lambda \Delta t$**
  - **No simultaneous arrivals**
- We will next see an important example. Assuming that arrivals are occurring at rate λ, we will calculate probability of n arrivals in time t.

Poisson process: analysis

- A process is in state I, if I arrivals have occurred.
- $P_i(t)$ is the probability the process is in state i.



- In state i, probability is flowing in from state i-1, and is flowing out to state i+1, in both cases governed by the rate λ. Thus

$$\frac{dP_i(t)}{dt} = -\lambda P_i(t) + \lambda P_{i-1}(t) \quad n = 0,1,..$$

We'll solve it first for $P_0(t)$, then for $P_1(t)$, then …

Poisson Process: General solution

We need to solve $$\frac{dP_i(t)}{dt} = -\lambda P_i(t) + \lambda P_{i-1}(t) \quad n = 0,1,..$$

Using the expression for $P_0(t)$, we can solve it for $P_1(t)$.

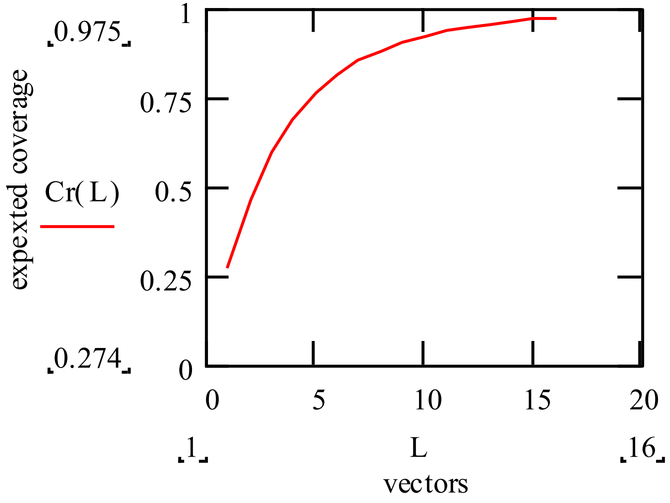
$$\text{Solving recursively, we get}$$

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad n = 0,1,..$$

Which we know is Poisson distribution!

Coverage Achieved

- **Coverage grows fast in the beginning, saturates near end.**
- **Is it described by**
  - $C(L) = 1 - e^{-aL}$?
  - No, doesn't fit.
- **It is controlled by distribution of detectability of faults.**
- **Detectability profile (Malaiya &Yang '84):**
- $\boldsymbol{H = \{h_1, h_2, .. h_N\}}$
  - N: total possible vectors
  - h_k: number of faults detected by exactly k vectors.



- **Total faults $\boldsymbol{M = \Sigma h_k}$**
- **h_1: number of least testable faults**

Ex: Circuit with higher h_1 would be harder to test.

Coverage with L random vectors

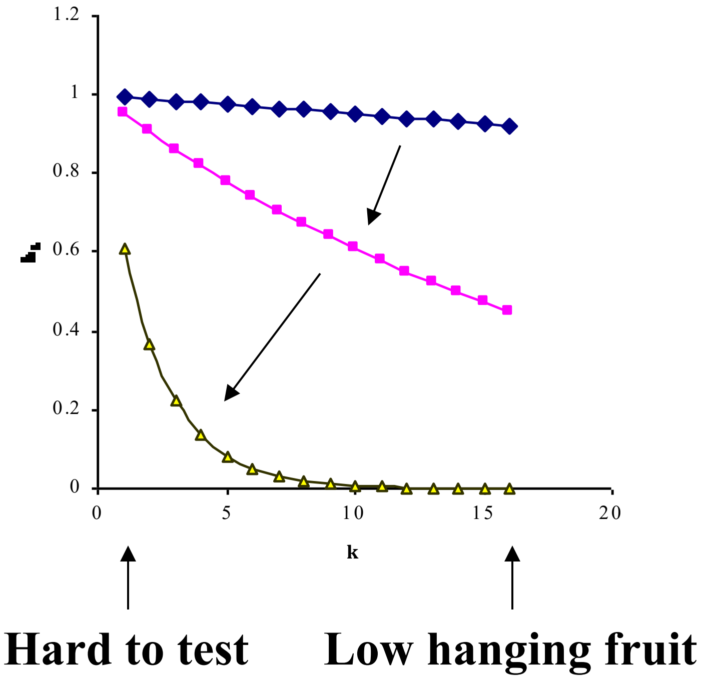
- h_k out of M defects detectable by exactly k vectors: detection probability k/N
- P{a defect with dp k/N not detected by a vector} = $(1 - \frac{k}{N})$
- P{a defect with dp k/N not detected by L vectors} = $(1 - \frac{k}{N})^L$
- Of h_k faults, expected number not covered is $(1 - \frac{k}{N})^L h_k$
- Expected test coverage with L vectors

$$\mathrm{C(L)} = 1 - \sum_{k=1}^{N} (1 - \frac{k}{N})^L \frac{h_k}{M}$$

Detectability Profile: Software

- Software detectability profile is exponential
- Justification: Early testing will find & remove easy-to-test faults.
- Testing methods need to focus on hard-to-find faults.

As testing time progresses, more of the faults are clustered to the left.



Implications: Fault Seeding

- A program has x defects. We want to estimate x.
- Seed j new faults.
- Do some testing. Let faults found be j_1 seeded faults and x_1 original faults.
- Assuming $j_1/j = x_1/x$ we get $x = x_1 \frac{j}{j_1}$

- However, in reality the x faults include harder faults to test,

$$\frac{j_1}{j} > \frac{x_1}{x} \quad hence\ x > \frac{x_1 j}{j_1}$$

Mean Time to Failure (MTTF)

- There is a very useful general relation between MTTF and R(t). Here T is time to failure, which is a random variable.

$$MTTF = E(T) = \int_0^\infty t\, f(t)dt$$

$$= -\int_0^\infty t \frac{dR(t)}{dt} dt$$

$$= [-t R(t)]_0^\infty + \int_0^\infty R(t)dt$$

$$Thus \; \boxed{MTTF = \int_0^\infty R(t)dt}$$

Worth Remembering!

$Note:$

$$R(t) = 1 - P\{failure\ in\ (0,t)\}$$
$$= 1 - P\{0 \le T \le t\}$$
$$= 1 - F(t)$$
$$\therefore \frac{dF(t)}{dt} = -\frac{dR(t)}{dt}$$
$$or\ \ f(t) = -\frac{dR(t)}{dt}$$

$Note:$

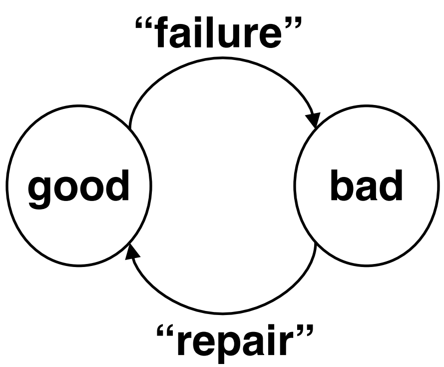
$xe^{-x} \to 0\ as\ x \to \infty$

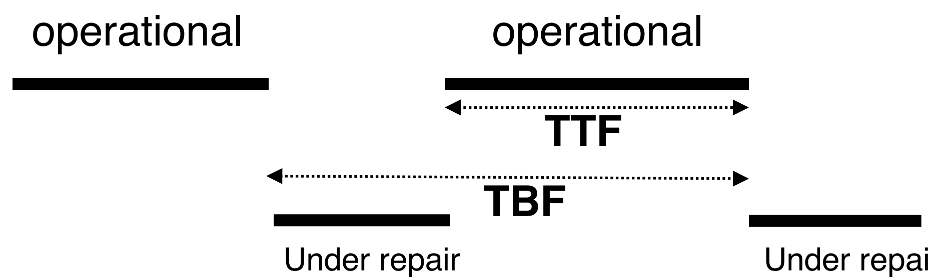
and R(t) is generally of the form e^{-at}

Thus $tR(t) \to 0\ as\ t \to \infty$.

Failures with Repair

- **Time between failures: time to repair + time to next failure**

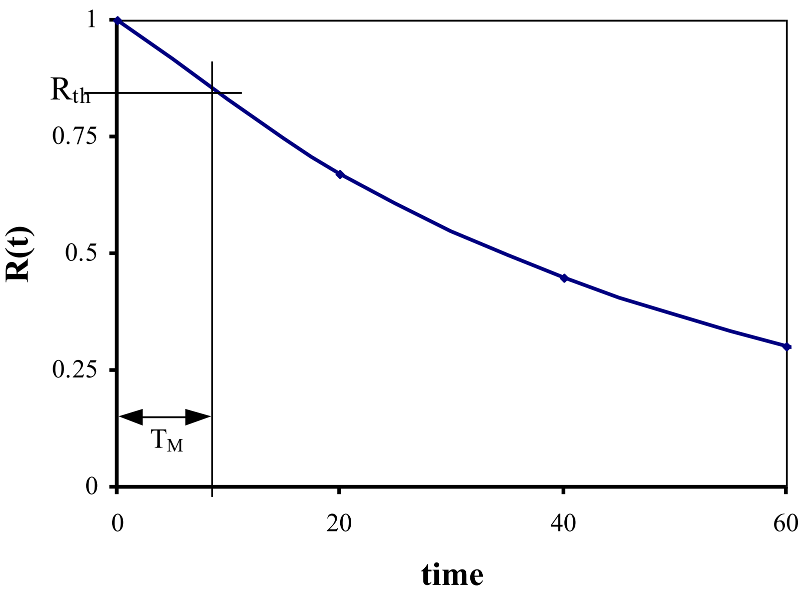




- **MTBF = MTTF + MTTR**
  - MTBF, MTTF are same same when MTTR ≈0
- **Steady state availability = MTTF / (MTTF+MTTR)**

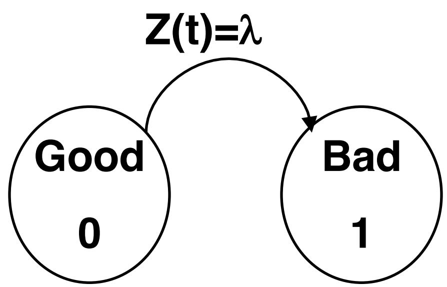
Mission Time (High-Reliability Systems)

- Reliability throughout the missic must remain above a threshold reliability R_{th}.
- Mission time T_M: defined as the duration in which $R(t) \geq R_{th}$.
- R_{th} may be chosen to be perhaps 0.95.
- Mission time is a strict measure, used only for very high reliability missions.



Basic Cases: Single Unit with Permanent Failure

- **Failure rate is the probability of failure/unit time**
- **Assumption: constant failure-rate λ**



The state transition diagram & the differential equation represent What we call Markov Modeling.

$$\frac{dp_0(t)}{dt} = -\lambda\ p_0(t) \text{ since the rate of leaving state 0 depends}$$

$$\text{on probability of being in state 0}$$

$$p_0(0) = 1 \qquad \text{initial condition}$$

Single Unit with Permanent Failure (2)

$$\frac{dp_0(t)}{dt} = -\lambda\ p_0(t)$$

$$p_0(0) = 1$$

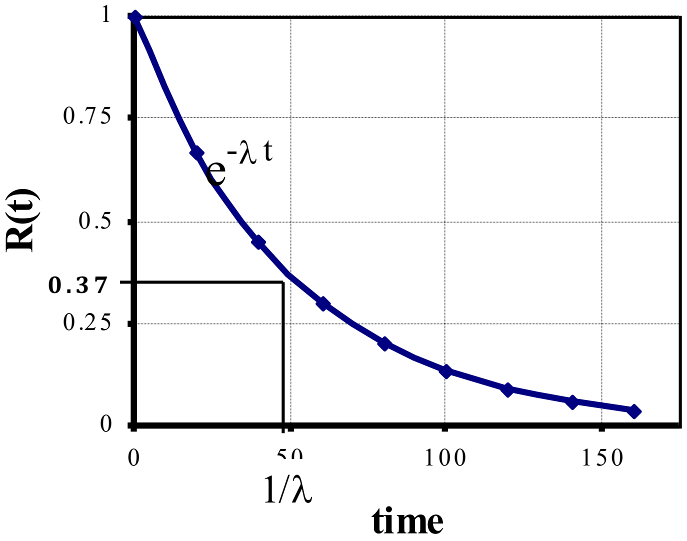
$$Solution: p_0(t) = e^{-\lambda\ t}$$

$$Since\ \ R(t) = p_0(t)$$

$$\boxed{R(t) = e^{-\lambda\ t}}$$

"The Exponential reliability law"

$$At \quad t = \frac{1}{\lambda},\ R(t) = e^{-1} = 0.368$$



Single Unit: Permanent Failure (3)

$$R(t) = e^{-\lambda t}$$

A(t) is same as R(t) in this case.

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt$$

$$= [-\frac{e^{-\lambda t}}{\lambda}]_0^\infty$$

$$= \frac{1}{\lambda}$$

- **Ex 1:** a unit has MTTF =30,000 hrs. Find failure rate.

  λ=1/30,000=3.3x10^{-5}/hr
- **Ex 2:** Compute mission time T_M if R_{th} =0.95.

  $e^{-\lambda T_M}$ =0.95 T_M= - ln(0.95)/ λ

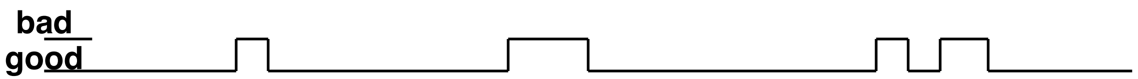
  ≈0.051/ λ
- **Ex 3:** Assume λ=3.33x10^{-5}, and R_{th} =0.95 find T_M.

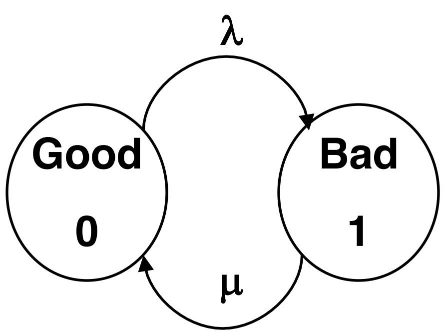
  Ans: T_M = 1538.8 hrs

  (compare with MTTF =30,000)

Single Unit: Temporary Failures(1)

- **Temporary: intermittent, transient, permanent with repair**





$$\frac{dp_0(t)}{dt} = -\lambda\ p_0(t) + \mu\ p_1(t)$$

$$\frac{dp_1(t)}{dt} = +\lambda\ p_0(t) - \mu\ p_1(t)$$

can be solved by laplace transform etc.

Note state diagram & Differential equations for Markov modeling

$$p_0(t) = p_0(0)e^{-(\lambda + \mu)t} + \frac{\mu}{\lambda + \mu}(1 - e^{-(\lambda + \mu)t})$$

Similarly we can get an expression for $p_1(t)$, however it is not needed since $p_1(t) = 1 - p_0(t)$.

Y. K. Malaiya, S. Y. H. Su: Reliability Measure of Hardware Redundancy Fault-Tolerant Digital Systems with Intermittent Faults. IEEE Trans. Computers 30(8): 600-604 (1981)

Single Unit: Temporary Failures(2)

- $p_0(t) = p_0(0)e^{-(\lambda+\mu)t} + \dfrac{\mu}{\lambda+\mu}(1 - e^{-(\lambda+\mu)t})$
- Availability $\text{A(t)} = p_0(t)$

Thus $A(t) = p_0(0)e^{-(\lambda+\mu)t} + \dfrac{\mu}{\lambda+\mu}(1 - e^{-(\lambda+\mu)t})$

- Note that steady – state probabilities exist :

$$\text{t} \to \infty, p_0(t) = \frac{\mu}{\lambda+\mu} \qquad p_1(t) = \frac{\lambda}{\lambda+\mu}$$

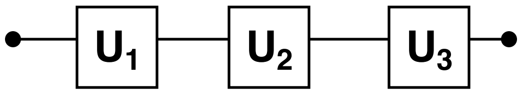
- Steady - state availability is $\dfrac{\mu}{\lambda+\mu}$

Series configuration

Series configuration: all units are essential. System fails if one of them fails .

- **Assumption:** statistically independent failures in units**.**

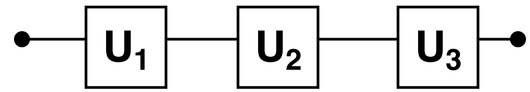
$$
\begin{aligned}
R_S &= P\{U_1\, good \bigcap U_2\, good \bigcap U_3\, good\} \\
&= P\{U_1\, g\}P\{U_2\, g\}P\{U_3\, g\} \\
&= R_1 R_2 R_3
\end{aligned}
$$



In general $R_S = \prod_{i=1}^{n} R_i$

Series configuration

The reliability block diagrams like this are only conceptual, not physical.



$$\text{If } R_i(t) = e^{-\lambda_i t}$$

$$\text{then } R_s(t) = \Pi e^{-\lambda_i t} = e^{-[\lambda_1 + \lambda_2 + \cdots + \lambda_n]t}$$

i.e. system failure rate is the sum of individual failure rates:

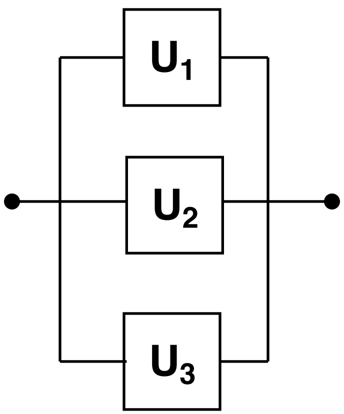
$$\lambda_S = \lambda_1 + \lambda_2 + \cdots + \lambda_n$$

This gives us a nice way to estimate the overall failure rate, when all the individual units are essential. This is the basis of the approach used in the popular "Military Handbook" MIL-HDBK-217 approach for estimating the failure rates for different systems.

The failure rates of individual units are estimated using empirical formulas. For example the failure rate of a VLSI chip is related to its complexity etc.

Combinatorial: Parallel

- **Parallel configuration:** System is good when least one of the several replicated units is good. A parallel configuration represents an *ideal* redundant system, ignoring any overhead.



$$
\begin{aligned}
R_s &= 1 - P\{all\ units\ \ bad\} \\
&= 1 - P\{U_1\ bad \bigcap U_2\ bad \bigcap U_3\ bad\} \\
&= 1 - P\{U_1\ b.\}P\{U_2\ b.\}P\{U_3\ b.\} \\
&= 1 - (1 - R_1)(1 - R_2)(1 - R_3)
\end{aligned}
$$

$$\text{In general } R_s = 1 - \prod_{i=1}^{n} (1 - R_i)$$

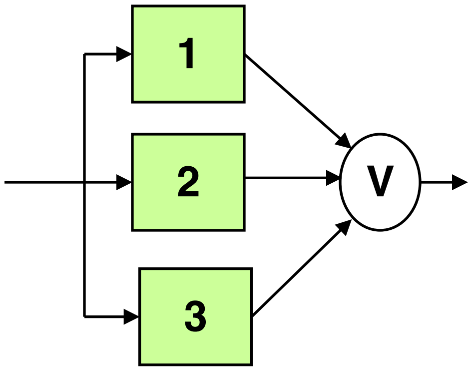
$$i.e. \quad \overline{R}_s = \prod_{i=1}^{n} \overline{R}_i$$

Where $\overline{\text{R}}$ represents 1-R, i.e. "*unreliability*"

Combinatorial: Parallel

Triple Modular Redundancy

- **Popular high-reliability scheme: 2-out-of-3 system**
- **Output is obtained using a majority voter**



$$R_{TMR} = \sum_{i=2}^{3} \binom{3}{i} R^i (1-R)^{3-i}$$

$$= 3R^2(1-R) + R^3$$

$$= 3R^2 - 2R^3$$

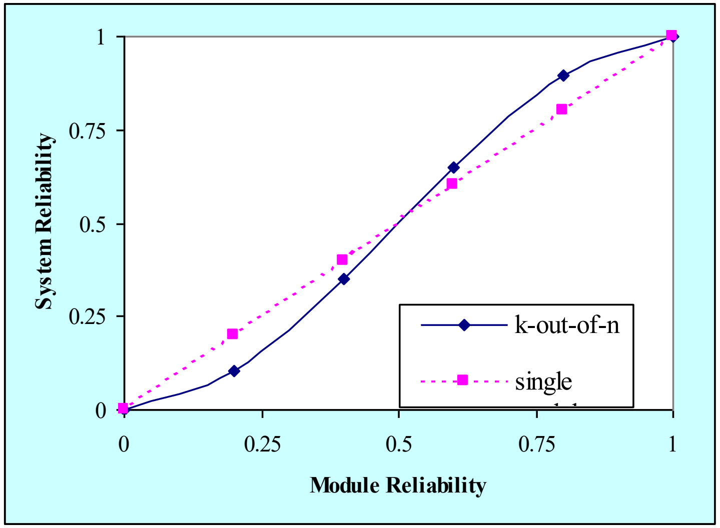
Where R is the reliability of a single module. This assumes that the voter is perfect, a reasonable assumption if the voter complexity is much less than an individual module.

Triple Modular Redundancy

Here is a plot of the system reliability, when the individual module reliability varies between 0 to 1.

Note that if the reliability of an individual module is less than 0.5, it is more likely to be bad. Having several such modules, and taking majority vote, will actually make the system less reliable, as you see in the figure.

A political application of the principle: majority-based democracy works only if the individuals are more likely to make the right decision than wrong!



TMR: Permanent Failures

$$\text{Let } R = e^{-\lambda t}$$

$$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

$$MTTF = \int_0^\infty R_{TMR}(t)dt$$

$$= \int_0^\infty (3e^{-2\lambda t} - 2e^{-3\lambda t})dt$$

$$= \frac{5}{6\lambda} \quad (\text{single module MTTF} : \frac{1}{\lambda})$$

- When is a single module as reliable as TMR?

  Solving $3R^2 - 2R^3 = R$

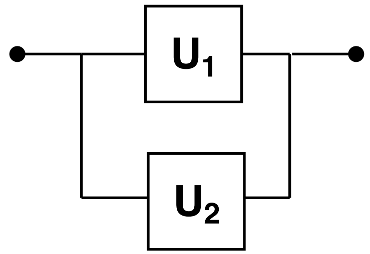
  we get $R_{cross} = 0.5$.

  TMR worse after $R < 0.5$!

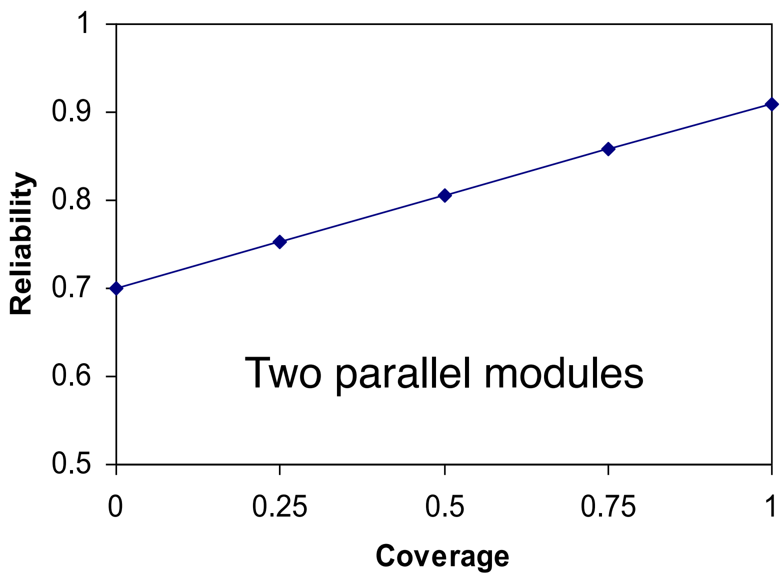
MTTF may not be a good measure when very high reliability levels are maintained.

Thus TMR has a lower MTTF than a single module!

Imperfect Coverage: Example



- $R_s = R_1 + R_2 C(1 - R_1)$
- Assuming $R_1 = R_2 = 0.7$
- In general $R_s = R_m \sum_{i=0}^{n-1} C^i (1 - R_m)^i$



Note that better coverage improves the reliability. When coverage =1, full potential of the parallel configuration is achieved.

TMR+Spares

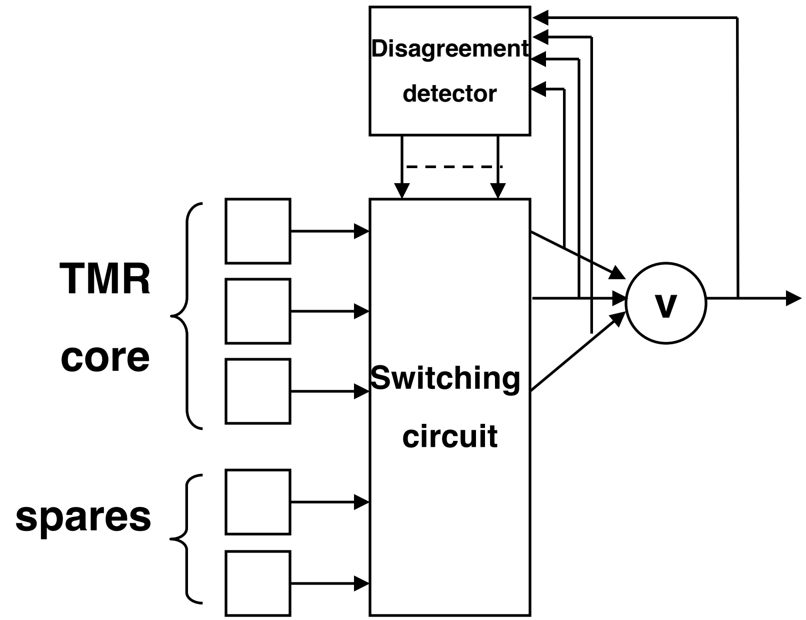
- TMR core, n-3 spares (assume same failure rate)
- **Scheme A**: System failure when all but one modules have failed. If we start with 3 in the core and 2 spares, the sequence will be

  3+2 → 3+1 → 3+0 → 2+0 → failure
- Reliability of the system then is

  $R_s=R_{sw}[1-nR(1-R)^{n-1}-(1-R)^n]$

  Where R is reliability of a single module and R_{sw} is the reliability of the switching circuit overhead.
- R_{sw} should depend on total number of modules n, and relative complexity of the switching logic.
- Let us assume that $R_{sw}=(R^a)^n$,

  where a is measure of relative complexity, generally a <<1. Then
- $\boxed{R_s=R^{an}\,[1-nR(1-R)^{n-1}-(1-R)^n]}$



Why is Defect Density Important?

- **Important measurement of reliability**
- **Often used as release criteria.**
- **Typical values of defect density /1000 LOC mentioned in literature:**

Beginning Of Unit Testing	On Release		
	Frequently Cited in literature	Highly Tested programs	NASA Space Shuttle Software
16	2.0	0.33	0.1

- **Long term trend: tolerable defect density limits have been gradually dropping, i.e. reliability expectations have risen.**

Note: NASA space shuttle controversy: see appendix.

A Static Defect Density Model

- **Li, Malaiya, Denton (93, 97)**

$$\boxed{D = C.F_{ph}.F_{pt}.F_{m}.F_{s}.F_{rv}}$$

- ***C is a constant of proportionality, based on prior data, used for calibration.***
- ***Default value of each function F_i (submodel) is 1.***
- ***Each function F_i is a function of some measure of the attribute.***

Possible factors

Phase

Programming Team

Process Maturity

Structure

Requirement Volatility

Reuse factor: A simple analysis

- **u: fraction of software reused**
- **d_r, d_n: defect density of reused software, defect density of new software, $d_r < d_n$**
- **Total defects = $[u.\ d_r + (1-u).\ d_n]S$**

  Where S is software size
- **If there was no reuse, defects would be $d_n S$**
- **Normalizing,**
  - **Reuse factor $F_r(u, d_r/d_n) = [u.\ d_r / d_n + (1-u)]$**
  - **F_r is 1 if there is no reuse, <1 if reuse.**

Static Model: Example

$$D = C.F_{ph}.F_{pt}.F_m.F_s.F_{rv}$$

•For an organization, C is between 12 and 16. The team has average skills and SEI maturity level is II. About 20% of code in assembly. Other factors are average (or *same as past projects*).

Estimate defect density at beginning of subsystem test phase.

•Upper estimate=16×2.5×1×1×(1+0.4 ×0.20)×1=43.2/KSLOC

•Lower estimate= 12×2.5×1×1×(1+0.4×0.20)×1=32.4/KLOC

Here the structure factor is 1+0.4×0.20 because of some assembly code. Factor 2.5 is for the beginning of the subsystem phase.

Exponential SRGM Derivation Pt 1

- Notation
  - T_s: average single execution time
  - k_s: expected fraction of faults found during T_s
  - T_L: time to execute each program instruction once

$$-\frac{dN(t)}{dt}T_s = k_s N(t)$$

Key assumption

$$-\frac{dN(t)}{dt} = \frac{K}{T_L}N(t) = \beta_1 N(t)$$

Notation: Here we replace K_s and T_s by more convenient K and T_L.

where $K = k_s \frac{T_L}{T_s}$ is fault exposure ratio

Exponential SRGM Derivation Pt 2

- **We get**

$$N(t) = N(0)\, e^{-\beta_1 t}$$

$$\mu(t) = \beta_o (1 - e^{-\beta_1 t})$$

$$\lambda(t) = \beta_o \beta_1 e^{-\beta_1 t}$$

The 2 equations contain the same information.

- For $t \to \infty$, total β_o=N(0) faults would be eventually detected. A "*finite-faults-model*".
- Assumes no new defects are generated during debugging.
- Proposed by Jelinski-Muranda '71, Shooman '71, Goel-Okumoto '79 and Musa '75-' 80. also called Basic.

A Basic SRGM (cont.)

- **Note that parameter β_1 is given by:**

$$\beta_1 = \frac{K}{T_L} = \frac{K}{(S.Q.\frac{1}{r})}$$

- S: source instructions,
- Q: number of object instructions per source instruction typically between 2.5 to 6 (see page 7-13 of Software rteliability Handbook, sec 7)
- r: object instruction execution rate of the computer
- K: *fault-exposure ratio*, range 1×10^{-7} to 10×10^{-7}, (t is in CPU seconds). Assumed constant here*.
- Q, r and K should be relatively easy to estimate.

***Y. K. Malaiya, A. von Mayrhauser and P. K. Srimani, "An examination of fault exposure ratio," in IEEE Transactions on Software Engineering, vol. 19, no. 11, pp. 1087-1094, Nov 1993**

SRGM : "Logarithmic Poisson"

- Many SRGMs have been proposed.
- **Another model Logarithmic Poisson** model, by Musa-Okumoto, has been found to have a good predictive capability

$$\mu(t) = \beta_o \ln(1 + \beta_1 t) \qquad \lambda(t) = \frac{\beta_o \beta_1}{1 + \beta_1 t}$$

- Applicable as long as $\mu(t) \leq N(0)$. Practically always satisfied. Term *infinite-faults-model* misleading.
- Parameters β_o and β_1 don't have a simple interpretation. An interpretation has been given by Malaiya and Denton (What Do the Software Reliability Growth Model Parameters Represent?).

Example: SRGM with Test Data (cont.)