Fault Trees

- Fault Trees
  - dual of Reliability Block Diagram
  - logic failure diagram
  - think in terms of logic where
    - 0 = operating, 1 = failed

- AND Gate
  - all inputs must fail for the gate to fail

- OR Gate
  - any input failure causes the gate to fail

- k-of-n Gate
  - k or more input failures cause gate to fail

---

e.g. Triplex Bus Guardian

- Active mode
  - M1 and M2 and M3 fail =>
  - AND Gate

- Passive Mode
  - “cutoff” with any single unit failure =>
  - OR Gate
**e.g. Triplex Bus Guardian**

- Total Failure
  - caused by either active or passive mode

![Fault Tree Diagram]

**e.g. Triplex Bus Guardian**

- How can one use the fault tree effectively to isolate those parts of the system that need reliability considerations?
**e.g. Triplex Bus Guardian**

- Combined fault model

\[ Q(1000h) = 0.9851243 \cdot 10^{-6} \]

\[ Q(1000h) = 0.295545 \cdot 10^{-1} \]

**Examples**

- Simple Passive TMR (no diagnosis)
  - RBD = (2 of 3): 2 operable => System operable
  - F-Tree = (2 of 3): 2 failed => System failed

- Simple TMR with *Benign* failures
  - RBD = (1 of 3): 1 operable => System operable
  - F-Tree = (3 of 3): 3 failed => System failed

- Summary
  - Parallel => AND
  - Series => OR
  - K-of-N => (n-k+1 of n)
SHARPE

- SYMBOLIC HIERARCHICAL AUTOMATED RELIABILITY AND PERFORMANCE EVALUATOR
- SHARPE provides a specification language and analysis algorithms for the following model types:
  - reliability block diagrams
  - fault trees
  - reliability graphs
  - series-parallel acyclic directed graphs
  - product-form queuing networks
  - Markov and semi-Markov chains
  - generalized stochastic Petri nets
- note: we are using the latest version of SHARPE and there could be changes that are not reflected in this handout.

SHARPE

- Read the “SHARPE LANGUAGE DESCRIPTION” for details. (The material below is directly adopted from there).
- Basic language components
  - **comments** ‘*’ indicates a comment line
  - **echo** *anytext* writes *anytext* to the output
  - **include** *filename* takes inputs from the file indicated
  - names: command **info constants** on the current version under solaris indicates:
    - size of input line: 512
    - maximum number of intervals for eval: 100
    - number of significant characters in words: 30
    - maximum number of phases in a phased-mission system (pms): 100
SHARPE

- **Basic language components**
  - *A word* is a sequence of characters (no white space, commas, semicolons, ( ) \ $).
  - *A subword* is a string $n$ or $(expression)$.
    - $n$ is a single letter
    - any expression can be used within the parantheses
  - *evaluated word* provides limited means of index
    - e.g. let $i=4$ and $j=5$, then the evaluated word $i$A-$(j-i)$B-$j$ evaluates to the component name: 4A-1B-5
  - length of words: as before, use **info constants**

---

**SHARPE**

- **Basic language components (cont.)**
  - Arithmetic Expressions
    - addition +
    - subtraction -
    - negation -
    - division /
    - multiplication *
    - exponentiation ^
    - “power of e” ^
Basic language components (cont.)

- Arithmetic Expressions
  » evaluated from left to right
  » examples:

\[
\begin{align*}
360 & \div (360 + (k-1) \times x) \\
k & \times (1 - c(x, k)) \\
3 & \times \lambda & \times \text{prob} \ (\text{second-fault, recovered; } 2*\lambda) \\
\neg((k-1) & \times \tau) \\
\delta & \div (((k-1) \times x) \times (1 - \neg((k-1) \times x / \delta)) \\
\sum (i,1,n,sum(j,1,i,j)) & \sum (i,1,n,\text{prob}(m,\$\{i\}-\$\{j\}))
\end{align*}
\]

- simple variables (simple_var), a name that must be bound to a value

\text{bind} \quad \text{simple_var expression}

\text{bind}
\text{<simple_var expression>}
\text{end}

- function

\text{func} \quad \text{function_name (param_list) expression}

- defined variable (is the same as a function without arguments)

\text{var} \quad \text{defined_var expression}
**SHARPE**

- Basic language components (cont.)
  - Scope of names and words
    » simple/defined variables, function names, model names, exp. polynomial names are global
    » component names are local to the model in which they appear
    » parameter names are local to their system, function etc.
    » index of a loop is local to the loop
    » index of a sum function is local to the function
  - parameter list (*param_list*)
    \[ \text{name, name, ..., name} \]
  - argument list (*arg_list*)
    \[ \text{expression, expression, ..., expression} \]

\[ cdf \quad F(t) \]

- \( F(t) \), is the cumulative distribution function for the failure probability of the component.
- \( F(t) \) is the probability that component \( i \) has failed by time \( t \). SHARPE can compute the system failure distribution function and the mean and variance of the function. It can be asked to evaluate the function at particular values of \( t \).

\[ F(t) = \sum_{j=1}^{k} a_j t^{k_j} e^{b_j t} \]

- real or complex
- non negative integer
- real or complex
**SHARPE Command Language**

- Model Definition
- Binding Values to Variables
- Calling Functions
- Printing (with built-in functions)

- silly details
  - `*` comment
  - `\` line continuation
  - names are truncated to 14 characters

**Reliability Block Diagram (RBD)**

- block
- comp
- parallel
- series
- kofn

SHARPE manual page 22
Fault Trees

- ftree
- basic
- repeat
- transfer
- and
- or
- kofn

SHAPRE manual page 23

Calling Functions + Printing

- See Section B.5 of the SHARPE manual for details

- **cdf** (system_name {,state_eword} {,arg_list})
  - all models except irreducible chain or gspn.
  - *state_eword* used only in markov chains
  - *arg_list* matches # of parameters in system definition
  - prints cdf, mean, variance

- **expr** expression {,expression...}
  - sharpe prints the value of the *expressions*
  - e.g.
    - expr(lambda)
    - expr(lam_a + lam_b)
Calling Functions + Printing

- **eval** `{system_name} {;state_eword} {;arg_list} lhi {function}
  - same as for cdf
  - *lhi* = low, high, increment are all expressions
  - *function* is cdf (default) or any other function
  - e.g. generate F(t) for t specified and print result
  - for Markov chain
    » argument {;state} allows to evaluate the probability of being in the state specified.
  - *lhi* low, high, increment e.g.
    » 100 200 25
    » evaluate F(t) at t = 100, 125, 150, 175 and 200 hours

Calling Functions + Printing

- * comment
- **echo** *string*
  - echoes *string* to the output
  - e.g. echo this is just a test
- **var** *var_name* *expression*
  - defines variable *var_name*
  - assigns the value in *expression*
  - does not print
Calling Functions + Printing

- **value**(t;system_name {,state_eword} {;arg_list})
  - calculates value of cdf at time \( t \)
  - does not print
  - use on right hand side of **var**
  - e.g.
    » var Fat1000 value(1000;bus_gd_pass)
    » expr Fat1000

- **format** constant
  - number of digits after decimal point to be printed

Values of “epsilon”
- user-controlled epsilon: small values that determine when algorithms have converged or two floating point numbers are equal

**epsilon** epsilon_id expression

- **basic** determines when two floats are even
- **result** determines when a printed result is considered to be zero

- e.g.
  » epsilon basic 1.0*10^-11
  » epsilon results 1.0*10^-10
**Bus Guardian (Passive)**

* SYSTEM: TRIPLEX BUS GUARDIAN -- PASSIVE FAILURE MODE  
* MODEL: RELIABILITY BLOCK DIAGRAM  
* -- Model Definition: block name, components, connectivity --  
*  
block bus_gd_pas  
comp z exp(lampas)  
series z3 z z z  
end  

* -- Calculate CDF for System Failure --  
*  
cdf(bus_gd_pas)  

* -- Bind Values to Variable Names --  
*  
bind  
lampas 1.0*10^-5  
end  

* -- Evaluate CDF at Specified Points --  
*  
eval(bus_gd_pas) 1 5 2  
eval(bus_gd_pas) 10 50 20  
eval(bus_gd_pas) 100 500 200  
end

---

**Bus Guardian (Passive)**

CDF for system bus_gd_pas:

\[
\begin{align*}
1.0000e+00 \ t(0) \ \exp(0.0000e+00 \ t) \\
+ \ -1.0000e+00 \ t(0) \ \exp(-3.0000e-05 \ t)
\end{align*}
\]

mean: 3.3333e+04  
variance: 1.1111e+09  

-------------------------------

**Bus Guardian (Passive)**

<table>
<thead>
<tr>
<th>t</th>
<th>F(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000 e+00</td>
<td>3.0000 e-05</td>
</tr>
<tr>
<td>3.0000 e+00</td>
<td>8.9996 e-05</td>
</tr>
<tr>
<td>5.0000 e+00</td>
<td>1.4999 e-04</td>
</tr>
</tbody>
</table>

---

**Bus Guardian (Passive)**

<table>
<thead>
<tr>
<th>t</th>
<th>F(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000 e+01</td>
<td>2.9996 e-04</td>
</tr>
<tr>
<td>3.0000 e+01</td>
<td>8.9960 e-04</td>
</tr>
<tr>
<td>5.0000 e+01</td>
<td>1.4989 e-03</td>
</tr>
</tbody>
</table>

-------------------------------

**Bus Guardian (Passive)**

<table>
<thead>
<tr>
<th>t</th>
<th>F(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000 e+02</td>
<td>2.9955 e-03</td>
</tr>
<tr>
<td>3.0000 e+02</td>
<td>8.9596 e-03</td>
</tr>
<tr>
<td>5.0000 e+02</td>
<td>1.4888 e-02</td>
</tr>
</tbody>
</table>
**Bus Guardian (Active)**

* SYSTEM: TRIPLEX BUS GUARDIAN -- ACTIVE FAILURE MODE  
* MODEL: RELIABILITY BLOCK DIAGRAM  
* -- Model Definition: block name, components, connectivity --  
*  
```plaintext
block bus_gd_act  
comp z exp(lamact)  
parallel z z z z  
end  
```

* -- Calculate CDF for System Failure  
*  
```plaintext
cdf(bus_gd_act)  
```

* -- Evaluate CDF at Specified Points  
*  
```plaintext
eval(bus_gd_act) 9 11 1  
eval(bus_gd_act) 90 110 10  
eval(bus_gd_act) 900 1100 100  
end  
```

---

**CDF for system bus_gd_act:**

\[
\begin{align*}
1.0000e+00 t(0) \exp(0.0000e+00 t) \\
+ -3.0000e+00 t(0) \exp(-1.0000e-05 t) \\
+ 3.0000e+00 t(0) \exp(-2.0000e-05 t) \\
+ -1.0000e+00 t(0) \exp(-3.0000e-05 t)
\end{align*}
\]

mean: 1.8333e+05  
variance: 1.3611e+10

---

<table>
<thead>
<tr>
<th>t</th>
<th>F(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.0000e+00</td>
<td>0.0000 e+00</td>
</tr>
<tr>
<td>1.0000 e+01</td>
<td>0.0000 e+00</td>
</tr>
<tr>
<td>1.1000 e+01</td>
<td>0.0000 e+00</td>
</tr>
</tbody>
</table>

---

**Bus Guardian (Active)**

**system bus_gd_act**

```
<table>
<thead>
<tr>
<th>t</th>
<th>F(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.0000 e+02</td>
<td>7.1923 e-07</td>
</tr>
<tr>
<td>1.0000 e+03</td>
<td>9.8512 e-07</td>
</tr>
<tr>
<td>1.1000 e+03</td>
<td>1.3092 e-06</td>
</tr>
</tbody>
</table>
```