Embedded Systems 25


Failure modes of subsystems REVIEW

- Fail-silent failures
  - subsystem either produces correct results
    or produces (recognizable) incorrect results
    or remains quiet
  - **can be masked as long as at least one system survives**
- Consistent failures
  - If subsystem produces incorrect results all recipients receive same (incorrect) result
  - **can be masked iff the failing systems form a minority**
- Byzantine failures
  - subsystem reports different results to different dependent systems
  - **can be masked iff strictly less than a third of the systems fail**

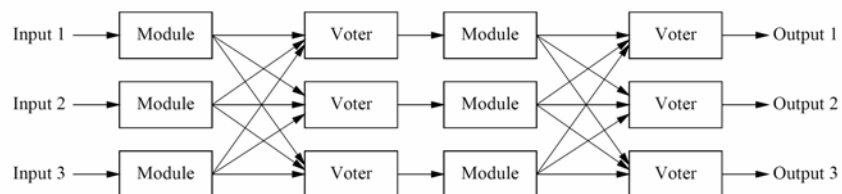## Dynamic hardware redundancy: standby spare arrangement
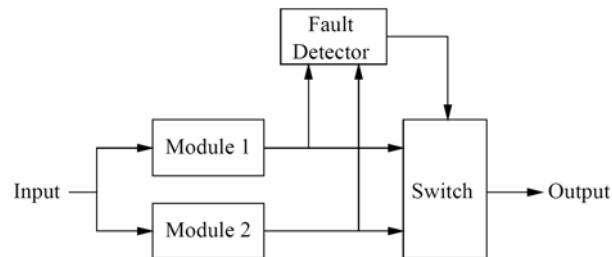
- Fault detection based on outputs (consistency check) not on voting
- Advantage: less redundant hardware
- Disadvantage: fault detection may take time $\Rightarrow$ fault not masked

BF - ES

- 3 -

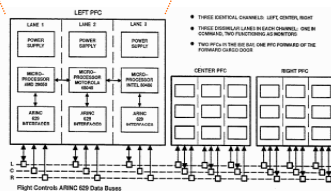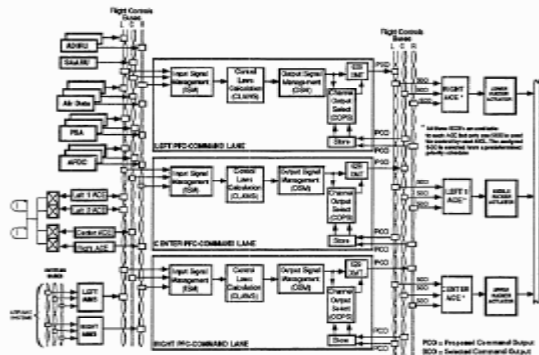## Static hardware redundancy: Multiple Stage TMR

BF - ES
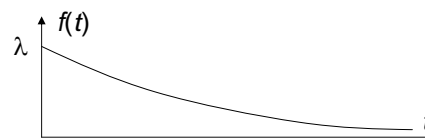
- 4 -

Boeing 777

Reliability: f(t), F(t)   REVIEW

- Let T: time until first failure, T is a random variable
- Let $f(t)$ be the density function of T

Example: Exponential distribution

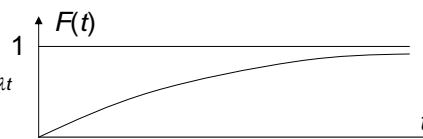$f(t) = \lambda e^{-\lambda t}$

- $F(t)$ = probability of the system being faulty at time t:

$$F(t) = \Pr(T \le t) \qquad F(t) = \int_0^t f(x)\,dx$$

Example: Exponential distribution

$$F(t) = \int_0^t \lambda e^{-\lambda x} dx = -[e^{-\lambda x}]_0^t = 1 - e^{-\lambda t}$$

Reliability: $R(t)$ REVIEW

Reliability $R(t)$ = probability that the time until the first failure is larger than some time t:
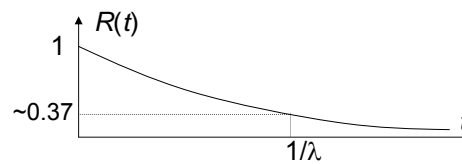
$$R(t)=\Pr(T>t),\ t\geq0 \qquad R(t) = \int_{t}^{\infty} f(x)dx$$

$$F(t) + R(t) = \int_{0}^{t} f(x)dx + \int_{t}^{\infty} f(x)dx = 1$$

$$R(t) = 1 - F(t)$$

Example: Exponential distribution

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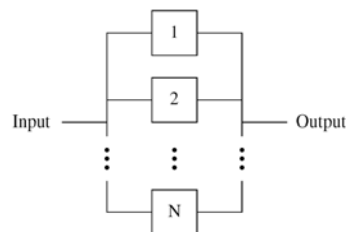

Reliability block analysis REVIEW

- **Goal:** compute reliability of a system from the reliability of its components
- **Serial composition**
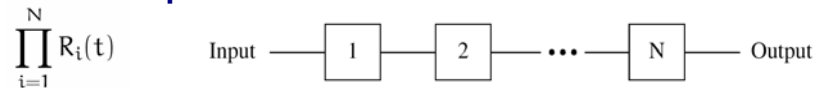
- **Parallel composition**
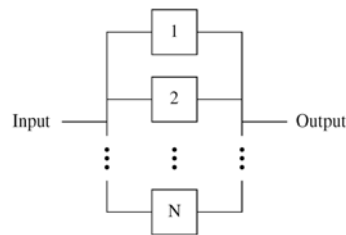
Inductive computation of reliability REVIEW

- **Assumption:** failures of the individual components are independent
- **Serial composition**

$$\prod_{i=1}^{N} R_i(t)$$

Input — 1 — 2 — $\cdots$ — N — Output

- **Parallel composition**

$$1 - \prod^{N} (1 - R_i(t))$$

Input — [1] [2] $\vdots$ [N] — Output

Approximation: Minimal Cuts REVIEW

- A **minimal cut** is a minimal set of components such that their simultaneous failure causes a system failure

- $$1 - \sum_{j \in MinimalCuts} \prod_{i \in j} [1 - R_i(t)]$$

  is a lower bound for the reliability R(t) of the full system.

- Minimal cuts with a single component are called *single point failures*.

Approximation: Minimal Tie Sets

- A **minimal tie set** is a minimal set of components such that their simultaneous functioning guarantees the functioning of the system
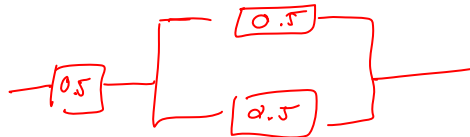
$$\sum_{j \in MinimalTies} \prod_{i \in j} R_i(t)$$

is an upper bound for the reliability R(t) of the full system.

Example
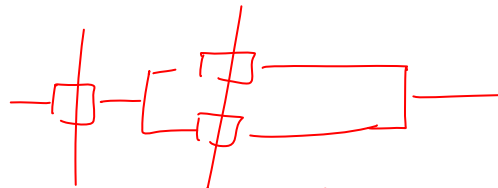

Minimal cuts:
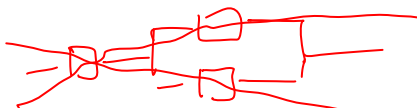
$$R \geq 1 - \left[(1 - 0.5) + (1 - 0.5)^2 \right] = 1 - [0.5 + 0.25]$$
$$= 0.25$$

Minimal ties:

$$R \leq 0.5 \cdot 0.5 + 0.5 \cdot 0.5 = 0.5$$

Exact: $0.5 \cdot \left(1 - (1 - 0.5)^2 \right) = 0.5 \cdot 0.75$
$$= 0.375$$

6

Fault tree Analysis (FTA)

- FTA is a top-down method of analyzing risks. Analysis starts with possible damage, tries to come up with possible scenarios that lead to that damage.

- FTA typically uses a graphical representation of possible damages, including symbols for AND- and OR-gates.

- OR-gates are used if a single event could result in a hazard.

- AND-gates are used when several events or conditions are required for that hazard to exist.

Example: Brake fluid warning lamp

Neil Storey:
Safety-critical computer systems
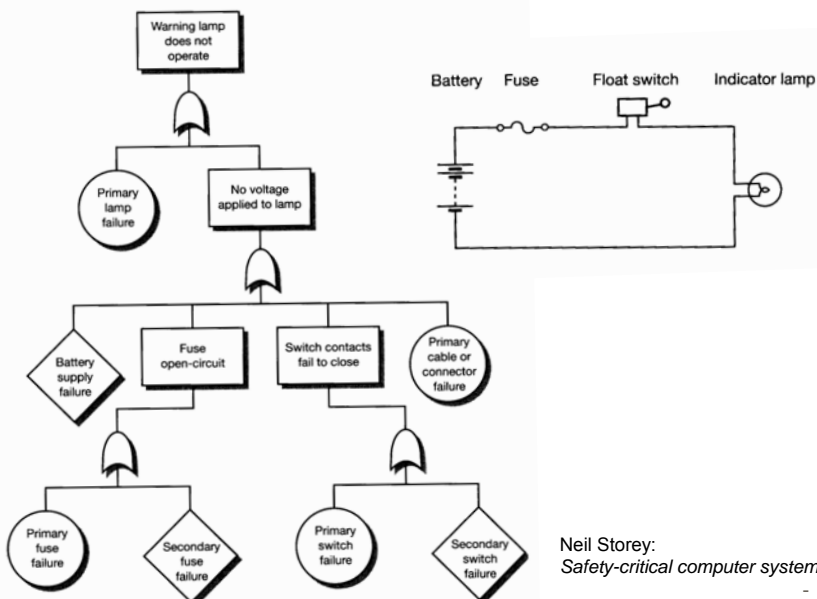
Direct Analysis

$$1 - \sum_{\vec{p} \in \{0,1\}^n} \left(FT(\vec{p}) \cdot \prod_{i=1}^{n} (1 - R_i(t))^{p_i} \cdot R_i(t)^{1-p_i} \right)$$

where

$\vec{p} = (p_1, ..., p_n)$ denotes the occurrence of the base events, and

$FT(\vec{p})$ denotes the value of the top event
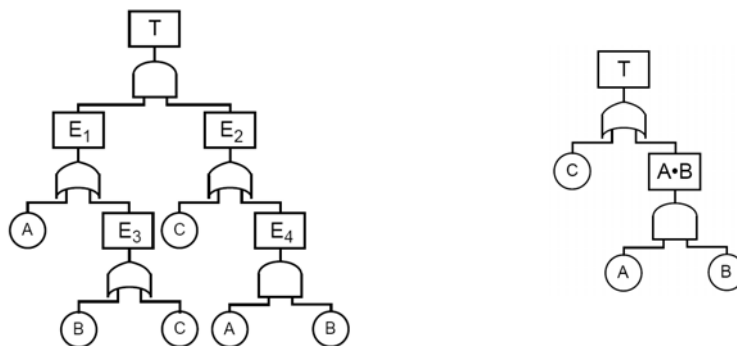
Problem: combinatorial explosion!

Equivalence

- Two fault trees are equivalent if the associated logical formulas are equivalent.
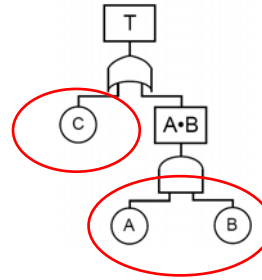- E.g., $(A \vee (B \vee C) \wedge (C \vee (A \wedge B))) \equiv (C \vee (A \wedge B))$

Minimal cut sets

Minimal cut set = "smallest set of basic events which, in conjunction, cause the top level event to occur".

Logically: Disjunctive Normal Form (DNF) = disjunction of conjunctions of basic events.

Example:
C   (single point of failure)   and
$A \wedge B$.

Mocus Algorithm (1972) „Method of Obtaining Cut Sets"

- Initialize the first element of a matrix with the top event operator
- As long as there is still an operator in the matrix:
  - If it is an AND operator, replace it with its inputs in the column
  - If it is an OR operator, replace it with its inputs in the row.
- Each column corresponds to a cut set; reduce to obtain minimal cut sets.
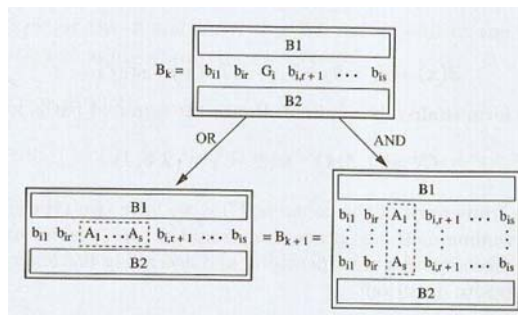
Nikolaos Limnios: *Fault Trees*

Example

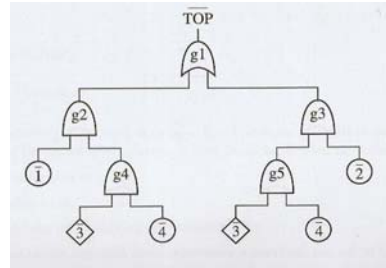$[g_1] \longrightarrow [g_2 \; g_3]$

$\longrightarrow \begin{bmatrix} 1 & g_3 \\ g_4 & g_3 \end{bmatrix}$

TOP
g1
g2   g3
1   g4   g5   2
3   4   3   4

$\longrightarrow \begin{bmatrix} 1 & 2 \\ 1 & g_5 \\ g_4 & g_3 \end{bmatrix}$

$\longrightarrow \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ g_4 & g_3 \end{bmatrix} \longrightarrow \cdots \longrightarrow \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 3 & 2 \\ 3 & 3 \\ 3 & 4 \\ 4 & 2 \\ 4 & 3 \\ 4 & 4 \end{bmatrix}$

$\Rightarrow$ cut sets: $\{1,3,4\}$, $\{2,3,4\}$.


Binary decision trees

- Let X be a set of boolean variables and $<$ a total order on X

- Binary decision tree (BDT) is a complete binary tree over $\langle X, < \rangle$
  - each leaf v is labeled with a boolean value $val(v) \in \mathbb{B}$
  - non-leaf v is labeled by a boolean variable $Var(v) \in X$
  - such that for each non-leaf v and vertex w:

$$w \in \{\, left(v), right(v)\,\} \;\Rightarrow\; (Var(v) < Var(w) \;\vee\; w \text{ is a leaf})$$

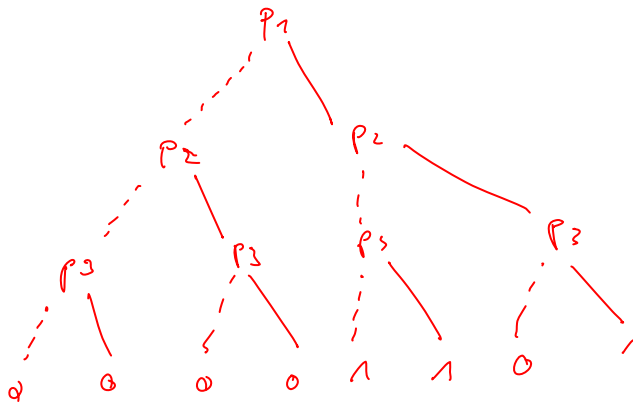$\Rightarrow$ On each path from root to leaf, variables occur in the same order

v ◯
left(v) ◯   ◯ right(v)
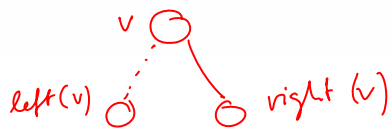
10

Example $BDT \simeq P_1 \wedge (\neg P_2 \vee P_3)$

Shannon Expansion

- Each boolean function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ can be written as:

$$f(x_1, \ldots, x_n) = (x_i \;\wedge\; f[x_i := 1]) \;\vee\; (\neg\, x_i \;\wedge\; f[x_i := 0])$$

  - where $f[x_i := 1]$ stands for $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$
  - and $f[x_i := 0]$ is a shorthand for $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$

- The boolean function $f_B(v)$ represented by vertex v in BDT B is:

  - for v a leaf: $f_B(v) = val(v)$
  - otherwise:

$$f_B(v) = (Var(v) \;\wedge\; f_B(right(v))) \;\vee\; (\neg\, Var(v) \;\wedge\; f_B(left(v)))$$

- $f_B = f_B(v)$ where v is the root of B

Considerations on BDTs

- BDTs are not compact
    - a BDT for boolean function $f : \mathbb{B}^n \to \mathbb{B}$ has 2^n leafs
  $\Rightarrow$ they are as space inefficient as truth tables!

$\Rightarrow$ BDTs contain quite some redundancy
    - all leafs with value one (zero) could be collapsed into a single leaf
    - a similar scheme could be adopted for isomorphic subtrees

- The size of a BDT does not change if the variable order changes

Ordered Binary Decision Diagrams

share equivalent expressions [Akers 76, Lee 59]

- Binary decision diagram (OBDD) is a directed graph over $\langle X, < \rangle$ with:
    - each leaf v is labeled with a boolean value $val(v) \in \{0, 1\}$
    - non-leaf v is labeled by a boolean variable $Var(v) \in X$
    - such that for each non-leaf v and vertex w:

$$w \in \{\, left(v), right(v)\,\} \;\; \Rightarrow \;\; (Var(v) < Var(w) \;\lor\; w \text{ is a leaf})$$

$\Rightarrow$ An OBDD is acyclic
    - f_B for OBDD B is obtained as for BDTs

Example

$$p_1 \wedge (\neg p_2 \vee p_3)$$

p_1 — p_2 — p_3

0 $\quad$ 1 $\quad$ 0 $\quad$ 1

p_1 — p_2 — p_3

0 $\quad$ 1
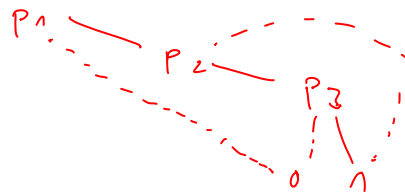
Reduced OBDDs

OBDD B over $\langle X, < \rangle$ is called *reduced* iff:

1. for each leaf v, w: $(val(v) = val(w)) \Rightarrow v = w$

   $\Rightarrow$ identical terminal vertices are forbidden

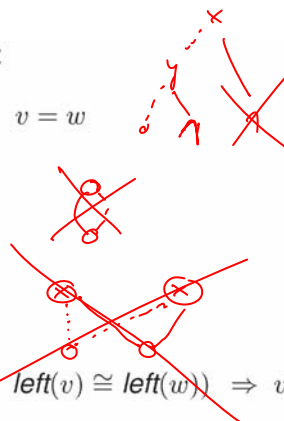2. for each non-leaf v: $left(v) \neq right(v)$

   $\Rightarrow$ non-leafs may not have identical children

3. for each non-leaf v, w:

$(Var(v) = Var(w) \wedge right(v) \cong right(w) \wedge left(v) \cong left(w)) \Rightarrow v = w$

   $\Rightarrow$ vertices may not have isomorphic sub-dags

   this is what is mostly called BDD; in fact it is an ROBDD!

Canonicity

For ROBDDs B and B′ over $\langle X, < \rangle$ we have:

$(f_B = f_{B'})$ implies B and B′ are isomorphic

$\Rightarrow$ *for a fixed variable ordering, any boolean function*
can be uniquely represented by an ROBDD (up to isomorphism)

MakeNode(var,v_1,v_2)

If H(var,v_1,v_2) $\neq$ empty then return H(var,v_1,v_2);   `lookup in hashtable`
If (v_1=v_2) then return v_1

res := new node(var,v_1,v_2);
H(var,v_1,v_2) := res;   `memorize result`
return res;

Computing AND and OR

- Shannon expansion for binary operations:

$$f \ op \ g \quad = \quad (x_1 \ \wedge \ (f[x_1 := 1] \ op \ g[x_1 := 1]))$$
$$\vee \ (\neg x_1 \ \wedge \ (f[x_1 := 0] \ op \ g[x_1 := 0]))$$

- A top-down evaluation scheme using the Shannon's expansion:
    - let v be the variable highest in the ordering occurring in B_f orB_g
    - split the problem into subproblems for $v := 0$ and $v := 1$, and solve recursively
    - at the leaves, apply the boolean operator op directly
    - reduce afterwards to turn the resulting OBDD into an ROBDD

- Efficiency gain is obtained by *dynamic programming*
    - the time complexity of constructing the ROBDD of $B_{f \ op \ g}$ is in $\mathcal{O}(|B_f| \cdot |B_g|)$

Apply(op,v_1,v_2)

```
if G(v₁, v₂) ≠ empty then return G(v₁, v₂) fi;
if (v₁ and v₂ are terminals) then res := val(v₁) op val(v₂) fi;
else if (v₁ is terminal and v₂ is nonterminal)
    then res := MakeNode(Var(v₂), APPLY(op, v₁, left(v₂)), APPLY(op, v₁, right(v₂)));
else if (v₁ is nonterminal and v₂ is terminal)
    then res := MakeNode(Var(v₁), APPLY(op, left(v₁), v₂), APPLY(op, right(v₁), v₂));
else if (Var(v₁) = Var(v₂))
    then res := MakeNode(Var(v₁), APPLY(op, left(v₁), left(v₂)), APPLY(op, right(v₁), right(v₂)));
else if (Var(v₁) < Var(v₂))
    then res := MakeNode(Var(v₁), APPLY(op, left(v₁), v₂), APPLY(op, right(v₁), v₂));
else                                                 (* Var(v₁) > Var(v₂) *)
    res := MakeNode(Var(v₂), APPLY(op, v₁, left(v₂)), APPLY(op, v₁, right(v₂)));
G(v₁, v₂) := res;
return res
```
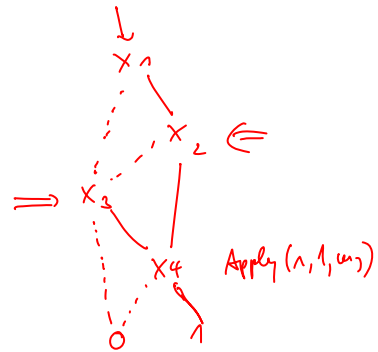
lookup in hashtable

memorize result
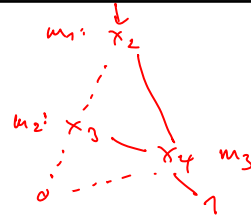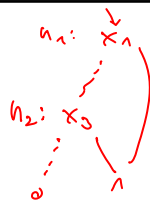
Example

ROBDDs of Fault Trees

- Each path through the BDD from the root to a leaf node represents a disjoint combination of component failures and non-failures
- A path with a leaf node labeled with a 1 leads to system failure
- Probabilities associated with arcs on each path are either (1-$R(t)$) (component failure probability) for the right branch or $R(t)$ for the left branch
- System unreliability is given by the sum of the probabilities for all paths from the root to a leaf node labeled 1

16

Example
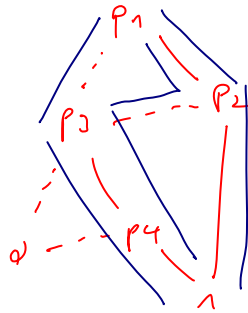
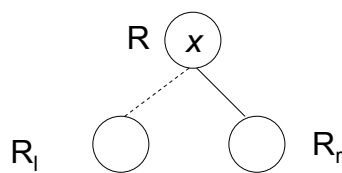$$\Rightarrow R(t) =$$
$$(1 - R_1(t)) \cdot (1 - R_2(t))$$
$$+ (1 - R_1(t)) \cdot R_2(t)$$
$$\qquad \cdot (1 - R_3(t)) \cdot (1 - R_4(t))$$
$$+ R_1(t)$$
$$\qquad \cdot (1 - R_3(t)) \cdot (1 - R_4(t))$$

Recursive BDD evaluation

R $\quad x$

R_l $\qquad$ R_r
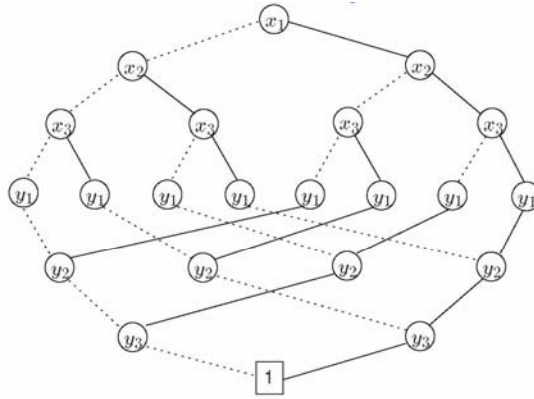
$$R(t) = R_x(t) * R_l(t) + (1 - R_x(t)) * R_r(t)$$

$$R_1(t) = 1$$
$$R_0(t) = 0$$

ROBDDs can be exponentially large

The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \ldots \wedge (x_n \leftrightarrow y_n)$

has $3 \cdot 2^n - 1$ vertices under ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$

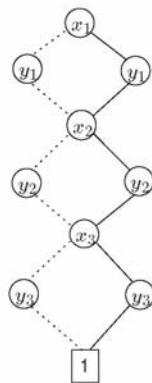## Alternative variable ordering

The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \ldots \wedge (x_n \leftrightarrow y_n)$

has $3 \cdot n + 2$ vertices under ordering $x_1 < y_1 < \ldots < x_n < y_n$

Optimal variable ordering

- The size of ROBDDs is dependent on the variable ordering

- Is it possible to determine $<$ such that the ROBDD has minimal size?
  - the optimal variable ordering problem for ROBDDs is NP-complete
  - polynomial reduction from the 3SAT problem   (Bollig & Wegener, 1996)

- There are many boolean functions with large ROBDDs
  - for almost all boolean functions the minimal size is in $\Omega(\frac{2^n}{n})$

- How to deal with this problem in practice?
  - guess a variable ordering in advance
  - rearrange the variable ordering during the manipulations of ROBDDs

---

Sifting Algorithm (1993)

Dynamic variable reordering using variable swapping

1. Select some variable x_i
2. By successive swapping determine position where the ROBDD has least size
3. Shift to its optimal position
4. Go back to 1 until no more improvement.

Often only yields local optimum, but works well in practice.

Limitations of combinatorial models

- Assumption that failure probability is independent of the system state is often wrong.

Example: cold-spare redundancy
- Failure during standby is unlikely
- Failure during activation is likely

$\Rightarrow$ state-based models are required