ECE454/544: Fault-Tolerant Computing & Reliability Engineering

Lecture #12–

Binary Decision Diagrams (BDD)

Instructor: Dr. Liudong Xing

Fall 2022

Administrative Issues (10/24, Mon.)

- Project meeting (in-person or virtual)
  - Due by **Oct. 28, Friday**
- Homework#5 assigned today
  - Due by **Oct. 31, Monday**
- Today’s topics
  - Finish Lecture#11 (RBD)
  - Then Lecture#12 (BDD)

Review of Lecture #11

- A RBD is a **success-oriented** network describing the function of the system
- In terms of modeling capability, RBD is equivalent to the static/traditional fault trees, and they can be converted into each other easily
- Path-sets and cut-sets can be generated from both RBD and fault trees
- I/E and SDP can be applied to the quantitative analysis based on both path sets and cut sets

Topics

- Binary decision diagrams (BDD)
  - Basic concepts
  - BDD manipulation & construction
  - Variable ordering in BDD
  - Calculating (un)-reliability from the BDD

Shannon Decomposition & *ite* Format

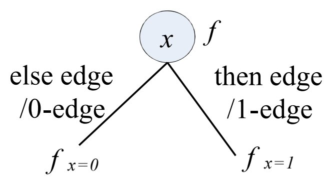
- Let f be a Boolean expression and x be a Boolean variable:

$$f = x \cdot f_{x=1} + \bar{x} \cdot f_{x=0}$$

  - $f_{x=1}$ represents the function f evaluated at $x=1$
  - $f_{x=0}$ represents the function f evaluated at $x=0$

- The if-then-else (***ite***) format of the Shannon decomposition

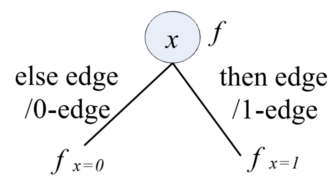
$$f = ite(x, F_1, F_2) \equiv x \cdot F_1 + \bar{x} \cdot F_2 \; ;$$
$$F_1 \equiv f_{x=1}, \quad F_2 \equiv f_{x=0} \; .$$



Binary Decision Diagrams (BDD)

- A BDD is a directed acyclic graph (DAG) based on Shannon decomposition

- A BDD is a binary tree
  - two sink nodes labeled with constants 1 and 0
  - each non-sink node is labeled with a Boolean variable x and has two outgoing edges called **1-edge (then-edge)** and **0-edge (else-edge),** respectively

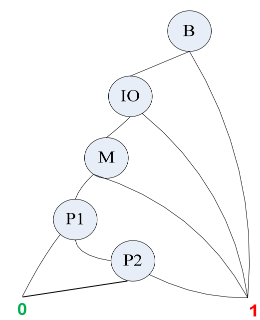


Other Forms

- An **ordered BDD (OBDD)** is a BDD with the constraint that the variables are ordered and every source-to-sink path in the OBDD visits the variables in an ascending order
- A **reduced OBDD (ROBDD)** is an OBDD where each node represents a distinct Boolean expression

ROBDD

- ROBDD are widely used in practice!
- In reliability engineering:
  - Sink node 1: the system fails
  - Sink node 0: the system is functioning
  - x=1: component x is failed
  - x=0: component x is functioning



BDD of computer system

Agenda

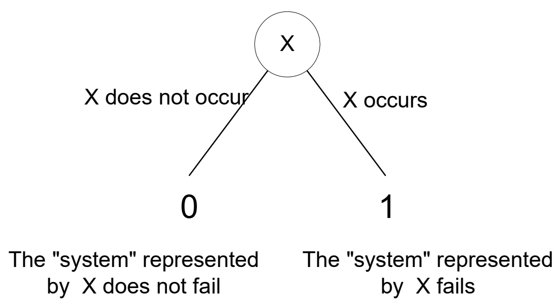
- Binary decision diagrams (BDD)
  - √ Basic concepts
  - **BDD manipulation & construction**
  - Variable ordering in BDD
  - Calculating (un)-reliability from the BDD

BDD Construction (1)

- To generate ROBDD, the ordering of the variables (basic events) must be selected
  - An index is assigned to each variable to indicate its position in the ordering
  - The order is not changed during the generation
  - Index(x) < index (y) implies that y is behind x in the order of variables

BDD Construction (2)

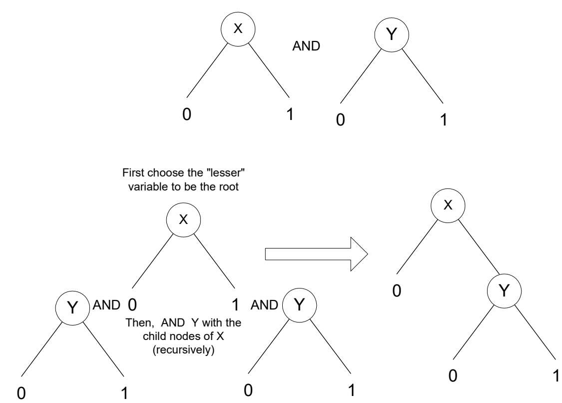
- BDD is constructed **from the bottom-up**
- The BDD of a **basic event** in fault tree (i.e., a system component):



ite(x, 1, 0)

BDD Construction (3)

- **AND**'ing two BDD: index (x) < index (y)



Assume: $g = ite(x,1,0)$, $h = ite(y,1,0)$

if $index(x) < index(y)$, then

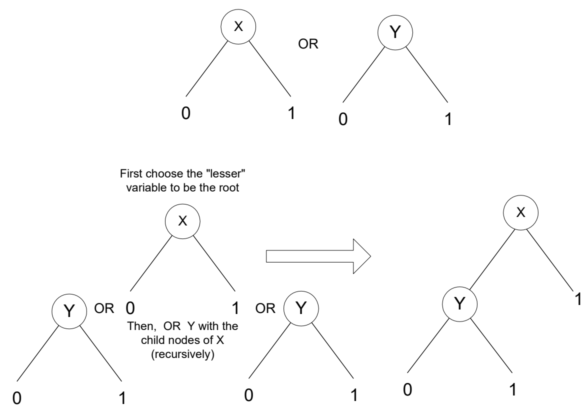
$g\ AND\ h = ite(x,1,0)\ AND\ ite(y,1,0)$

$= ite(x,\ 1\ AND\ h,\ 0\ AND\ h)$

$= ite(x,\ h,\ 0)$

BDD Construction (4)

- **OR**'ing two BDD: index (x) < index (y)



Assume: $g = ite(x,1,0)$, $h = ite(y,1,0)$

if $index(x) < index(y)$, then

g AND $h = ite(x,1,0)$ OR $ite(y,1,0)$

$= ite(x, 1$ OR $h, 0$ OR $h)$

$= ite(x, 1, h)$

Combination Operation on BDD

- Let Boolean expression g and h be:

$$g = ite(x, g_{x=1}, g_{x=0}) = ite(x, G_1, G_2)$$
$$h = ite(y, h_{y=1}, h_{y=0}) = ite(y, H_1, H_2)$$

- A logic operation (AND/OR, $\diamond$) between g and h can be represented by the following BDD manipulation:

$$ite(x, G_1, G_2) \diamond ite(y, H_1, H_2) = \begin{cases} ite(x, G_1 \diamond H_1, G_2 \diamond H_2) & index(x) = index(y) \\ ite(x, G_1 \diamond h, G_2 \diamond h) & index(x) < index(y) \\ ite(y, g \diamond H_1, g \diamond H_2) & index(x) > index(y) \end{cases}$$

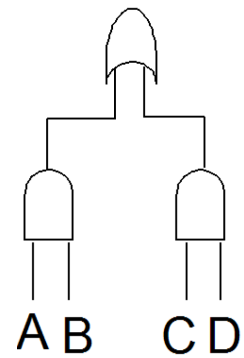
- The same rule can be used for logic operation between sub-expressions **until one of them becomes a constant 0 or 1**

Reduction Rules

- To achieve ROBDD, two reductions are performed as the BDD is built. These ensure that the BDD that results is minimal for the chosen ordering
  - Merge isomorphic subtrees (if two identical sub-BDD result, at least one is superfluous)
  - Delete useless nodes (A node with two equal children is useless and should be replaced with one of its children.)

Hands-On Problem (1)

- Generate the BDD for the following fault tree using variable ordering of A<B<C<D



FT-BDD1

Agenda

- Binary decision diagrams (BDD)
  - √ Basic concepts
  - √ BDD manipulation & construction
  - – **Variable ordering in BDD**
  - – Calculating (un)-reliability from the BDD

Variable Ordering in BDD

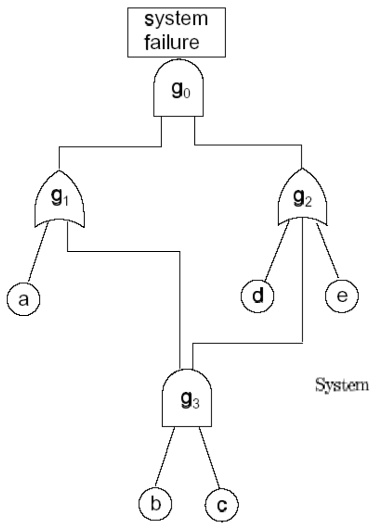
- The size of BDD depends heavily on the input variable ordering used to build the BDD

M. Bouissou, F. Bruyere, and A. Rauzy, "BDD based fault-tree processing: A comparison of variable ordering heuristics," *ESREL'97 conference*, June 1997.

Example

- Generate BDD using two different ordering for the following fault tree (FT-BDD2):

$a \prec b \prec c \prec d \prec e$ and $b \prec c \prec a \prec d \prec e$



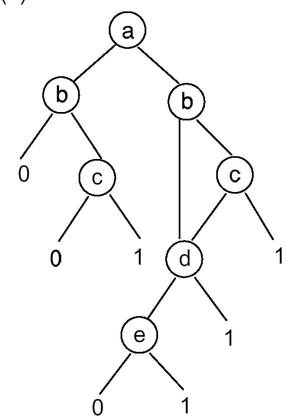
$$
\begin{aligned}
\text{System failure} &= g_0 = g_1 \wedge g_2 \\
g_1 &= a \vee g_3 \\
g_2 &= d \vee g_3 \vee e \\
g_3 &= b \wedge c
\end{aligned}
$$

$$f = (a \vee (b \wedge c)) \wedge (d \vee (b \wedge c) \vee e)$$

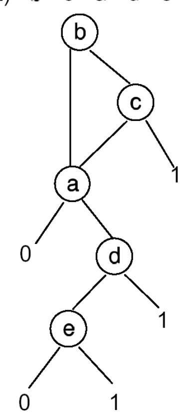
Example (Cont'd)

- BDD for the example fault tree:

(1) a->b->c->d->e



(2) b->c->a->d->e

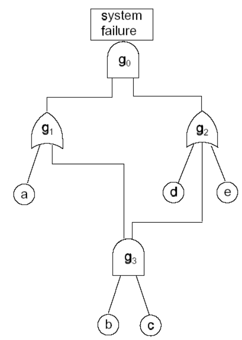


Note!

- The poor ordering can significantly affect the size of the BDD, thus the reliability analysis solution time for large systems!
- Currently there is no rule-based means of determining the best way of ordering basic events for a given fault tree structure
- Fortunately heuristics can usually be used to find a reasonable variable ordering

Example Heuristics

- **H1**: Top-down, left-right approach
  - The indexing results from a **depth-first left-most traversal** of the fault tree
  - **Example:** the order in which variables are visited for FT-BDD2 induces the ordering



$$a \prec b \prec c \prec d \prec e.$$

1	2	3	4	5	6	7	8	9	10
g_0	g_1	a	g_3	b	c	g_2	d	g_3	e

Example Heuristics (Cont'd)

- **H2**: works in 3 steps
  - Associate each terminal variable with weight 1 and propagate these weights bottom-up through fault tree by associating to each intermediate variable the sum of weights of variables occurring in its definition
  - Sort arguments of connectives in increasing order of their weights
  - Apply **H1** on the resulting fault tree

H2: Example

- **Example:** consider fault tree FT-BDD2
  - Associate weight 1 to *a,b,c,d,e*
  - Compute the weights of g0, g1, g2, g3

$$\begin{aligned}\omega(g_3) &= \omega(b)+\omega(c)=2\\ \omega(g_2) &= \omega(d)+\omega(g_3)+\omega(e)=4\\ \omega(g_1) &= \omega(a)+\omega(g_3)=3\\ \omega(g_0) &= \omega(g_1)+\omega(g_2)=7\end{aligned}$$

  - Rewrite g2 as $g_2 = d \vee e \vee g_3$
  - g0, g1, g3 keep unchanged!
  - Perform a depth-first, left-most traversal of the resorted fault tree

1	2	3	4	5	6	7	8	9	10
g_0	g_1	a	g_3	b	c	g_2	d	e	g_3

$$a \prec b \prec c \prec d \prec e$$

Example Heuristics (Cont'd)

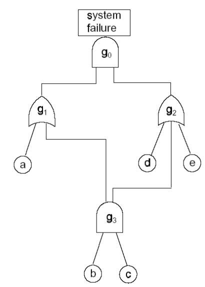
- H3: works in two steps
  - Sort the arguments of each connective according to their number of references or fanouts (decreasing order)
  - Apply **H1** on the resulting fault tree

H3: Example

- **Example:** consider FT-BDD2
  - g1, g2 have only 1 fanout → definition of g0 remain unchanged
  - g3 has 2 fanouts (g1, g2), a,d,and e have 1 fanout → g1, g2 are rewritten as

$$g_1 = g_3 \vee a \qquad g_2 = g_3 \vee d \vee e.$$

  - b, c have 1 fanout → definition of g3 remains unchanged!
  - Visiting order resulted from a depth-first, left-most traversal of rewritten fault tree



1	2	3	4	5	6	7	8	9	10
g_0	g_1	g_3	b	c	a	g_2	g_3	d	e

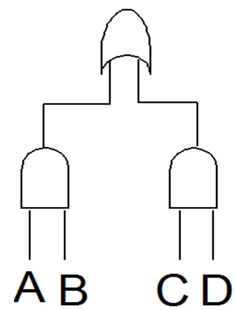
$$b \prec c \prec a \prec d \prec e.$$

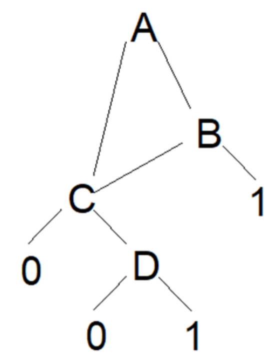
Agenda

- Binary decision diagrams (BDD)
  - √ Basic concepts
  - √ BDD manipulation & construction
  - √ Variable ordering in BDD
  - – **Calculating (un)-reliability from the BDD**

Hands-On Problem (1) Revisit

- Generate the BDD for the following fault tree using variable ordering of A<B<C<D





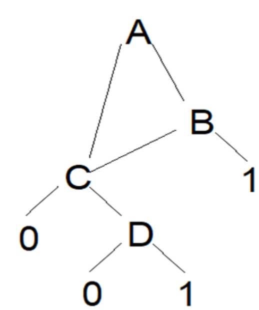
  - Sink/leaf node 1: system fails
  - Sink/leaf node 0: system is functioning
  - x=1: component x is failed (right-edge)
  - x=0: component x is functioning (left-edge)

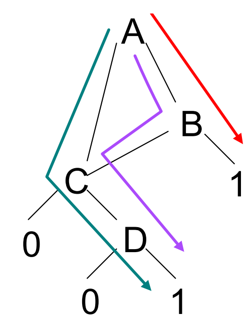
Calculating Unreliability using BDD

- 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 *q* (**component failure probability**) for the right branch or (1-*q*)(**component reliability**) 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**
- System **reliability** is given by the sum of the probabilities for all paths from the root to a leaf node labeled **0**

Example

- Consider FT-BDD1:





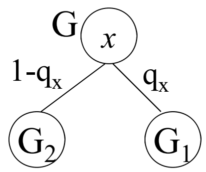
- Three paths from root (A) to leaf node labeled 1.
- Each path is disjoint by construction, so no subtraction is needed.
- System Unreliability=Pr{system failure}

= Pr{A and B}+
Pr{A and (not B) and C and D} +
Pr{ (not A) and C and D}
$= q_A q_B + q_A(1-q_B)q_C q_D + (1-q_A)q_C q_D$

Recursive BDD Evaluation Algorithm

- Consider a BDD branch

$G = ite(x, G_1, G_2)$



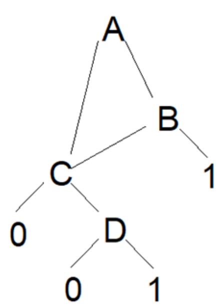
$$P\{G\} = q_x P\{G_1\} + (1-q_x) P\{G_2\}$$

– When x is the root node of the BDD, P{G} gives the system unreliability.

– Exit conditions:

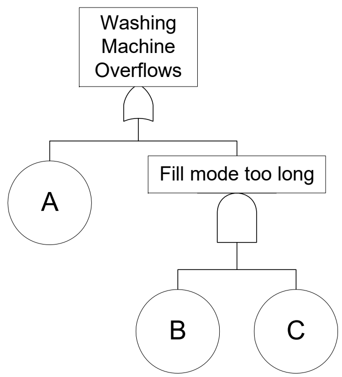
If G=0 then P{G}=0

If G=1 then P{G}=1



Hands-On Problem (2)

- Find the BDD model for the following fault tree and evaluate the system reliability given that the failure probabilities of components are: Pr{A} = 0.01, Pr{B} = 0.05, Pr{C} = 0.075 and that all failures are *s*-independent



Review (Lecture#9)

I-E Method

$$\Pr\{F\} = \Pr\{A + BC\}$$
$$= \Pr\{A\} + \Pr\{BC\} - \Pr\{ABC\}$$
$$= 0.01 + 0.00375 - 0.0000375$$
$$= 0.0137125$$

SDP Method

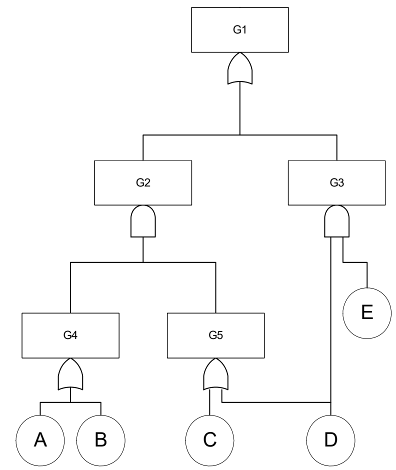
$$\Pr\{F\} = \Pr\{A + BC\}$$
$$= \Pr\{A\} + \Pr\{\overline{A}BC\}$$
$$= 0.01 + 0.99*0.05*0.075$$
$$= 0.0137125$$

Hands-On Problem (3)

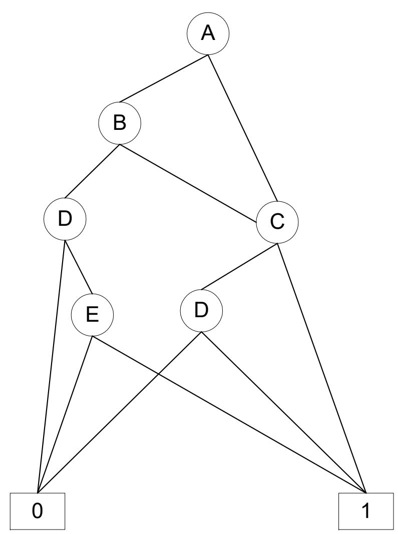
- For the fault tree called example FT2 in L#9, generate the BDD and calculate the probability of occurrence for the top event in the fault tree.
- Assume that the probability of occurrence for each of the basic events is:

  $\Pr\{A\} = 0.05$, $\Pr\{B\} = 0.10$, $\Pr\{C\} = 0.15$,

  $\Pr\{D\} = 0.20$, $\Pr\{E\} = 0.25$

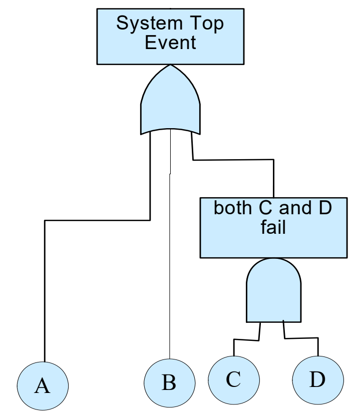


BDD for Example FT2



Hands-On Problem (4)

- For the following fault tree model, generate the binary decision diagrams using the ordering of D<B<C<A.
- Given each component fails with a fixed failure probability of 0.1, determine the reliability of the entire system.



References

- J. B. Dugan and S. A. Doyle. "New Results in Fault-Tree Analysis" Tutorial Notes of the Annual Reliability and Maintainability Symposium, January 1997
- R. E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. on Computers, vol. C-35, no. 8, pp. 677–691, August 1986.
- A. Rauzy, "New algorithms for fault tree analysis," Reliability Engineering and System Safety, vol. 40, pp. 203–211, 1993.
- M. Bouissou, F. Bruyere, and A. Rauzy. "BDD Based Fault-Tree Processing: A Comparison of Variable Ordering Heuristics", Proceedings of ESREL'97 conference, June 1997.
- L. Xing and S. V. Amari, Binary Decision Diagrams and Extensions for System Reliability Analysis, Wiley-Scrivener, MA, ISBN: 978-1-118-54937-7, July 2015

Summary of Lecture #12

- BDD can be used to efficiently and accurately solve the combinatorial reliability models without the use of cutsets
- The size of the BDD depends heavily on the input variable ordering

Next topic:

Sensitivity Analysis