STK3405 – Lecture 10
Part 1

A. B. Huseby

Department of Mathematics
University of Oslo, Norway



Binary decision diagrams

A binary decision diagram is a graphical representation of a binary function $\phi : \{0,1\}^n \to \{0,1\}$. We may think of this as the structure function of some binary monotone system.

- The root and the intermediate nodes are drawn as circular nodes, and labelled with indices of the binary input variables
- The edges represent decisions regarding the values of the input variables, i.e., whether the value of an input variable is either 0 or 1
- The square leaf nodes represent cases where the associated binary function is fully determined, and the labels in this case represent the resulting binary values, i.e., either 0 or 1.



Binary decision diagrams (cont.)

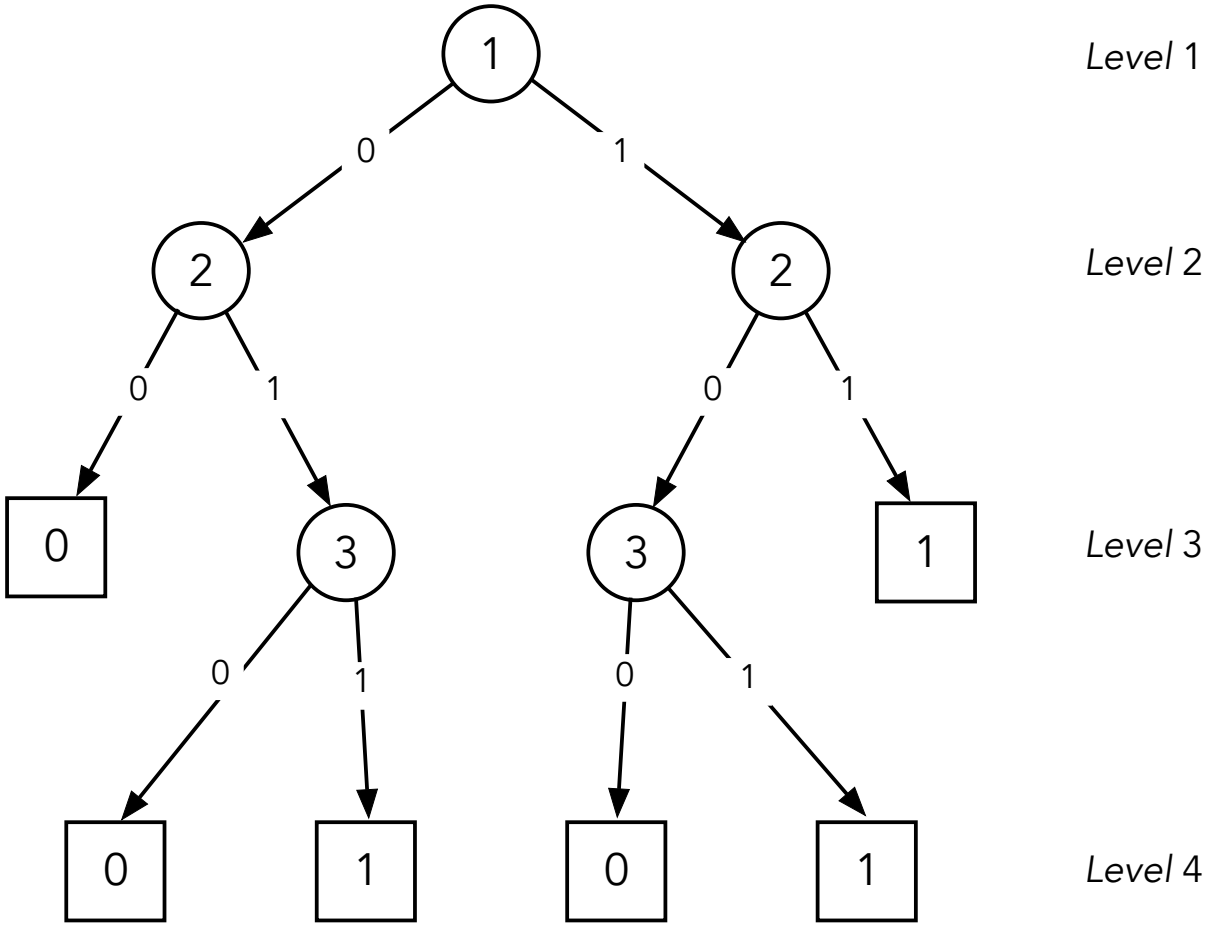


Figure: A binary decision diagram of a 2-out-of-3 system



Binary decision diagrams (cont.)

- To each node in the diagram we associate a binary function of the remaining binary variables whose values are not yet fixed.
- The root node in the diagram, labelled 1, is associated with the function ϕ itself since at this stage none of the variables are fixed.
- The lower level nodes are associated with structure functions of minors of the system under consideration.

Since each input variable has two possible values, 0 and 1, each node has exactly two children.

CONVENTION:

- The lefthand child has the binary function given that the respective input variable is fixed to be 0
- The righthand child has the binary function given that the respective input variable is fixed to be 1



A generic BDD construction algorithm

Algorithm (BDD)

Let $\phi = \phi(\mathbf{x})$ be a binary function, and let $C = \{1, \ldots, n\}$ denote the indices of the input variables.

STEP 1. *At level 1 create the root node with the function $\phi(\mathbf{x})$.*

- *If $\phi(\mathbf{x})$ is trivial, merge the root with the relevant leaf node.*
- *Select $i \in C$, and label the root node by this i.*

...

STEP k. *At level k create the lefthand and righthand children of each of the nodes at level $(k-1)$ with functions obtained by fixing the value of their respective input variables (indicated by their labels) to 0 and 1 respectively.*

- *Merge nodes with trivial functions with the relevant leaf nodes (if any).*
- *For each intermediate node select $i \in C$ among the variables that have not yet been fixed in this node's function, and label the node by this i.*

...

BDD construction – The missing pieces

NOTE: This is a generic algorithm which can be used on different types of systems. In order to apply this script to a specific class of systems, we need:

- A method for representing the binary functions obtained as a result of fixing the values of the input variables (*restriction* and *contraction*)
- A method for determining if a binary function is trivial
- A method for selecting input variables for each node in the BDD

Implementations of these methods depend strongly on the class of systems under consideration, and on the chosen representation of the binary functions.



ROBDD (cont.)

By merging the two intermediate nodes at level 3 and the leaf nodes, we get the reduced diagram below:

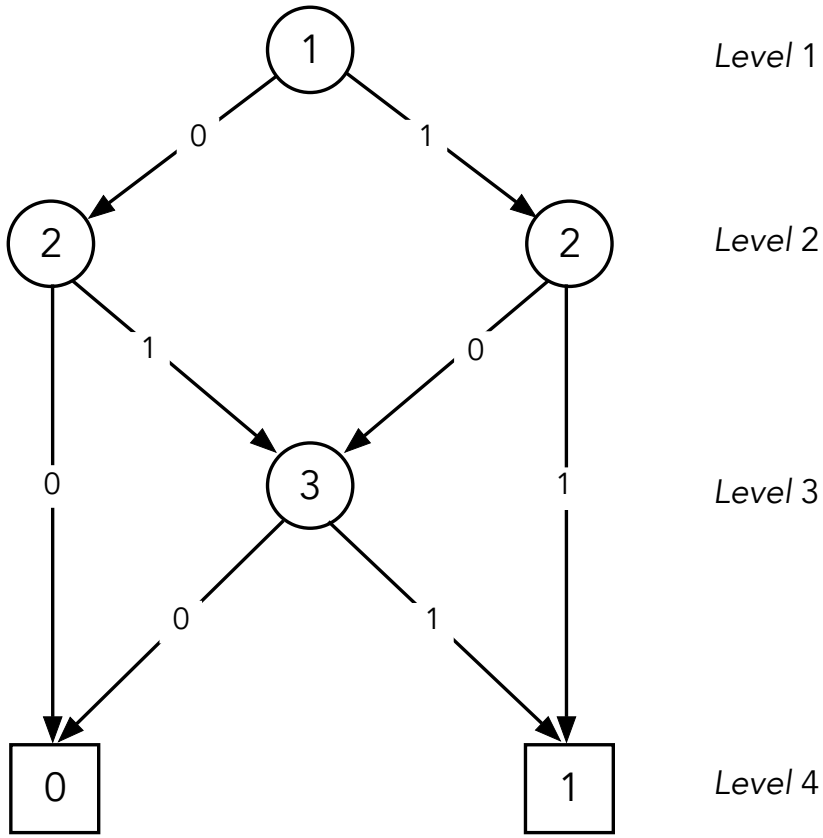


Figure: A reduced ordered binary decision diagram of a 2-out-of-3 system



A generic ROBDD construction algorithm

Algorithm (ROBDD)

Let $\phi = \phi(\mathbf{x})$ be a binary function, where $\mathbf{x} = (x_1, \ldots, x_n)$ is indexed in accordance with a given order of the input variables.

STEP 1. *At level 1 create the root node with the function $\phi(\mathbf{x})$.*

- *If $\phi(\mathbf{x})$ is trivial, merge the root with the relevant leaf node.*
- *Label the root node by the index 1.*

...

STEP k. *At level k create the lefthand and righthand children of each of the nodes at level $(k-1)$ with functions obtained by fixing the value of x_{k-1} to 0 and 1 respectively.*

- *Merge nodes with equal functions (if any).*
- *Merge nodes with trivial functions with the relevant leaf nodes (if any).*
- *Label all the intermediate nodes by the index k.*

...

BDD and ROBDD construction – The missing pieces

NOTE: The proposed algorithms are generic and can be used on different types of systems. In order to apply the algorithms to a specific class of systems, we need:

- A method for representing the binary functions obtained as a result of fixing the values of the input variables (restriction and contraction).
- A method for determining if a binary function is trivial.
- A method for determining whether two binary functions are identical.

Implementations of these methods depend strongly on the class of systems under consideration, and on the chosen representation of the binary functions.



ROBDD Implementation – Object structure

NOTE: Since we only consider non-decreasing functions, we have:

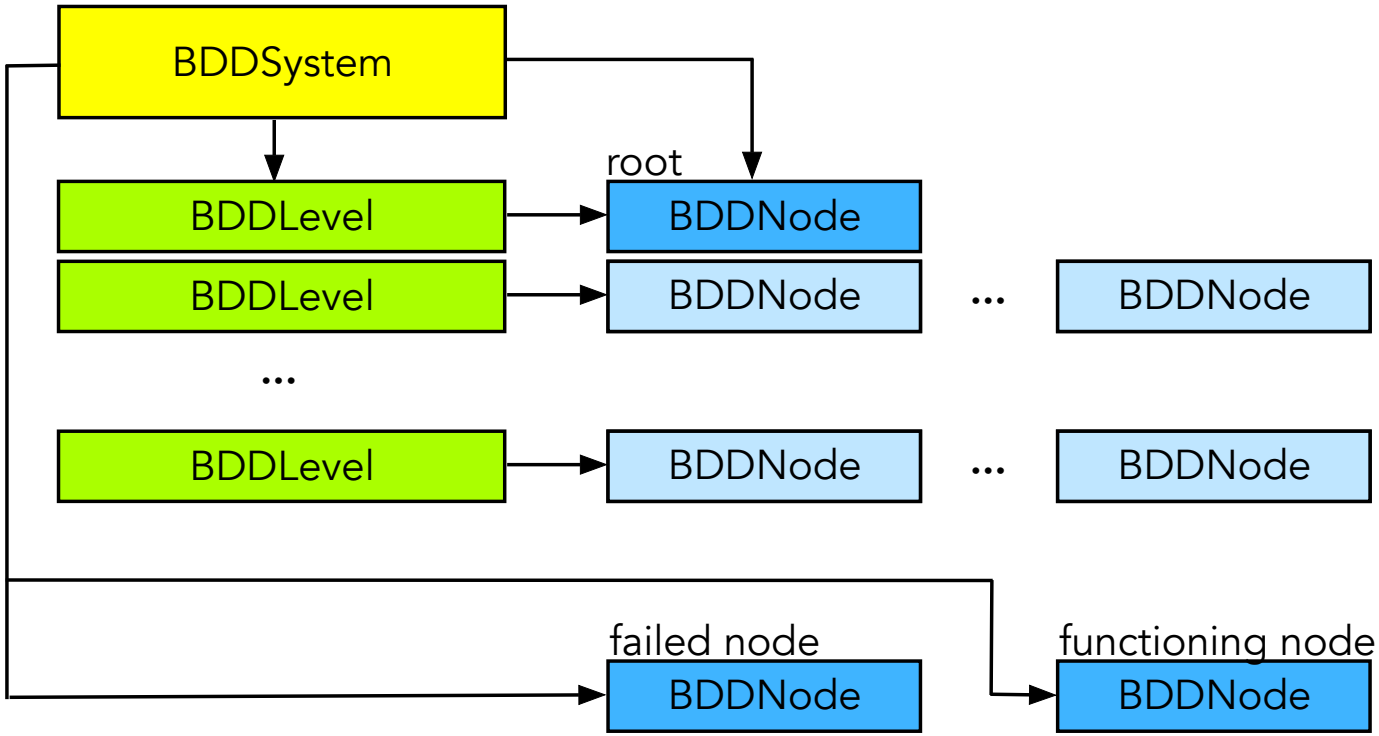
$$\begin{aligned} \phi(\mathbf{x}) \equiv 0 \quad &\Leftrightarrow \quad \phi(\mathbf{1}) = 0 \\ \phi(\mathbf{x}) \equiv 1 \quad &\Leftrightarrow \quad \phi(\mathbf{0}) = 1 \end{aligned} \tag{1}$$

This observation usually makes it easy to check if a binary function is trivial:

- A binary function ϕ is trivial and equal to 0 if and only if $\phi(\mathbf{1}) = 0$
- A binary function ϕ is trivial and equal to 1 if and only if $\phi(\mathbf{0}) = 1$



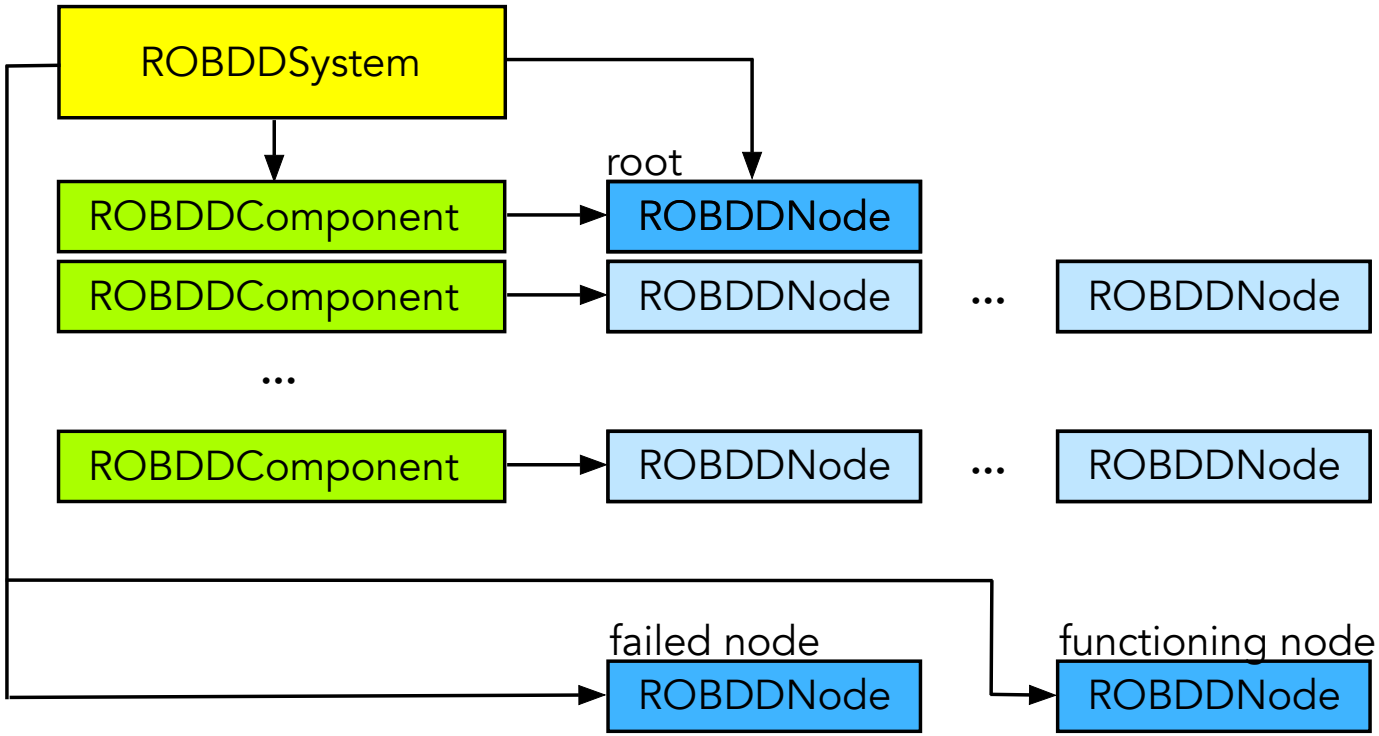
BDD Implementation – Object structure



- Each BDDNode object except the failed node and the functioning node has its BDDStructure object, which is a representation of the binary function associated with the node.
- Each BDDNode object except the failed node and the functioning node has its BDDComponent object (as indicated by its label).



ROBDD Implementation – Object structure



- Each ROBDDNode object except the failed node and the functioning node has its ROBDDStructure object, which is a representation of the binary function associated with the node.
- Since there is a one-to-one correspondence between a ROBDDComponent and its level in the diagram, ROBDDLevel objects are avoided.



ROBDDSystem – Implementation 1

```
class ROBDDSystem:
    def __init__(self, st):
        self.structure = st
        self.order = self.structure.getOrder()
        self.components = []
        for i in range(self.order):
            comp = ROBDDComponent(self, i)
            self.components.append(comp)
        self.root = ROBDDNode(self.components[0], self.structure)
        self.failedNode = ROBDDNode(None, None, nm = "D0")
        self.functioningNode = ROBDDNode(None, None, nm = "D1")
        for i in range(self.order):
            self.components[i].createLeaves()
            if (i+1) < self.order:
                self.components[i+1].simplify()

    def getComponent(self, i):
        if i < 0:
            return None
        elif i < self.order:
            return self.components[i]
        else:
            return None

    def getRoot(self):
        return self.root

    def getFailedNode(self):
        return self.failedNode

    def getFunctioningNode(self):
        return self.functioningNode
```



ROBDDSystem – Implementation 2

```
# Calculate reliability when all components have equal reliability.
# rel = The common reliability
def calculateReliability0(self, rel):
    self.root.initializeProbability(1)
    for i in range(1, self.order):
        self.components[i].initializeProbability()
    self.failedNode.initializeProbability()
    self.functioningNode.initializeProbability()
    for i in range(self.order):
        self.components[i].propagateProbability(rel)
    return self.failedNode.getProbability(), self.functioningNode.getProbability()

# Calculate reliability for a given vector of component reliabilities.
# rv = The vector of component reliabilities
def calculateReliability(self, rv):
    self.root.initializeProbability(1)
    for i in range(1, self.order):
        self.components[i].initializeProbability()
    self.failedNode.initializeProbability()
    self.functioningNode.initializeProbability()
    for i in range(self.order):
        self.components[i].propagateProbability(rv[i])
    return self.failedNode.getProbability(), self.functioningNode.getProbability()

def printSystem(self):
    for i in range(self.order):
        self.components[i].printComponent()
```



ROBDDComponent – Implementation 1

```
class ROBDDComponent:
    def __init__(self, sys, i):
        self.system = sys
        self.index = i
        self.nodes = []

    def getName(self):
        return COMP_PREFIX + str(self.index)

    def getNextComponent(self):
        return self.system.getComponent(self.index + 1)

    def getPrevComponent(self):
        return self.system.getComponent(self.index - 1)

    def getFailedNode(self):
        return self.system.getFailedNode()

    def getFunctioningNode(self):
        return self.system.getFunctioningNode()

    def addNode(self, nd):
        self.nodes.append(nd)

    def removeNode(self, nd):
        self.nodes.remove(nd)

    def getNodeIndex(self, nd):
        return self.nodes.index(nd)

    def createLeaves(self):
        k = len(self.nodes)
        for j in range(k):
            self.nodes[j].createLeaves()
```



ROBDDComponent – Implementation 2

```
def simplify(self):
    if DO_SIMPLIFY:
        k1 = len(self.nodes)
        for j1 in range(k1-1):
            i1 = k1-2-j1
            n1 = self.nodes[i1]
            k2 = len(self.nodes)
            for j2 in range(k2-i1-1):
                i2 = k2-1-j2
                n2 = self.nodes[i2]
                if n1.isEquivalentTo(n2):
                    n1.mergeNode(n2)

def initializeProbability(self):
    k = len(self.nodes)
    for j in range(k):
        self.nodes[j].initializeProbability()

def propagateProbability(self, rel):
    k = len(self.nodes)
    for j in range(k):
        self.nodes[j].propagateProbability(rel)

def printComponent(self):
    print(self.getName(), "[", sep = "", end = "")
    k = len(self.nodes)
    for j in range(k):
        self.nodes[j].printNode()
        if j < k-1:
            print(", ", sep = "", end = "")
    print("]")
```



ROBDDNode – Implementation 1

```
class ROBDDNode:
    def __init__(self, cp, st, rt = None, nm = ""):
        self.component = cp
        if cp != None:
            self.component.addNode(self)
        self.structure = st
        if rt == None:
            self.roots = None
        else:
            self.roots = [rt]
        self.left = None
        self.right = None
        self.probability = 0
        self.name = nm

    def getName(self):
        if self.component != None:
            return NODE_PREFIX + str(self.component.getNodeIndex(self))
        else:
            return self.name

    def addRoot(self, rt):
        if self.roots == None:
            self.roots = [rt]
        else:
            self.roots.append(rt)

    def replaceSucc(self, old_nd, new_nd):
        if self.left == old_nd:
            self.left = new_nd
        if self.right == old_nd:
            self.right = new_nd
```



ROBDDNode – Implementation 2

```
def isEquivalentTo(self, nd):
    return self.structure.isEquivalentTo(nd.structure)

def mergeNode(self, nd):
    m = len(nd.roots)
    for j in range(m):
        self.addRoot(nd.roots[j])
        nd.roots[j].replaceSucc(nd, self)
    self.structure.mergeStructure(nd.structure)
    self.component.removeNode(nd)

def isRoot(self):
    return (self.roots == None)

def isLeaf(self):
    return (self.component == None)

def initializeProbability(self, p = 0):
    self.probability = p

def addProbability(self, p):
    self.probability += p

def getProbability(self):
    return self.probability
```



ROBDDNode – Implementation 3

```
def createLeaves(self):
    if not self.isLeaf():
        nextComponent = self.component.getNextComponent()
        leftStructure = self.structure.restriction()
        if leftStructure.isFailed():
            self.left = self.component.getFailedNode()
        elif nextComponent != None:
            self.left = ROBDDNode(nextComponent, leftStructure, self)
        else:
            self.left = self.component.getFailedNode()
        rightStructure = self.structure.contraction()
        if rightStructure.isFunctioning():
            self.right = self.component.getFunctioningNode()
        elif nextComponent != None:
            self.right = ROBDDNode(nextComponent, rightStructure, self)
        else:
            self.right = self.component.getFunctioningNode()

def propagateProbability(self, p):
    self.left.addProbability(self.probability * (1-p))
    self.right.addProbability(self.probability * p)

def printNode(self):
    if PRINT_PROBABILITIES:
        print(self.getName(), "(", self.getProbability(), ")", sep = "", end = "")
    else:
        print(self.getName(), "()", sep = "", end = "")
```



ROBDDStructure – Abstract implementation

```
class ROBDDStructure:
    def __init__(self, n):
        self.order = n

    def getOrder(self):
        return self.order

    def restriction(self):
        pass

    def contraction(self):
        pass

    def isFunctioning(self):
        pass

    def isFailed(self):
        pass

    def isEquivalentTo(self, s):
        pass

    def mergeStructure(self, s):
        pass
```



Threshold systems



Threshold systems

Recall that structure function of a threshold system has the following form:

$$\phi(\mathbf{x}) = I(\sum_{i=1}^{n} a_i x_i \geq b), \tag{2}$$

where $a_1, \ldots, a_n$ and b are non-negative real numbers.

In order to construct BDDs and ROBDDs for such systems we need to implement the methods listed in relation to the generic implementation:

- A method for representing the binary functions obtained as a result of fixing the values of the input variables (*restriction* and *contraction*).
- A method for determining if a binary function is trivial.
- A method for determining whether two binary functions are identical.

NOTE: For threshold systems the input variables should be ordered with respect to their weights: Input variable 1 is the one with the highest weight, input variable 2 is the one with the second highest weight etc.



Threshold systems – Restrictions and contractions

The structure function of a threshold system is uniquely defined by the weights and threshold. Thus, these quantities provide a representation the system. Formally, we refer to such a system as a TS($\mathbf{a}, b$)-system.

The binary functions obtained by fixing the value of the first input variable, are given by:

$$\phi(0_1, \mathbf{x}) = I(\sum_{i=2}^{n} a_i x_i \geq b),$$

$$\phi(1_1, \mathbf{x}) = I(\sum_{i=2}^{n} a_i x_i \geq b - a_1)$$

Hence, we have:

- The restriction of a TS($\mathbf{a}, b$)-system is a TS($\mathbf{a}_{-1}, b$)-system.
- The contraction of a TS($\mathbf{a}, b$)-system is a TS($\mathbf{a}_{-1}, b - a_1$)-system.

where $\mathbf{a}_{-1} = (a_2, \ldots, a_n)$.



Threshold systems – Trivial systems

We recall that since we only consider non-decreasing functions, we have:

$$\begin{aligned} \phi(\mathbf{x}) \equiv 0 \quad &\Leftrightarrow \quad \phi(\mathbf{1}) = 0 \\ \phi(\mathbf{x}) \equiv 1 \quad &\Leftrightarrow \quad \phi(\mathbf{0}) = 1 \end{aligned}$$

For a threshold system we have:

$$\phi(\mathbf{0}) = I(\sum_{i=1}^{n} a_i \cdot 0 \geq b) = I(0 \geq b)$$

$$\phi(\mathbf{1}) = I(\sum_{i=1}^{n} a_i \cdot 1 \geq b) = I(\sum_{i=1}^{n} a_i \geq b)$$

Hence, we have:

$$\begin{aligned} \phi(\mathbf{x}) \equiv 0 \quad &\Leftrightarrow \quad \sum_{i=1}^{n} a_i < b \\ \phi(\mathbf{x}) \equiv 1 \quad &\Leftrightarrow \quad b \leq 0 \end{aligned}$$



Threshold systems – Equal systems

Since a threshold system is uniquely defined by the weights and threshold, it follows that a sufficient condition for that two threshold systems are equal is that they have equal weights and threshold.

Moreover, since we only need to compare threshold systems at the same level in the diagram, these systems always have the same weights.

More specifically, all threshold systems at level k in the diagram has weights:

$$a_k, a_{k+1}, \ldots, a_n$$

Thus, to check if two threshold systems at the same level are equal, we only need to compare their respective thresholds.



ROBDDThreshold – Implementation

```
class ROBDDThreshold(ROBDDStructure):
    def __init__(self, w, wsum, th):
        super().__init__(len(w))
        self.weights = w
        self.weightsum = wsum
        self.threshold = th

    def restriction(self):
        w = []
        for i in range(1, self.order):
            w.append(self.weights[i])
        wsum = self.weightsum - self.weights[0]
        return ROBDDThreshold(w, wsum, self.threshold)

    def contraction(self):
        w = []
        for i in range(1, self.order):
            w.append(self.weights[i])
        wsum = self.weightsum - self.weights[0]
        return ROBDDThreshold(w, wsum, self.threshold - self.weights[0])

    def isFunctioning(self):
        return self.threshold <= 0

    def isFailed(self):
        return self.threshold > self.weightsum

    def isEquivalentTo(self, s):
        return self.threshold == s.threshold
```



BDDThreshold – Example

```
##########################################################
#                                                        #
#    Construct a BDD for a given threshold               #
#    system and evaluate its reliability                 #
#                                                        #
##########################################################

from c_bdd import BDDSystem
from c_threshold import BDDThreshold

# Component reliabilities
p = 0.5

# Component weights
a = [12, 11, 8, 7, 6, 5, 3, 2]

# Threshold
b = 30

# Number of components
n = len(a)

# Sum of weights
wsum = 0
for wg in a:
    wsum += wg

sys = BDDSystem(BDDThreshold(a, wsum, b))

result = sys.calculateReliability0(p)
sys.printSystem()

print("")
print("System unreliability = ", result[0])
print("System reliability = ", result[1])
```



BDDThreshold – Example (cont.)

```
L0[N0(1)]
L1[N0(0.5), N1(0.5)]
L2[N0(0.25), N1(0.25), N2(0.25), N3(0.25)]
L3[N0(0.125), N1(0.125), N2(0.125), N3(0.125), N4(0.125), N5(0.125)]
L4[N0(0.0625), N1(0.0625), N2(0.0625), N3(0.0625), N4(0.0625), N5(0.0625), N6(0.0625), N7(0.0625)]
L5[N0(0.03125), N1(0.03125), N2(0.03125), N3(0.03125), N4(0.03125), N5(0.03125), N6(0.03125), N7(0.03125), N8(0.03125), N9(0.03125)]
L6[N0(0.015625), N1(0.015625), N2(0.015625), N3(0.015625), N4(0.015625), N5(0.015625), N6(0.015625), N7(0.015625), N8(0.015625), N9(0.015625)]
L7[N0(0.0078125), N1(0.0078125), N2(0.0078125), N3(0.0078125), N4(0.0078125), N5(0.0078125), N6(0.0078125), N7(0.0078125), N8(0.0078125)]

System unreliability =  0.58984375
System reliability =  0.41015625
```



ROBDDThreshold – Example

```
#########################################################
#                                                       #
#    Construct an ROBDD for a given threshold           #
#    system and evaluate its reliability                #
#                                                       #
#########################################################

from c_robdd import ROBDDSystem
from c_threshold import ROBDDThreshold

# Component reliabilities
p = 0.5

# Component weights
a = [12, 11, 8, 7, 6, 5, 3, 2]

# Threshold
b = 30

# Number of components
n = len(a)

# Sum of weights
wsum = 0
for wg in a:
    wsum += wg

sys = ROBDDSystem(ROBDDThreshold(a, wsum, b))

result = sys.calculateReliability0(p)
sys.printSystem()

print("")
print("System unreliability = ", result[0])
print("System reliability = ", result[1])
```



ROBDDThreshold – Example (cont.)

```
C0[N0(1)]
C1[N0(0.5), N1(0.5)]
C2[N0(0.25), N1(0.25), N2(0.25), N3(0.25)]
C3[N0(0.125), N1(0.125), N2(0.125), N3(0.125), N4(0.125), N5(0.125)]
C4[N0(0.0625), N1(0.0625), N2(0.125), N3(0.0625), N4(0.0625), N5(0.0625), N6(0.0625)]
C5[N0(0.03125), N1(0.03125), N2(0.0625), N3(0.0625), N4(0.03125), N5(0.03125), N6(0.03125), N7(0.03125)]
C6[N0(0.046875), N1(0.03125), N2(0.046875), N3(0.015625), N4(0.015625)]
C7[N0(0.0390625), N1(0.03125)]

System unreliability =  0.58984375
System reliability =  0.41015625
```



BDDThreshold – Example

```
############################################################
#                                                          #
#    Construct a BDD for a given 8-out-of-12               #
#    system and evaluate its reliability                   #
#                                                          #
############################################################

from c_bdd import BDDSystem
from c_threshold import BDDThreshold

# Component reliabilities
p = 0.5

# Component weights
a = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

# Threshold
b = 8

# Number of components
n = len(a)

# Sum of weights
wsum = 0
for wg in a:
    wsum += wg

sys = BDDSystem(BDDThreshold(a, wsum, b))

result = sys.calculateReliability0(p)
sys.printSystem()

print("")
print("System unreliability = ", result[0])
print("System reliability = ", result[1])
```



BDDThreshold – Example (cont.)

```
L0[N0(1)]
L1[N0(0.5), N1(0.5)]
L2[N0(0.25), N1(0.25), N2(0.25), N3(0.25)]
L3[N0(0.125), N1(0.125), N2(0.125), N3(0.125), N4(0.125), N5(0.125), N6(0.125), N7(0.125)]
L4[N0(0.0625), N1(0.0625), N2(0.0625), N3(0.0625), N4(0.0625), N5(0.0625), N6(0.0625), N7(0.0625), N8(0.0625), N9(0.0625), N10(0.0625), N11(0.0625), N12(0.0625), N13(0.0625), N14(0.0625),
L5[N0(0.03125), N1(0.03125), N2(0.03125), N3(0.03125), N4(0.03125), N5(0.03125), N6(0.03125), N7(0.03125), N8(0.03125), N9(0.03125), N10(0.03125), N11(0.03125), N12(0.03125), N13(0.03125)
L6[N0(0.015625), N1(0.015625), N2(0.015625), N3(0.015625), N4(0.015625), N5(0.015625), N6(0.015625), N7(0.015625), N8(0.015625), N9(0.015625), N10(0.015625), N11(0.015625), N12(0.015625),
L7[N0(0.0078125), N1(0.0078125), N2(0.0078125), N3(0.0078125), N4(0.0078125), N5(0.0078125), N6(0.0078125), N7(0.0078125), N8(0.0078125), N9(0.0078125), N10(0.0078125), N11(0.0078125), N1
L8[N0(0.00390625), N1(0.00390625), N2(0.00390625), N3(0.00390625), N4(0.00390625), N5(0.00390625), N6(0.00390625), N7(0.00390625), N8(0.00390625), N9(0.00390625), N10(0.00390625), N11(0.0
L9[N0(0.001953125), N1(0.001953125), N2(0.001953125), N3(0.001953125), N4(0.001953125), N5(0.001953125), N6(0.001953125), N7(0.001953125), N8(0.001953125), N9(0.001953125), N10(0.00195312
L10[N0(0.0009765625), N1(0.0009765625), N2(0.0009765625), N3(0.0009765625), N4(0.0009765625), N5(0.0009765625), N6(0.0009765625), N7(0.0009765625), N8(0.0009765625), N9(0.0009765625), N10
L11[N0(0.00048828125), N1(0.00048828125), N2(0.00048828125), N3(0.00048828125), N4(0.00048828125), N5(0.00048828125), N6(0.00048828125), N7(0.00048828125), N8(0.00048828125), N9(0.0004882

System unreliability =  0.80615234375
System reliability =  0.19384765625
```



ROBDDThreshold – Example

```
############################################################
#                                                          #
#    Construct an ROBDD for a given 8-out-of-12            #
#    system and evaluate its reliability                   #
#                                                          #
############################################################

from c_robdd import ROBDDSystem
from c_threshold import ROBDDThreshold

# Component reliabilities
p = 0.5

# Component weights
a = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

# Threshold
b = 8

# Number of components
n = len(a)

# Sum of weights
wsum = 0
for wg in a:
    wsum += wg

sys = ROBDDSystem(ROBDDThreshold(a, wsum, b))

result = sys.calculateReliability0(p)
sys.printSystem()

print("")
print("System unreliability = ", result[0])
print("System reliability = ", result[1])
```



ROBDDThreshold – Example (cont.)

```
C0[N0(1)]
C1[N0(0.5), N1(0.5)]
C2[N0(0.25), N1(0.5), N2(0.25)]
C3[N0(0.125), N1(0.375), N2(0.375), N3(0.125)]
C4[N0(0.0625), N1(0.25), N2(0.375), N3(0.25), N4(0.0625)]
C5[N0(0.15625), N1(0.3125), N2(0.3125), N3(0.15625), N4(0.03125)]
C6[N0(0.234375), N1(0.3125), N2(0.234375), N3(0.09375), N4(0.015625)]
C7[N0(0.2734375), N1(0.2734375), N2(0.1640625), N3(0.0546875), N4(0.0078125)]
C8[N0(0.2734375), N1(0.21875), N2(0.109375), N3(0.03125)]
C9[N0(0.24609375), N1(0.1640625), N2(0.0703125)]
C10[N0(0.205078125), N1(0.1171875)]
C11[N0(0.1611328125)]

System unreliability =  0.80615234375
System reliability =  0.19384765625
```

