Institut für Echtzeitsysteme
und Softwaretechnik
Prof. Dr. Derk Rembold

Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

2022

Praktikum Betriebssicherheit

Aufgabenblatt 3

Umwandlung eines Fehlerbaums in ein BDD

Anforderungen:

- Die Aufgabe wird in Python programmiert.
- Die Aufgabe wird von jedem Teilnehmer einzeln erstellt!
- Der Teilnehmer kommt rechtzeitig zur Abnahme auf den Dozenten zu. Die Abnahme erfolgt für jeden Teilnehmer einzeln. Die Kenntnis des Quellcodes wird erwartet.
- Programmcode wird auf Ilias hochgeladen. Die Lokation wird im Praktikum bekanntgegeben. Das File hat folgendes Format:
    - <Name>_<Vorname>_<Matrikelnummer>_Aufgabe_3_Programmcode.py

- **Es dürfen keine existierenden Libraries verwendet werden, welche Fehlerbäume in BDD's oder ähnliches umwandeln**
- **Es gibt eine Frist für die Abnahme und das Hochladen der Files. Diese sind in Ilias ersichtlich.**
- **Die hochgeladenen Files werden nach der Frist nochmals kontrolliert. Erst nach dieser Kontrolle gilt die Aufgabe als vollständig bestanden.**

Einleitung

In Aufgabe 2 hat der Teilnehmer bereits den Fehlerbaum als Baum programmiert. In der Vorlesung wurden u.a. Binary Decision Diagrams (BDD) behandelt. Hier wurde gezeigt, wie ein Fehlerbaum als BDD umgewandelt werden kann. Ein BDD hat Knoten (dargestellt als Kreise) mit zwei Ausgängen: „0“ und „1“. Diese Ausgänge sind wiederrum mit weiteren Knoten verbunden. Am unteren Ende des Graphen sind zwei Stubs (dargestellt als Rechtecke: „0“ und „1“). Dort laufen die Verbindungen der Knoten wieder zusammen. Die Knoten werden in dieser Aufgabenstellung als BDDEVENTs bezeichnet.
Bei dieser Aufgabe soll ein Fehlerbaum in einen BDD umgewandelt werden. Zur Vereinfachung sollen hier nur Fehlerbaum-Elemente mit zwei Eingängen verwendet werden; also alle OR-Gatter haben nur zwei Eingänge und alle AND-Gatter haben nur zwei Eingänge. Die NOT-Gatter werden in dieser Aufgabe nicht behandelt.

Aufgabe

Wandeln Sie in Python einen beliebigen Fehlerbaum (siehe Aufgabe 2) in ein BDD um und stellen Sie diesen graphisch dar. Siehe Bild 1.

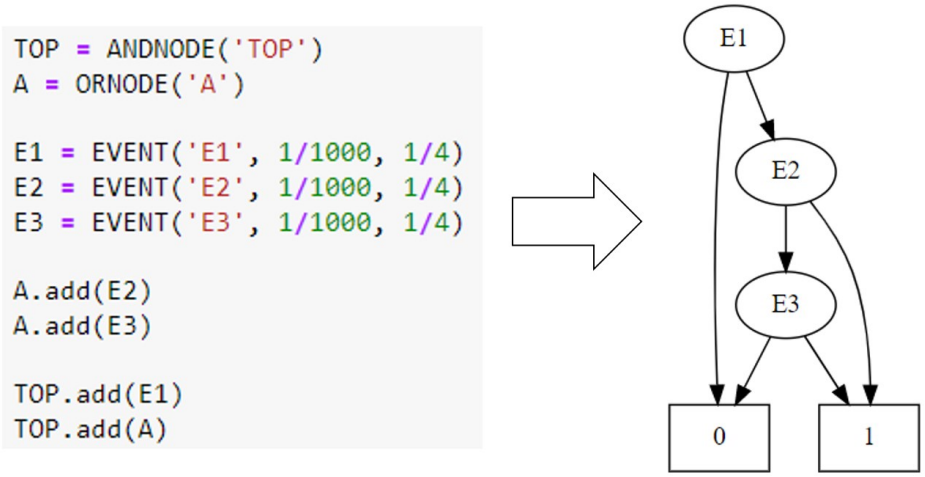

Bild 1: Fehlerbaum (siehe Aufgabe 2) umgewandelt in BDD

Die folgenden Erklärungen beschreiben einen möglichen Lösungsansatz. Auch weitere Lösungsansätze können zur erfolgreichen Bearbeitung dieser Aufgabe führen.

In Bild 1 links wird Python-Code dargestellt (siehe Aufgabe 2), dabei wird ein AND-Gatter instanziiert und der Variable TOP zugeordnet. TOP hat an den Eingängen ein Event 1 und ein Event A, welches der Ausgang des Gatters A ist. Das Gatter A selbst ist ein OR-Gatter mit zwei weiteren Events E2 und E3 an seinen Eingängen. Zum Aufbau des BDD's wird zunächst nur das Gatter TOP mit seinen Events E1 und A betrachtet. Das zugehörige BDD wird in Bild 2 dargestellt. Das Event E1 aus dem Fehlerbaum wird im BDD als ein Objekt der Klasse BDDEVENT gezeigt. Da hinter Event A noch ein weiteres Schaltelement (in diesem Fall ein OR-Gatter) befindet, wird dieses als ein NODE-Objekt dargestellt.

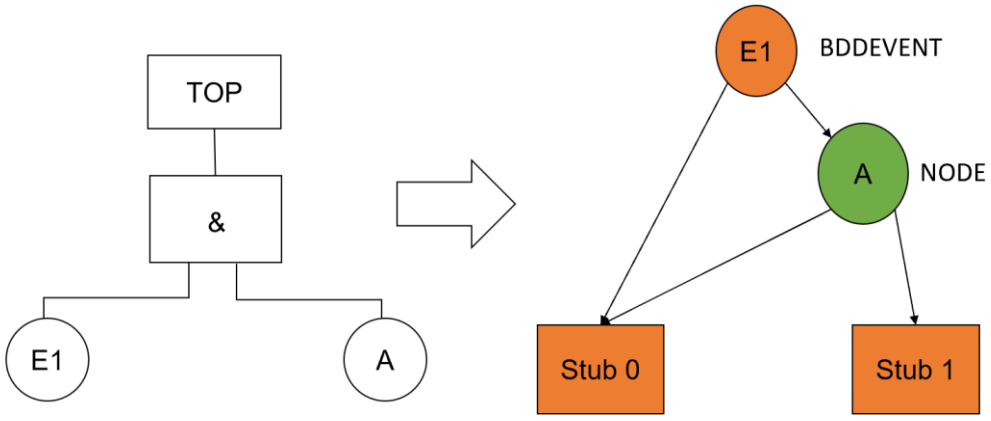

Bild 2: Fehlerbaum TOP als BDD mit Eingängen E1 und A

In einem zweiten Schritt kann der Inhalt von Gatter A mit seinen Events E2 und E3 in den BDD gehängt werden. In Bild 3 sieht man, dass aus den Events 2 und 3 des Fehlerbaums BDDEVENT-Objekte instanziiert wurden. Die CONNECT-Objekte (siehe Bild 3) sind Schnittstellen, um die

Verbindungen der BDDEVENT-Objekte aus dem NODE A zu führen. Der Einsatz der CONNECT-Objekte erweist sich als vorteilhaft, da jetzt NODE A klare Schnittstellen nach außen hat.

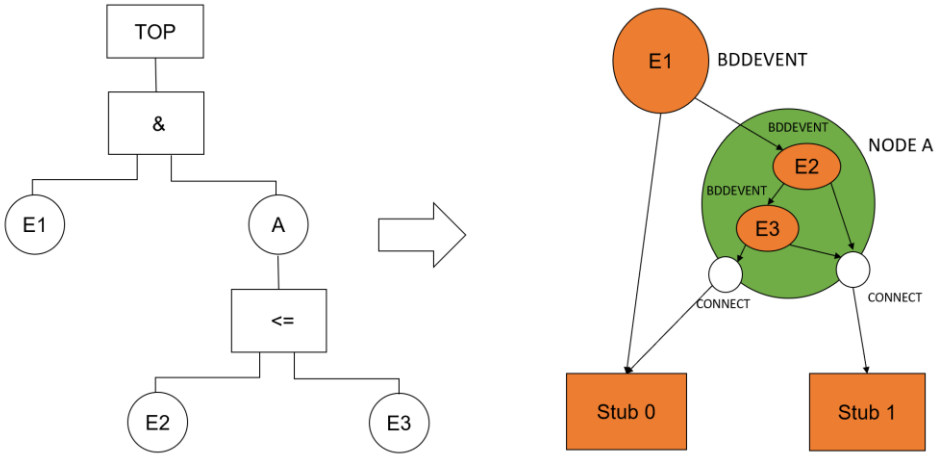

Bild 3: Vollständiger BDD aus Fehlerbaum

Die Klasse CONNECT kann wie folgt programmiert werden, siehe Bild 4.

```
class CONNECT:
    def __init__(self,name):
        self.name = name
        self.con = None
    #...
    def getname(self):
        return self.name
```

Bild 4: Klasse CONNECT

Das Attribut self.con kann die Verbindung (Pointer) zum Folge-Knoten (NODE oder BDDEVENT) enthalten. Im Beispiel von Bild 3 wäre dies das Stub 0 bzw. 1 (Stub's können auch als Python-None definiert werden, wenn gewünscht). Das Attribut self.con kann weitere NODE-Objekte und BDDEVENT-Objekte verbinden. Eine NODE-Klasse wird in Bild 5 gezeigt. Bei der Instanziierung werden auch zwei CONNECT-Objekte aufgebaut, einen für den linken Ausgang (self.zero), und einen für den rechten Ausgang (self.one). Innerhalb der Klasse NODE wird die Datenstruktur mit BDDEVENTs und weiteren möglichen NODEs aufgebaut. Siehe Bild 3 rechts, grün; hier sieht man, dass in NODE A die Datenstruktur für ein OR-Gatter eingehängt wurde.

```
class NODE:
    def __init__(self,name):
        self.name = name

        self.zero = CONNECT("ZERO")
        self.one = CONNECT("ONE")

    #...

    def getname(self):
        return self.name
```

Bild 5: Klasse NODE

Bild 6 zeigt die Klasse BDDEVENT. Der zu implementierende Algorithmus instanziiert ein BDDEVENT-Objekt, sobald bei Iterieren durch den Fehlerbaum ein Eingangs-Event gefunden wird. Wie in Bild 3 dargestellt, wird dieser in den BDD gehängt. Die Attribute self.zero und self.one können die Verbindungen zum Folge-Objekt enthalten, auch hier siehe Bild 2 oder Bild 3. Ein Folge-Objekt kann ein BDDVENT-Objekt, ein NODE-Objekt oder ein Stub sein (wobei ein Stub als ein Python-None repräsentiert werden kann).

```
class BDDEVENT:
    def __init__(self,name):
        self.name = name
        self.zero = None
        self.one = None

    #...

    def getname(self):
        return self.name
```

Bild 6: Klasse BDDEVENT

Bild 7 zeigt den unvollständigen Beispiel-Code zur Generierung eines BDDs aus einem Fehlerbaum. Das oberste Objekt der zu generierende BDD-Datenstruktur wird an das Attribut self.nroot gehängt. Die create-Methode hat das oberste Gatter eines Fehlerbaum als Eingangsparameter (im Beispiel oben würde man TOP eingeben). Die Funktion bestimmt zunächst, ob der Eingangsparameter vom Typ eines AND bzw. eines OR-Gater ist. Danach werden die Methoden createand oder createor aufgerufen. Dort wird bestimmt ob BDDEVENTs oder NODEs instanziiert werden und entsprechend miteinander verbunden (bzw. verpointert). Die BDDEVENT Ausgänge innerhalb eines NODE-Objekts werden mit den CONNECT-Objekten der NODE verbunden. Der Algorithmus ruft Methode create solange **rekursiv** auf, bis alle Fehlerbaum-Gatter durchiteriert sind und der BDD aufgebaut ist. Die Anzahl der BDDEVENTs muss gleich sein wie die Anzahl der Events des Fehlerbaums.
Die show-Methode der Klasse FT2BDD generiert den BDD-Baum mit graphviz aus der generierten Datenstruktur, welche an self.nroot angehängt ist.

```
class FT2BDD:

    def __init__(self):
        self.nroot = None
        #...

    def show(self):
        #...
        return self.dot

    def create(self, ntop, ft):

        if ntop == None:
            ntop = NODE("root")

        if self.nroot == None:
            self.nroot = ntop

        if isinstance(ft, ANDNODE):
            self.createand(ntop, ft)
        elif isinstance(ft, ORNODE):
            self.createor(ntop, ft)
        else:
            assert(0> 1)

        return ntop

    def createand(self, ntop, ft):

        print("->createand")
        #...
        print("<-createand")

        return ntop

    def createor(self, ntop, ft):

        print("->createor")
        #...
        print("<-createor")

        return ntop
```

Bild 7: Klasse FT2BDD