

Lab 1: Fun with ROBDDs

I have implemented the functions associated with the ROBDD using Python 3.6.3 on the PyCharms IDE. The implementation involved manual generation of expressions and encoding them in the ROBDDs.

Following are some test cases which check the thoroughness of the code.

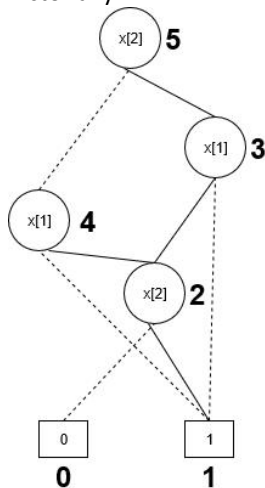
1. Build and Make

A. Simple Test Case -
 $\text{or}(\text{equiv}(x_1, x_2), x_3)$

The table generated by build and make is

```
✓ 1 3 T = {list} <class 'list'>: [[4, -1, -1], [4, -2, -2], [2, 0, 1], [1, 1, 2], [1, 2, 1], [0, 3, 4]]
> 1 3 0 = {list} <class 'list'>: [4, -1, -1]
> 1 3 1 = {list} <class 'list'>: [4, -2, -2]
> 1 3 2 = {list} <class 'list'>: [2, 0, 1]
> 1 3 3 = {list} <class 'list'>: [1, 1, 2]
> 1 3 4 = {list} <class 'list'>: [1, 2, 1]
> 1 3 5 = {list} <class 'list'>: [0, 3, 4]
1 3 6 _len_ = {int} 6
```

Pictorially :-



B. Testing all functions-

`and(implies(not(x0),equiv(1,x1)),not(x2))`

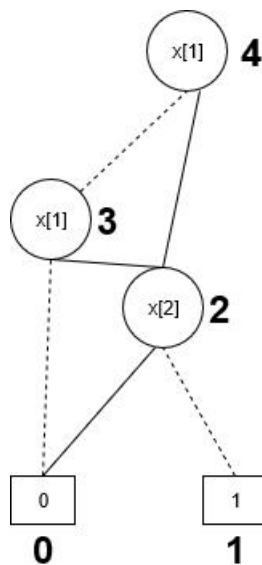
The table generated by build and make is

```

✓ T = {list} <class 'list': [[4, -1, -1], [4, -2, -2], [2, 1, 0], [1, 0, 2], [0, 3, 2]]
> 0 = {list} <class 'list': [4, -1, -1]
> 1 = {list} <class 'list': [4, -2, -2]
> 2 = {list} <class 'list': [2, 1, 0]
> 3 = {list} <class 'list': [1, 0, 2]
> 4 = {list} <class 'list': [0, 3, 2]

```

Pictorially ->



C. Testing by increasing the number of variables

`and(and(x0,x1),and(x2,x3)),and(and(x4,x5),and(x6,x7)))`

```

✓ T = {list} <class 'list': [[9, -1, -1], [9, -2, -2], [7, 0, 1], [6, 0, 2], [5, 0, 3], [4, 0, 4], [3, 0, 5], [2, 0, 6], [1, 0, 7], [0, 0, 8]]
> 00 = {list} <class 'list': [9, -1, -1]
> 01 = {list} <class 'list': [9, -2, -2]
> 02 = {list} <class 'list': [7, 0, 1]
> 03 = {list} <class 'list': [6, 0, 2]
> 04 = {list} <class 'list': [5, 0, 3]
> 05 = {list} <class 'list': [4, 0, 4]
> 06 = {list} <class 'list': [3, 0, 5]
> 07 = {list} <class 'list': [2, 0, 6]
> 08 = {list} <class 'list': [1, 0, 7]
> 09 = {list} <class 'list': [0, 0, 8]

```

Note : The time taken for this operation increases considerably due to recursion.

2. SatCount & AnySat

Satcount - Returns the possible number of satisfying conditions evaluating to True, for a given expression.

A. Simple Test Case-

`or(equiv(x1,x2),x3)`

Count of Satisfying Conditions is (SatCount) -> 6.0

One of the satisfying conditions is (AnySat) -> [0, 0, -1]

B. Testing all functions-

`and(implies(not(x0),equiv(1,x1)),not(x2))`

Count of Satisfying Conditions is (SatCount) -> 6.0

One of the satisfying conditions is (AnySat) -> [0, 1, 0, -1]

C. Testing by increasing the number of variables

`and(and(x0,x1),and(x2,x3)),and(x4,x5),and(x6,x7))`

Count of Satisfying Conditions is (SatCount) -> 1.0

One of the satisfying conditions is (AnySat) -> [1, 1, 1, 1, 1, 1, 1, 1]

D. Testing Negative test Case

`and(0,x1)`

Count of Satisfying Conditions is (SatCount) -> 0.0

Error!! :(

3. Restrict

Test Case 1 : Using the Simple Test Case1 -
or(equiv(x1,x2),x3)

Restricting $x_2 = 0$

Pictorially

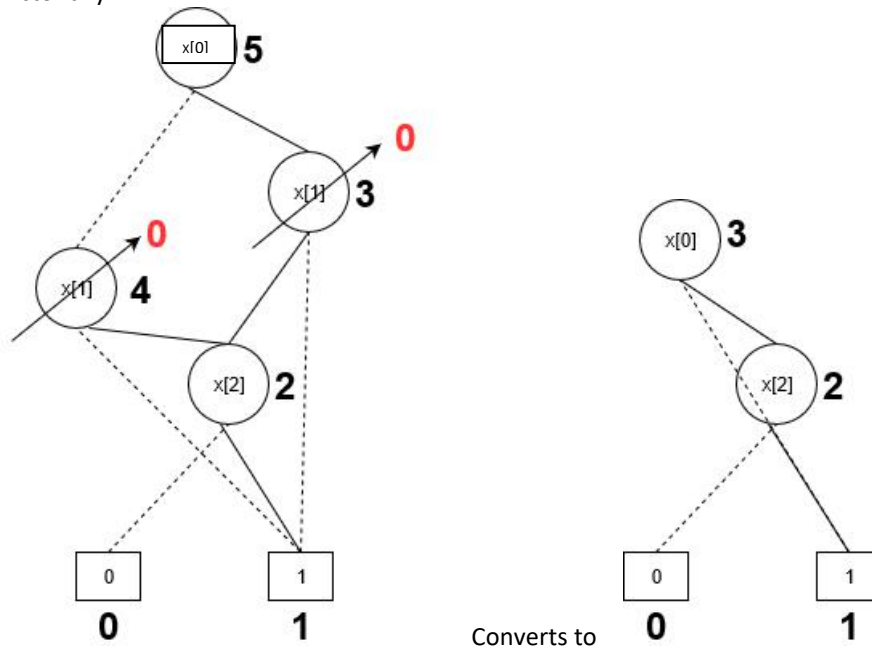







Table Looks like ->

```
▼  T_ = {list} <class 'list'>: [[4, -1, -1], [4, -2, -2], [2, 0, 1], [0, 1, 2]]
>  0 = {list} <class 'list'>: [4, -1, -1]
>  1 = {list} <class 'list'>: [4, -2, -2]
>  2 = {list} <class 'list'>: [2, 0, 1]
>  3 = {list} <class 'list'>: [0, 1, 2]
```

4. Apply

I have used the following expression to test Apply

ROBDD1 ->or(equiv(x1,x2),x3)

ROBDD2 ->equiv(and(x1,x2),x3)

Operator used -> and

ROBDD1 ->

```
▼ 1 3 T = {list} <class 'list': [[4, -1, -1], [4, -2, -2], [2, 0, 1], [1, 1, 2], [1, 2, 1], [0, 3, 4]]
  > 1 3 0 = {list} <class 'list': [4, -1, -1]
  > 1 3 1 = {list} <class 'list': [4, -2, -2]
  > 1 3 2 = {list} <class 'list': [2, 0, 1]
  > 1 3 3 = {list} <class 'list': [1, 1, 2]
  > 1 3 4 = {list} <class 'list': [1, 2, 1]
  > 1 3 5 = {list} <class 'list': [0, 3, 4]
```

ROBDD2 ->

```
▼ 1 3 T = {list} <class 'list': [[4, -1, -1], [4, -2, -2], [2, 1, 0], [2, 0, 1], [1, 2, 3], [0, 2, 4]]
  > 1 3 0 = {list} <class 'list': [4, -1, -1]
  > 1 3 1 = {list} <class 'list': [4, -2, -2]
  > 1 3 2 = {list} <class 'list': [2, 1, 0]
  > 1 3 3 = {list} <class 'list': [2, 0, 1]
  > 1 3 4 = {list} <class 'list': [1, 2, 3]
  > 1 3 5 = {list} <class 'list': [0, 2, 4]
```

Apply(ROBDD1,ROBDD,'and') ->

```
▼ 1 3 T = {list} <class 'list': [[4, -1, -1], [4, -2, -2], [2, 1, 0], [1, 2, 0], [2, 0, 1], [1, 0, 4], [0, 3, 5]]
  > 1 3 0 = {list} <class 'list': [4, -1, -1]
  > 1 3 1 = {list} <class 'list': [4, -2, -2]
  > 1 3 2 = {list} <class 'list': [2, 1, 0]
  > 1 3 3 = {list} <class 'list': [1, 2, 0]
  > 1 3 4 = {list} <class 'list': [2, 0, 1]
  > 1 3 5 = {list} <class 'list': [1, 0, 4]
  > 1 3 6 = {list} <class 'list': [0, 3, 5]
```

Parser ::

I had also built an expression parser to get the expression from a string, but due to lack of time, I was not able to integrate it in the ROBDD code.

Expression - >

```
test = 'and (and (1,2) , and (3,4)) '
```

```
▼ parseTree = {Tree} <__main__.Tree object at 0x00000205C4341588>
  data = {str} 'and'
  ▼ left = {Tree} <__main__.Tree object at 0x00000205C43415C0>
    data = {str} 'and'
    ▼ left = {Tree} <__main__.Tree object at 0x00000205C43416A0>
      data = {str} '1'
      left = {NoneType} None
      right = {NoneType} None
    ▼ right = {Tree} <__main__.Tree object at 0x00000205C4341668>
      data = {str} '2'
      left = {NoneType} None
      right = {NoneType} None
  ▼ right = {Tree} <__main__.Tree object at 0x00000205C4341710>
    data = {str} 'and'
    ▼ left = {Tree} <__main__.Tree object at 0x00000205C4341780>
      data = {str} '3'
      left = {NoneType} None
      right = {NoneType} None
    ▼ right = {Tree} <__main__.Tree object at 0x00000205C4341748>
      data = {str} '4'
      left = {NoneType} None
      right = {NoneType} None
```