Christopher Patterson - bool2cp Writeup
Summary of a Boolean to Conditional Permutation C++ Program
3/29/19

My program to convert a boolean function into conditional permutation form works on a specific input format I discuss below.

The program uses two input AND and NAND gates as well as NOT operations to implement the original boolean function (comprised of AND, NAND, OR, NOR, and XOR operators) specified as input to the program. To do this conversion, my program first converts the original boolean into a tree structure where each level is a further decomposition of the boolean function. The root is the whole boolean function and the leaves are single variables.

After obtaining the basic tree representation of decomposition, I recursively convert OR, NOR and XOR gates into their AND and NAND forms:

OR(x,y) -> NAND(x',y')

NOR(x,y) -> AND(x',y')

XOR(x,y) -> NAND(NAND(x,NAND(x,y)),NAND(y,NAND(x,y)))

and once I have the whole tree in NAND and AND form, I print out the original and converted version of the input function. From there I use BFS to print out each step of the decomposition (this may not be necessary but it helps with debugging and validation checking). Each level in the tree corresponds to a new level in the permutation decomposition. One thing to note with my implementation is that in lecture examples, NAND operations have a 5th, unconditional permutation that we fold into the 4th line of the expansion for brevity where mine does not fold this in.

To run the code:

```
cd <directory of folder>
make
./bool2cp
```

```
The program will then prompt for a boolean function input. In  order to parse the
boolean function, it must be converted to input format specified below where each
operator is a 2 input gate of variables x and y:
* AND(x,y)
* NAND(x,y)
* OR(x,y)
* NOR(x,y)
* XOR(x,y)
Examples of correct inputs are shown below as test cases.
```

Below are some test case boolean functions which show their conversion into well formed inputs for my program, the conversion to AND and NAND implementation, and the output of the final step in the form:

```
F(variables) = boolean function
      Program function input -> Conversion to AND and NAND
      Last permutation decomposition step
      .
      .
      .
```

Test Case 1 - Example from problem description

```
F(x, y, z) = (x z)' (x'y')'
     AND(NAND(x,z),NAND(x',y')) -> AND(NAND(x,z),NAND(x',y'))
     Step: 3
     1: {x: E, *}
     2: {z: D, *}
     3: {x: E', *}
     4: {z: D', *}
     5: {*: C, C}
     6: {x: *, D}
     7: {y: *, C}
     8: {x: *, D'}
     9: {y: *, C'}
     10: {*: B, B}
     11: {x: D, *}
     12: {z: E, *}
     13: {x: D', *}
     14: {z: E', *}
     15: {*: C', C'}
     16: {x: *, C}
     17: {y: *, D}
     18: {x: *, C'}
     19: {y: *, D'}
     20: {*: B', B'}
```

Test Case 2 - Expanding {xz: C'A, A} step

```
F(x,y,z) = ((xy)(xz))'
        NAND(AND(x,y),AND(x,z)) -> NAND(AND(x,y),AND(x,z))
        Step: 3
        1: {x: C, *}
        2: {y: D, *}
        3: {x: C', *}
        4: {y: D', *}
        5: {x: D, *}
        6: {z: E, *}
        7: {x: D', *}
        8: {z: E', *}
        9: {x: D, *}
        10: {y: C, *}
        11: {x: D', *}
        12: {y: C', *}
        13: {x: E, *}
        14: {z: D, *}
        15: {x: E', *}
        16: {z: D', *}
        17: {*: A, A}
```

Test Case 3 - XOR

```
F(x,y) = x ^ y
        XOR(x,y) -> NAND(NAND(x,NAND(x,y)),NAND(y,NAND(x,y)))
        Step: 4
        1: {x: C, *}
        2: {x: E, *}
        3: {y: B, *}
        4: {x: E', *}
        5: {y: B', *}
        6: {x: C', *}
        7: {x: B, *}
        8: {y: E, *}
        9: {x: B', *}
        10: {y: E', *}
        11: {y: D, *}
        12: {x: D, *}
```

```
13: {y: A, *}
14: {x: D', *}
15: {y: A', *}
16: {y: D', *}
17: {x: A, *}
18: {y: D, *}
19: {x: A', *}
20: {y: D', *}
21: {x: D, *}
22: {x: D, *}
23: {y: E, *}
24: {x: D', *}
25: {y: E', *}
26: {x: D', *}
27: {x: E, *}
28: {y: D, *}
29: {x: E', *}
30: {y: D', *}
31: {y: E, *}
32: {x: E, *}
33: {y: B, *}
34: {x: E', *}
35: {y: B', *}
36: {y: E', *}
37: {x: B, *}
38: {y: E, *}
39: {x: B', *}
40: {y: E', *}
41: {*: A, A}
```

Test Case 4 - Nesting

```
F(x,y,z) = (x|z)&((y&z)&x')'
    AND(OR(x,z),NAND(AND(y,z),x')) -> AND(NAND(x',z'),NAND(AND(y,z),x'))
    Step: 4
1: {x: *, E}
2: {z: *, D}
3: {x: *, E'}
4: {z: *, D'}
5: {*: C, C}
```

```
6: {y: E, *}
7: {z: B, *}
8: {y: E', *}
9: {z: B', *}
10: {x: *, C}
11: {y: B, *}
12: {z: E, *}
13: {y: B', *}
14: {z: E', *}
15: {x: *, C'}
16: {*: B, B}
17: {x: *, D}
18: {z: *, E}
19: {x: *, D'}
20: {z: *, E'}
21: {*: C', C'}
22: {y: D, *}
23: {z: E, *}
24: {y: D', *}
25: {z: E', *}
26: {x: *, D}
27: {y: E, *}
28: {z: D, *}
29: {y: E', *}
30: {z: D', *}
31: {x: *, D'}
32: {*: B', B'}
```

Test Case 5 - Yadu's OR test case

```
F(x,y) = x+y = (x'y')'
    OR(x,y) -> NAND(x',y')
    Step: 2
    1: {x: *, B}
    2: {y: *, C}
    3: {x: *, B'}
    4: {y: *, C'}
    5: {*: A, A}
```

Test Case 6 - Yadu's AND test case

```
F(x,y,z) = (x'y'z'),
    NAND(AND(x',y'),z') -> NAND(AND(x',y'),z')
    Step: 3
    1: {x: *, C}
    2: {y: *, D}
    3: {x: *, C'}
    4: {y: *, D'}
    5: {z: *, C}
    6: {x: *, D}
    7: {y: *, C}
    8: {x: *, D'}
    9: {y: *, C'}
    10: {z: *, C'}
    11: {*: A, A}
```