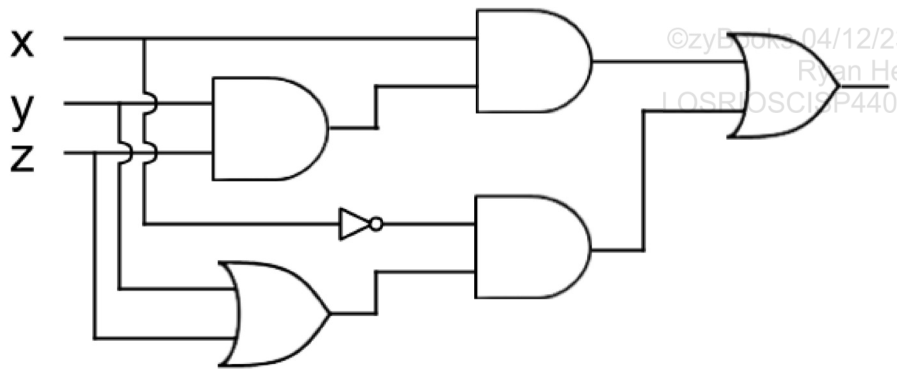
expression before drawing the circuit. Alternatively, you can reason about the function in order to derive a simple Boolean expression that expresses the function.

(a) If x = 0, the circuit will output y + z. If x = 1, the circuit will output yz.

Solution Visible to students

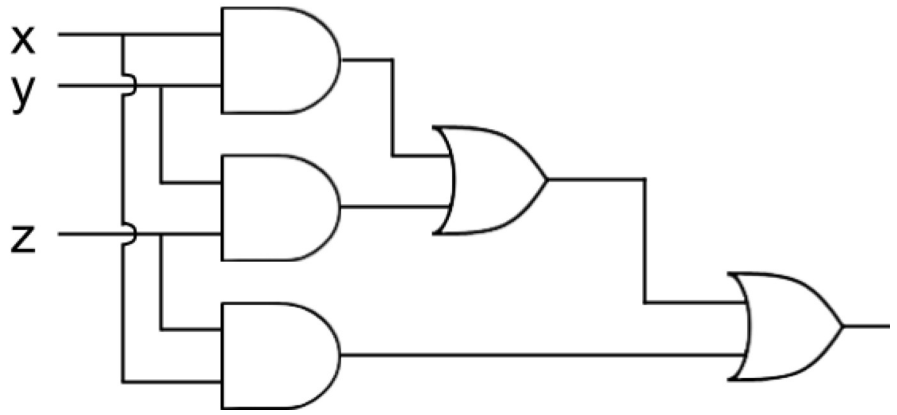
$f(x, y, z) = \overline{x}(y + z) + xyz$



(b) The output is 1 if at least two of the three inputs are 1.

Solution Visible to students

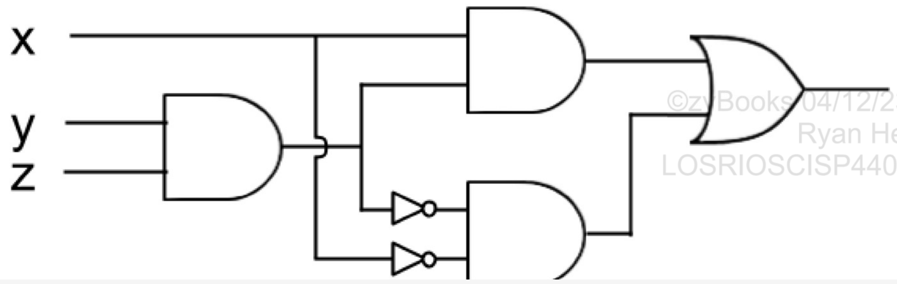
$f(x, y, z) = xy + yz + xz$



(c) The output is 1 if the value of x is equal to the product of y and z.

Solution Visible to students

$f(x, y, z) = xyz + \overline{x}\cdot\overline{(yz)}$



5.7 Boolean SAT Course Scheduler

In this assignment we will write code to explore the course scheduling problem in section 5.5. We will implement a truth table for all possible course schedules and identify which rows satisfy the requirements given in that section.

The table will be a 2D array of booleans:

```
bool table[1024][10];
```

Since there are 10 variables, there are 2^10 = 1024 different rows to the truth table. Each column represents a course, A - E, in either period 1 or 2. Index 1 maps to xA1, index 2 maps to xA2, etc., and index 9 maps to xE2, in the same order as it appears on the table in section 5.5.

The first task is to generate the truth table. It is not practical to manually assign 1024 x 10 truth values, so you must find a way to automate the process. A nice way to solve this problem is to alternate from true to false every time a counter is reached. In the first column that counter is 512, where each row less than 512 will be true until row 512 where the rest of the rows are false. In the next column, alternate every time the counter reaches 256. By the time you get to column 9, the last column, the counter will be 1, where it will alternate from true to false every row. You may want to write a function that prints the table and start out with smaller tables and work your way up to make sure the algorithm is working. The truth table should start the first row with all true and end the last row with all false, as we have done in this book.

The truth table will be unit tested with the following function header:

```
void initTable(bool table[1024][10])
```

Next, write a function that determines if a given row is satisfiable using the conditions specified in section 5.5:

```
bool isRowSat(bool row[10])
```

You only need to pass in one row at a time, so it will be called as `isRowSat(table[i])`. It will return true if that row's truth values meet the specified conditions. This function will also be unit tested.

Finally, implement `main`.

- Declare the `table`
- Call `initTable` on it to fill out its truth values
- Loop through each row.
- If `isRowSat` returns true, print each column of that row as T or F, separated by spaces.

It is interesting to note that there are only two successful configurations out of all 1024 possible schedules. One of them is listed in section 5.5 and the other will be discovered by our algorithm.

Note: There is no starter code for this project, so as long as you meet the above requirements you can code it how you want.

178542.377452.qx3zqy7