KIT103/KMA155 Practical 05: Logic & recursion

This week you'll practice the application of Karnaugh maps to simplify Boolean expressions and, as a bonus (read: optional) exercise, implement a recursive function by using inductive reasoning.

Preparation

In the Labs section of MyLO download the file **kmaps.py**. It contains just two functions: `plot_kmap` is a demonstration visualisation of a K-map, while `broken_if` implements a complex logical operation that is in dire need of simplification.

Refer to the Apps 04 (week 5) lecture slides for the process of creating and using a K-map. Your tutor will work through some of the sample problems below on the board.

1 K-maps for simplifying Boolean expressions

There are three Boolean expressions for you to simplify:

1. A three-variable expression defined by a truth table.
2. A four-variable problem of controlling a train running on a dedicated track, also described by a truth table.
3. The third requires you to inspect some existing code, construct the truth table and K-map, simplify it and then rewrite the code.

1.1 A three-variable warm up

Here is the truth table for three-variable Boolean expression.

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Task: First, write down the naive translation of this truth table into a Boolean expression. Then, on paper (or in Excel) transfer the values to a K-map, identify the groups and simplify the expression.

1.2 Example visualisation

Visualisation is a key part of most scientific computing. Although visualising a K-map is a bit of overkill (the patterns are easy enough to see on paper), it's a useful demonstration of a kind of visualisation that is used with other, more complex problems. The kmaps.py script includes a function `plot_kmap()` which accepts a list of lists (representing the K-map table) and another parameter for customising the axis labels (to suit problems with fewer than four variables or different variable names).

Task: Apply `plot_kmap()` to the sample K-maps in kmaps.py. `sample_kmap3` is an encoding of the three-variable problem above.

1.3 Train yard control

A railway yard operator controls a small train that runs back and forth on a dedicated track. On her control panel she has two buttons, (M)ove and (O)verride, and two indicator lights, (L)eft and (R)ight. The train will move when she holds down the Move button, but not if the train is at one of the two extremes, Left or Right. But she can force it to go past these normal limits if she also holds down the Override button. The logic that decides if the train will actually move is summarised in the truth table below.

M	L	R	O	Will move
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0

M	L	R	O	Will move
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Task: Write down a naive translation of this expression that has five *and* expressions join by logical *or*. Then create the K-map for it, identify the groups, and derive the simplified expression.

> **Tip:** You may find the `truth_table` function from last week useful as a way of testing your original and simplified expressions.

1.4 Simplify an if statement

The kmaps.py script includes a function called `broken_if`, which currently has an overly complex if statement that spans many lines. This is difficult to understand and to maintain.

Task: Derive the truth table for the `if` condition and then construct its K-map (or go straight to the K-map). You will find it easier if you use short aliases (such as a, b and c) for each of the boolean variables. Identify the groups, simplify the expression on paper and then rewrite the code to match.

2 (optional) A little recursion

Task: Write a function, called `sum_n_squares` that takes one parameter `n` > 0 and returns the result of $\sum_{i=1}^{n} i^2$. This function could be done with a loop, but your task is to write it recursively. Start by identifying the base case, then work out the recursive step, then implement it.