The Futurama Theorem

Ben Skeggs-Thirkettle

12th December 2014

1 Introduction

It must be said, Mathematicians are clearly the funniest type of people. The Simpsons, one of TV's longest running sitcoms, has numerous writers with qualifications in mathematics, such as Al jean, J. Stewart Burns and Ken Keeler, (It must also be mentioned that writers David S. Cohen and Jeff Westbrook have qualifications in Computer Science). So as you can imagine, the humor present in The Simpsons, and its sister show Futurama, is quite sophisticated. Some of it's subtle, like Homer accidentally disproving Fermat's Final Theorem, or Bender's Serial number being 1729. In this essay, I will be looking at one of Ken Keeler's theorems shown in the particular episode of Futurama "The Prisoner of Benda" and showing it work through computer coding.

2 The Source Material

In this episode, Professor Farnsworth and Amy invent a machine that is capable of switching the minds of two people that sit in it. So because Amy wants to eat more and not worry about her weight, while The Proffessor wants to be young again, they decide to swap bodies. After enjoying themselves for a while, they decide they want to swap back, only to realise that the machine will never swap minds for two bodies that have already swapped. The episode continues like this, with all the characters swapping bodies, such as Bender using different people to perform a heist. It gets to the point where everybody's mind is in someone else's body, while getting back would be difficult as bodies can't be used twice. So how could they fix their situation? I'll explain in due course...

3 Building My Mind Swapping Machine

```
previouslyswitched = []

class Character():

"""
This class is or my characters, with the attributes Name and Brain

"""

    def __init__(self, name, brain):
        self.name = name
        self.brain = brain

    def msm(p1, p2):

        if (p1.name, p2.name) in previouslyswitched:
            print "The machine does nothing"

        elif p1 == p2:
            return

        else:
            p1.brain, p2.brain = p2.brain, p1.brain
            previouslyswitched.append((p1.name, p2.name))
            previouslyswitched.append((p2.name, p1.name))
```

```
print p1.name + " now has " + p1.brain + "'s brain and " + p2.name + " has " + p2.brain +
"'s brain!"
```

After this is my function "MSM", which stands for Mind Swapping Machine. With the use of the list "previouslyswitched", I can, in essence, save what names have already been swapped and then with the built in "if" function, can get the function to check the list for the names to be already there. I decided that the two names needed to the list, as the two Characters need to be denied even if they go in the wrong order. To get a random set of date, I used the "Random" module included in python, which resulted in the following code:

```
Fry = Character("Fry", "Fry")
Farnsworth = Character("Farnsworth", "Farnsworth")
Leela = Character("Leela", "Leela")
Amy = Character("Amy", "Amy")
Zoidberg = Character("Zoidberg", "Zoidberg")
Hermes = Character("Hermes", "Hermes")
Bender = Character("Bender", "Bender")
vars = [Fry, Farnsworth, Leela, Amy, Zoidberg, Hermes, Bender]

import random

def repeat_test(times):
    for i in range(times):
        random.choice(vars).msm(random.choice(vars))

repeat_test(20)
```

4 The Solution

So now that I've repeated my test and randomly mixed up the minds of my characters, I should now think about I would go about fixing this mess. In the episode, Farnsworth call upon the help of the Harlem Globetrotters, who all have surprisingly keen scientific minds. However, to fix my problem, I only have to look at the theorem created by Ken Keeler. Keeler starts by letting M depict some k-cycle on [n] = 1, 2 ,3,... n:

```
M = 1  2  ...  k  k+1  ...  n
    2  3  ...  1  k+1  ...  n
```

All these numbers essentially represent the bodies of people. Next, he let ¡a, b¿ represent the action of the switching of minds. By hypothesis, M is generated by distinct switches on [n]. He then introduced two new bodies:

```
 M* = 1  2  ...  k  k+1  ...  n  x  y
      2  3  ...  1  k+1  ...  n  x  y
```

For any i from 1 to k, let S be the series of switches left to right:

```
S = (<x,1> <x,2> ... <x,i>) (<y,i+1> <y,i+2> ... <y,k>) (<x,i+1>) (<y,1>)
```

So we can see that we are switching the minds of the bodies with our two new bodies, so we don't run into the problem of the machine not swapping two bodies twice. So by routine verfication:

```
M* S = 1  2  ...  n  x  y
       1  2  ...  n  y  x
```

As shown here, the helper bodies x and y don't have the correct bodies, but that's ok, as during S he didn't perform the switch ¡x, y¿, so we can just switch them. This finally results in all bodies having the correct mind inside them. With this proof, Ken Keeler showed us that with any amount of minds muddled up, it only takes an extra two people x and y to fix it.

5 Fixing My Own Mess

Upon running the programme, Python returns this:

```
Leela now has Farnsworth's brain and Farnsworth has Leela's brain!
Farnsworth now has Zoidberg's brain and Zoidberg has Leela's brain!
Fry now has Leela's brain and Zoidberg has Fry's brain!
Hermes now has Zoidberg's brain and Farnsworth has Hermes's brain!
Bender now has Fry's brain and Zoidberg has Bender's brain!
Leela now has Zoidberg's brain and Hermes has Farnsworth's brain!
Leela now has Amy's brain and Amy has Zoidberg's brain!
The machine does nothing
Leela now has Bender's brain and Zoidberg has Amy's brain!
Farnsworth now has Fry's brain and Bender has Hermes's brain!
Hermes now has Hermes's brain and Bender has Farnsworth's brain!
Hermes now has Leela's brain and Fry has Hermes's brain!
The machine does nothing
The machine does nothing
Bender now has Zoidberg's brain and Amy has Farnsworth's brain!
Fry now has Zoidberg's brain and Bender has Hermes's brain!
Bender now has Bender's brain and Leela has Hermes's brain!

['Zoidberg', 'Bender', 'Amy', 'Farnsworth', 'Leela', 'Hermes', 'Fry']
['Amy', 'Bender', 'Farnsworth', 'Fry', 'Hermes', 'Leela', 'Zoidberg']
```

As you can see, at the bottom, I've also got the programme to print two list, similar to the one seen in Keeler's proof. The top list shows a list of the characters now they've all had their minds swapped, the list below is the mind that is currently in their body. From looking at the two lists, we can see that there are two loops present, so now I add my two helper bodies, Sweet Clyde and Bubblegum, and put my loops into separate groups to sort out.

```
Sweet_Clyde = Character("Sweet", "Sweet")
Bubblegum = Character("Bubblegum", "Bubblegum")

group1 = [Zoidberg, Amy, Farnsworth, Fry]
group2 = [Leela, Hermes]
```

Luckily for him, Bender managed to get his mind back in the process, so I don't have to worry about him. The next thing to do is to swap Bubblegum's mind with the last person in group 1, as shown with (1). Next, Sweet Clyde moves along the group, swapping minds with each person, effectively moving the minds across to where they need to be, as shown with (2). Finally, Bubblegum swaps his mind with the first person, which results in everybody in group 1 having the correct mind ((3)).

```
Bubblegum.msm(group1[-1])                    (1)
for j in group1:                             (2)
        Sweet_Clyde.msm(j)
Bubblegum.msm(group1[0])                     (3)
```

Now at this stage, Bubblegum and Sweetclyde have eachother's mind. However, because the process needs to be repeated with group 2, it results in those two getting their bodies back anyway, without them having to swap with eachother at the end.

6 To Conclude

It has to be said, when I first looked at Keeler's proof, the whole thing did seem incredibly complicated. But after coding it and using it in "real life", it is pretty straight forward. It can be boiled down to the idea putting your list of people in an order such that they are next to the body they want to be in, and then just moving the minds along one by one. It is still impressive that this can be done for an infinite amount of people; it worked for the two different sized groups in my code. If I were to create more instances of Characters, then it's been made certainly clear through the course of this essay that however many more Characters use my MSM function, my two helper Characters Sweet Clyde and Bubblegum will always be able to fix things.

7 References

[1] Futurama theorem - The Infosphere $http://theinfosphere.org/Futurama_theorem$ (Online; Accessed 5th December 2014)