Computer and Information Sciences
University of Strathclyde

Week 2 – The MIU System, part 1 (worth 5% of your overall grade)

Aim

This week, you will start the search project, which will continue for 3 weeks.

The MIU System

The MIU Puzzle is a formal system developed and discussed in Hofstadter, Douglas R. (1979) Gödel, Escher, Bach: An Eternal Golden Braid, Chapter 1: The MU-Puzzle. It is an example of a formal system, using symbols to represent maths and logic. It's a game, and in this project, you will implement it. The aim of this practical is to make the first steps towards developing an algorithm that will solve these problems

- The MIU system is presented as a puzzle: can you transform the input string "MI" into the string "MU"? In order to accomplish this, you are given the following rules:

- Rule 1: If your string ends in an I, you can add a U on the end: x I → x IU
- Rule 2: If the string starts with M, you can double what comes after the M: M x → M x x
- Rule 3: If you have III in a string, you can replace it with a U: x III y → x U y
- Rule 4: If you have UU in a string, you can delete it altogether: x UU y → x y

- It turns out that transforming MI into MU is impossible, but it is possible to derive lots
- of other strings!

String Derivation Example

- Given the string MI to use as the input, we can produce

  MI (Axiom)
  → MII (Rule 2)
  → MIIII (Rule 2)
  → MIIIIU (Rule 1)
  → MIUU (Rule 3)
  → MIUUIUU (Rule 2)
  → MIUUI (Rule 4)

- There are six steps in this derivation - is this the shortest possible derivation of this string? If you've not seen this puzzle before (or even if you have) try to write down a shorter derivation for MIUUI.
- If you are not sure how to get this, then use a pen and paper to identify all the possible states from MI by applying the rules. For example, ("MI") → ["MIU","MII"]. Based on the outputs from rule 1 and rule 2, rules 3 and 4 are not valid here.

Computer and Information Sciences
University of Strathclyde

The next_states(s) function

- `def next_states(s)`

- Given a string, s, as input, return a list of all possible strings that can be derived from s in a single step.

- `1. next-states("MI") → ["MIU","MII"]`
- `2. next-states("MIU") → ["MIUIU"]`
- `3. next-states("MUI") → ["MUIU","MUIUI"]`
- `4. next-states("MIIII") → ["MIIIIU","MIIIIIIII","MUI","MIU"]`
- `5. next-states("MUUII") → ["MUUIIU","MUUIIUUII","MII"]`
- `6. next-states("MUUUI") → ["MUUUIU","MUUUIUUUI","MUI"]`

- the rules must be applied in the order that they are given above

- if there is more than one way to apply a rule, the results must be ordered so that the application is done in a left-to-right order (see example 4)

- the resulting list must contain no duplicates: if a string is already in the result from an earlier application of a rule, don't add it again (see example 6)

- Write this function and then test it with the examples above (and others).

Some Python hints

- `def next_states(s)`

- Note what you should have done in week 1 about creating empty lists and converting strings to a list (`input_list = list(s)`)

- You can use `list.append(s)` to add items to a list

- You can create additional functions, as long as the primary function called is still `next_states(s)`

- You can join a list back together into a string using str = '#'.join(list), which will join a string together into a list using the # as a separator

- Ask the lab demonstrators for help, its what they are there for!

Submission Instructions

- This practical is worth 5% of the mark for the class.

- Please submit your completed `def next_states(s)` function to the online quiz, which will become visible during week 3. You have until the end of week 3 to complete it.

Modified January 2024
By Dr Andrew Abel, based on Dr Clemens Kupke

- Make sure you develop and test your code before attempting the quiz. Once you have started the quiz, you will be able to test your code, but remember that after the first check, you will be deducted marks.

- In the second part of this practical, we will use the next_states(s) function to implement various search methods, such as depth-first and breadth-first search.