CS814 AI for Autonomous Systems, 2019

Assignment 1, Part 1 - to be started in Week 2 (week beginning Tuesday 1st October)

The idea of this practical exercise is to work towards building an algorithm which can search for strings of the MIU system presented in Chapter 1 of "Gödel, Escher, Bach".

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

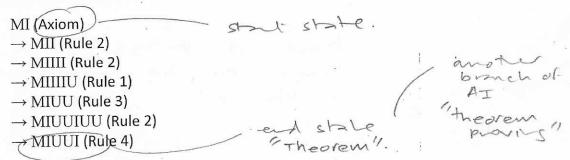
Rule 1: If your string ends in an I, you can add a U on the end: $xI \rightarrow xIU$

Rule 2: If the string starts with M, you can double what comes after the M: $Mx \rightarrow Mxx$

Rule 3: If you have III in a string, you can replace it with a $U:xIIIy \to xUy$

Rule 4: If you have UU in a string, you can delete it altogether: $xUUy \rightarrow xy$

It turns out that transforming $\boldsymbol{M}\boldsymbol{I}$ into $\boldsymbol{M}\boldsymbol{U}$ is impossible, but it is possible to derive lots of other strings, such as MIUUI:



(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.)

Since producing these derivations by hand can be tedious, we want to write a search program (in Python) to do the job for us. The overall aim is to start with $\overline{\mathrm{MI}}$ and use the rules provided to do a tree search for the string we want to produce.

Part 1: 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.

For example:

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

CS814 AI for Autonomous Systems, 2019

Assignment 1, Part 1 - to be started in Week 2 (week beginning Tuesday 1st October)

The idea of this practical exercise is to work towards building an algorithm which can search for strings of the MIU system presented in Chapter 1 of "Gödel, Escher, Bach".

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

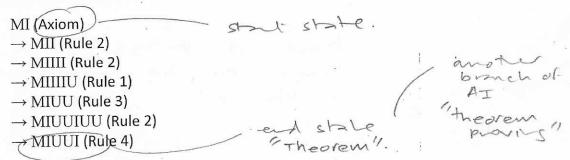
Rule 1: If your string ends in an I, you can add a U on the end: $xI \rightarrow xIU$

Rule 2: If the string starts with M, you can double what comes after the M: $Mx \rightarrow Mxx$

Rule 3: If you have III in a string, you can replace it with a $U:xIIIy \to xUy$

Rule 4: If you have UU in a string, you can delete it altogether: $xUUy \rightarrow xy$

It turns out that transforming $\boldsymbol{M}\boldsymbol{I}$ into $\boldsymbol{M}\boldsymbol{U}$ is impossible, but it is possible to derive lots of other strings, such as MIUUI:



(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.)

Since producing these derivations by hand can be tedious, we want to write a search program (in Python) to do the job for us. The overall aim is to start with $\overline{\mathrm{MI}}$ and use the rules provided to do a tree search for the string we want to produce.

Part 1: 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.

For example:

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

Note that 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).

This part of the assignment is worth 3% of the mark for the class.

To get the credit, you need to get the practical signed off by me. When your program is working correctly, please email your code to John.Levine@strath.ac.uk

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

John Levine 1st October 2019

> Number of strings that can be generaled is intimite;

" seuch space is infinite!"

@ see Combe ()

dof soarch for (s):

assumption

is that
string

string

socherable.

mivil

return shortest legal path from

MI to S.

"Mivil"

"Mivil"