C.4 Control systems

This lab extends the previous one by introducing nondeterminism and feedback. In particular, you will modify the virtual pet that you constructed last time so that it behaves nondeterministically. The modification will make it impossible to keep the pet alive by driving it with another state machine in a cascade composition. You will instead have to use a feedback composition.

This scenario is typical of a control problem. The pet is a system to be controlled, with the objective of keeping it alive. You will construct a controller that observes the output of the virtual pet, and based on that output, constructs an appropriate input that will keep the pet alive. Since this controller observes the output of the pet, and provides input to the pet, it is called a **closed-loop controller**.

C.4.1 Background

Nondeterministic state machines have a *possibleUpdates* function rather than an *update* function. The *possibleUpdates* function returns a set of possible updates. You will construct this function to return a cell array, which was explored in the previous lab.

A software implementation of a nondeterministic state machine can randomly choose from among the results returned by *possibleUpdates*. It could conceptually flip coins to determine which result to choose each time. In software, the equivalent of coin flips is obtained through **pseudo-random number generators**. The Matlab function `rand` is just such a pseudo-random number generator. The way that it works is that each time you use it, it gives you a new number (try `help rand`).

For this lab, you will need to be able to use cell arrays in more sophisticated ways than in the previous lab. Recall that a cell array is like an ordinary array, except that the elements of the array can be arbitrary Matlab objects, including strings, arrays, or even cell arrays. A cell array can be constructed using curly braces instead of square brackets, as in

```
>> letters = {'a', 'b', 'c', 'd', 'e'};
>> whos letters
  Name          Size         Bytes  Class

  letters       1x5            470  cell array

Grand total is 10 elements using 470 bytes
```

The elements of the cell array can be accessed like elements of any other array, but there is one subtlety. If you access an element in the usual way, the result is a cell array, which might not be what you expect. For example,

```
>> x = letters(2)

x =
```

```
    'b'

>> whos x
  Name      Size         Bytes  Class

  x         1x1             94  cell array

Grand total is 2 elements using 94 bytes
```

To access the element as a string (or whatever the element happens to be), then use curly braces when indexing the array, as in

```
>> y = letters{2}

y =

b

>> whos y
  Name      Size         Bytes  Class

  y         1x1              2  char array

Grand total is 1 elements using 2 bytes
```

Notice that now the result is a character array rather than a 1×1 cell array.

You can also use curly braces to construct a cell array piece by piece. Here, for example, we construct and display a two-dimensional cell array of strings, and then access one of the elements as a string.

```
>> t{1,1} = 'upper left';
>> t{1,2} = 'upper right';
>> t{2,1} = 'lower left';
>> t{2,2} = 'lower right';
>> t

t =

    'upper left'    'upper right'
    'lower left'    'lower right'

>> t{2,1}

ans =
```

```
lower left
```

You can find out the size of a cell array in the usual way for arrays

```
>> [rows, cols] = size(t)

rows =

     2


cols =

     2
```

You can also extract an entire row or column from the cell array the same way you do it for ordinary arrays, using ':' in place of the index. For example, to get the first row, do

```
t(1,:)

ans =

    'upper left'    'upper right'
```

C.4.2 In-lab section

1. Construct a Matlab function `select` that, given a cell array with one row as an argument, returns a randomly chosen element of the cell array. Use your function to generate a random sequence of 10 letters from the cell array

   ```
   >> letters = {'a', 'b', 'c', 'd', 'e'};
   ```

   Hint: The Matlab function `floor` combined with `rand` might prove useful to get random indexes into the cell array.

2. Construct a Matlab function `chooserow` that, given a cell array with one or more rows, randomly chooses one of the rows and returns it as a cell array. Apply your function a few times to the '`t`' array that we constructed above.

3. A nondeterministic state machine has a *possibleUpdates* function rather than *updates*. This function returns a set of pairs, where each pair is a new state and an output.

A convenient Matlab implementation is a function that returns a two-dimensional cell array, with each of the possible updates on one row. As a first step towards this, modify your realization of the *update* function for the virtual cat of the previous lab so that it returns a 1×2 cell array with the next state and output. Also modify your program that runs the cat (without the driver) so that it uses your new function. Verify that the cat still works properly.

4. Now modify the cat's behavior so that if it is hungry and you feed it, it sometimes gets happy and purrs (as it did before), but it sometimes stays hungry and rubs against your legs. I.e., change your *update* function so that if the state is hungry and you feed the cat, then return a 2×2 cell array where the two rows specify the two possible next state, output pairs. Modify the program that runs the cat to use your `chooserow` function to choose from among the options.

5. Compose your driver machine from the previous lab with your nondeterministic cat, and verify that the driver no longer keeps the cat alive. In fact, no open-loop controller will be able to keep the cat alive and allow time to pass. In the independent section of this lab, you will construct a **closed-loop controller** that keeps the cat alive. It is a feedback composition of state machines.

C.4.3 Independent section

Design a deterministic state machine that you can put into a feedback composition with your nondeterministic cat so that the cat is kept alive and time passes. Give the state transition diagram for your state machine and write a Matlab function that implements its *update* function. Write a Matlab program that implements the feedback composition.

Note that your program that implements the feedback composition faces a challenging problem. When the program starts, neither the inputs to the controller machine nor the inputs to the cat machine are available. So neither machine can react. For your controller machine, you should define Matlab functions for both *update*, which requires a known input, and *output*, which does not. The *output* function, given the current state, returns the output that will be produced by the next reaction, if it is known, or *unknown* if it is not known. In the case of your controller, it should always be known, or the feedback composition will not be well formed.

Verify (by running your program) that the cat does not die.

Instructor Verification Sheet for C.4

Name: _____ Date: _____

1. Generated random sequence of letters using `'select'`.

   Instructor verification: _____

2. Applied `chooserow` to the `'t'` array.

   Instructor verification: _____

3. The cat still works with the update function returning a cell array.

   Instructor verification: _____

4. The nondeterministic sometimes stays hungry when fed.

   Instructor verification: _____

5. The nondeterministic cat dies under open-loop control.

   Instructor verification: _____