Exercise Sheet 09 Solutions – Errors and Finite State Machines

Sebastian Höffner Aline Vilks

Deadline: Mon, 05 June 2017 08:00 +0200

Exercise 1: Making coffee

File: `coffee.py`

```
"""Coffee acceptor.

This module implements a finite state machine, a coffee acceptor.

It has the states A to H and the inputs P, C, F, and B. The transition is
defined as follows:

   P C F B
-- - - - -
A  C A B H
B  D B H H
C  H E D H
D  H F H H
E  H E F H
F  H F H G
G  H G H H
H  H H H H

The module reads a file coffeerecipes.txt and checks which recipes are good and
which ones are bad. Example recipes are:

    PFCFB
    PFCB

The first one, PFCFB, fails, while PFCB is okay.
"""
```

```python
def transition(state, action):
    """Implements the acceptor's transition function.

    Given a state (A, B, ..., H) and an action (P, C, F, B), this function
    determines the next state.

    Args:
        state: The state from which to move.
        action: The action to perform.

    Returns:
        The new state.
    """

    action_map = 'PCFB'
    state_map = 'ABCDEFGH'
    delta = [
        ['C', 'A', 'B', 'H'],
        ['D', 'B', 'H', 'H'],
        ['H', 'E', 'D', 'H'],
        ['H', 'F', 'H', 'H'],
        ['H', 'E', 'F', 'H'],
        ['H', 'F', 'H', 'G'],
        ['H', 'G', 'H', 'H'],
        ['H', 'H', 'H', 'H'],
    ]
    return delta[state_map.index(state)][action_map.index(action)]


def test(recipe):
    """Tests if recipe is accepted by the acceptor.

    Args:
        recipe: The recipe.

    Returns:
        A tuple of length 2. The first value is either True or False, depending
        on whether the recipe was accepted or not. The second value is a string
        representing the visited states.
    """
    states = 'A'
    for action in recipe:
        states += transition(states[-1], action)
    return states[-1] == 'G', states
```

```
def read_recipes(filename):
    """Reads the file and returns its content as a list of strings."""
    with open(filename, 'r') as file_handle:
        return file_handle.read().splitlines()


def main():
    """Tests all coffee recipes and prints the results."""
    recipes = read_recipes('coffeerecipes.txt')

    results = []
    for recipe in recipes:
        results.append(test(recipe))

    for recipe, (result, states) in sorted(zip(recipes, results),
                                           key=lambda x: x[1][0]):
        print('Recipe:  {}'.format(recipe))
        print('States: {}'.format(states))
        print('Result: {}.'.format('Okay' if result else 'Fail'), end='\n\n')


if __name__ == '__main__':
    main()
```

Output:

```
Recipe:  PFCFB
States: ACDFHH
Result: Fail.

Recipe:  FCCPCFCCBC
States: ABBBDFHHHHH
Result: Fail.

Recipe:  PCCCFCCFB
States: ACEEEFFFHH
Result: Fail.

Recipe:  CFPB
States: AABDH
Result: Fail.

Recipe:  PCFB
States: ACEFG
Result: Okay.
```

```
Recipe:  PFCB
States: ACDFG
Result: Okay.

Recipe:  FCCPCCCCB
States: ABBBDFFFFG
Result: Okay.

Recipe:  FPCCB
States: ABDFFG
Result: Okay.

Recipe:  CPCCFB
States: AACEEFG
Result: Okay.

Recipe:  CFPCB
States: AABDFG
Result: Okay.
```