SENG 5811 Homework #2

Date: 2/21/2025
Due: 3/1//2025
Topics: Black-box techniques
Problems: 4
Points: 50

This assignment is to be completed individually.

Consider the example discussed in class about determining overlaps between repeating patterns:

	7+5x	1+3x	6+8x	-2+12x	4+9x	6+4x	? + ?x (ALL ✓)
1		✓					
2							
3							
4		✓			✓		
5							
6			✓			✓	
7	✓	✓					
8							
9							
10		✓		✓		✓	
11							
12	✓						
13		✓			✓		
14			✓			✓	
15							
16		✓					
17	✓						
18						✓	
19		✓					
20							
21							
22	✓	✓	✓	✓	✓	✓	✓

Let us specify the needs in the form of the following requirements for a program (we will call it `crp` for “common repeating pattern”).

We will first define two terms that we use in the requirements that follow.

Definition 1: A **pattern** is a string of the form *A+B*x in which *A* is any integer and *B* is a positive integer, and it denotes an integer-indexed infinite sequence in which ✓ appears exactly at indices *A, A+B, A+2B, A+3B, … .*

Definition 2: Two patterns are said to **overlap** if the sequences denoted by them have a ✓ at some common index *n*. (The patterns are said to *overlap at n* if the index is relevant).

Requirements:

1. The program shall take as input up to 6 patterns as input arguments on the command-line, in which each integer may be up to 3 decimal digits long.

   (*E.g.*, for the above example is run as: **`crp 7+5x 1+3x 6+8x -2+12x 4+9x 6+4x`**)

2. When no input arguments are provided, the program shall produce an appropriate message explaining proper usage of the program.

3. Any input argument that does not conform to the format specified in 1, shall be identified as *malformed* and discarded.

4. Any input argument denoting a pattern that does not overlap with some pattern denoted by a preceding non-discarded input argument shall be identified as *conflicting* and discarded.

5. The program shall produce as output a list of lines such that the k^{th} line in the output shows the k^{th} input argument followed by a separator and then:

   5.1. if it is a discarded input argument, an appropriate error message.

   5.2. if otherwise, a pattern that overlaps with every non-discarded pattern in the first *k* input arguments at every index where a ✓ appears in its denoted sequence.

   E.g., the output for **`crp 7+5x 1+3x 6+8x -2+12x 6+4x`** should be:

```
7+5x => 7+5x
1+3x => 7+15x
6+8x => 22+120x
-2+12x => 22+120x
4+9x => 22+360x
6+4x => 22+360x
```

<u>Use the above requirements to answer the following questions.</u>

1. **14 Points**

As a first step in test design apply input space partitioning to model the input domain of the program.

A suggested way to organize your solution, is as a two-column table of *characteristics* and *partitions*, where each row lists:

- a *characteristic*, which is some aspect of the input domain that you think influences how the program behaves (e.g., number-of-command-line-arguments), and
- its corresponding *partition*, which is a finite set of value classes (*a.k.a.* blocks) for that characteristic (e.g., zero, one, many, too-many).

Note that a valid partitioning of a characteristic should have two properties: the blocks should be non-overlapping and when taken together they must cover the input space.

Provide a brief rationale for why your input domain model is "good" – this could be just an informal argument answering why the characteristics and the partitioning you designed are needed and adequate for testing the program.

2. **12 Points**

From your solution to the previous question, design a set of test cases for the program by answering the following.

a) Identify or briefly describe a strategy that you would use for deciding which combinations of value classes (blocks) are to be considered for creating test cases.

b) Provide three examples of combinations that would be chosen by your strategy, and the corresponding test cases (inputs and expected outputs) for those combinations.

c) Provide a rough estimate of the number of test cases you would need to achieve pairwise coverage of combinations and briefly explain how you arrived at that estimate.

3. **12 Points**

Creating and using a decision table: For this exercise, assume that we have restricted the scope of our testing to consider at most two input patterns (instead of the 6 in requirements).

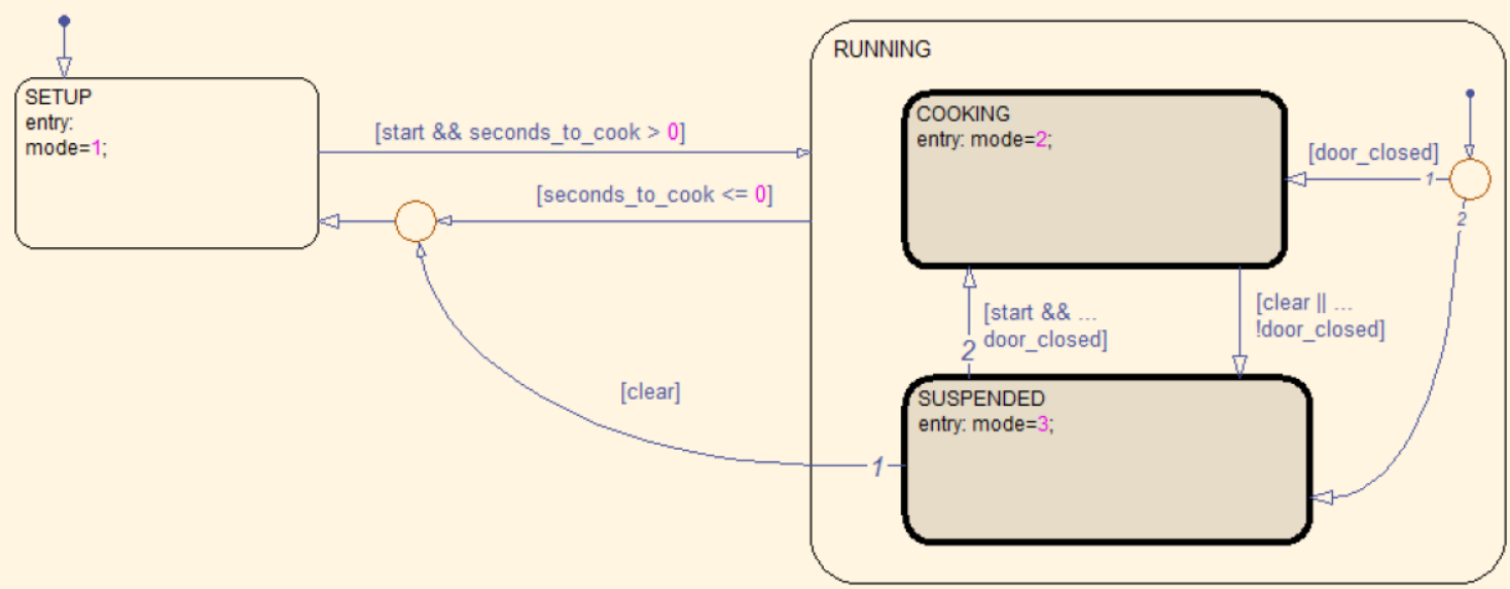
a) Create a decision table that represents the various possible combinations for determining outputs (error message, output pattern, …) for the different possibilities of the input

patterns (invalid, valid, non-overlapping, ….). The decision table must be complete (all possibilities are covered) and consistent (no conflicting actions for the same combination of conditions.).

b) Provide a set of test cases that achieves column coverage for the table. That is, for any given column, there must be a test case in which that column evaluates to true while the rest of the columns evaluate to false.

4. 12 Points

Tests from state machines: Consider the state-diagram below modeling the behavior of a simple microwave oven controller.



The initial state is SETUP. When the compound state RUNNING is entered, the controller will enter either the COOKING or the SUSPENDED state depending on whether the door is closed or not.

Numbers on transitions, if present, indicate priority. When multiple transitions out of a state are simultaneously enabled, the lowest numbered transition is taken. The "*entry*:" actions within each state sets the mode variable when that state is entered. The circles are simply split/join points for better visual representation transitions better.

Recall from class discussion that for a controller such as the one given here, **a test case is a sequence of inputs and the corresponding expected output at each step**.

Inputs: *start*, *clear*, *door_closed* are Boolean, *seconds_to_cook* is Integer.

Output: *mode* is an Integer output.

a) Provide a test case that achieves state coverage for this model but not transition coverage. Show which states are covered at each step of your test case.

b) Provide a test case that achieves transition coverage for this model. Show which transitions are covered at each step.

c) Will a test suite that achieves transition coverage also achieve coverage of all the atomic Boolean conditions on the transition labels?

   Note that an atomic Boolean condition is one which does not have sub-expressions that are Boolean: E.g. In **(x < 5 && b)**, the expressions, **x < 5** and **b** are considered atomic.

   A Boolean condition is considered "covered" by a test suite if it evaluates to True at least once and to False at least once, when the whole test suite is executed.

d) Suppose we have a buggy implementation of the controller that inverts the priority order on the transitions out the SUSPENDED state. Provide a test case that would detect this bug. State how the bug would be detected.