Problem Overview

To reach the ultimate goal, which is to automate the test case, we need to **define the problem and setup the test case properly**.

The function that will be tested as follows:
1. Minus
2. Division
3. CE

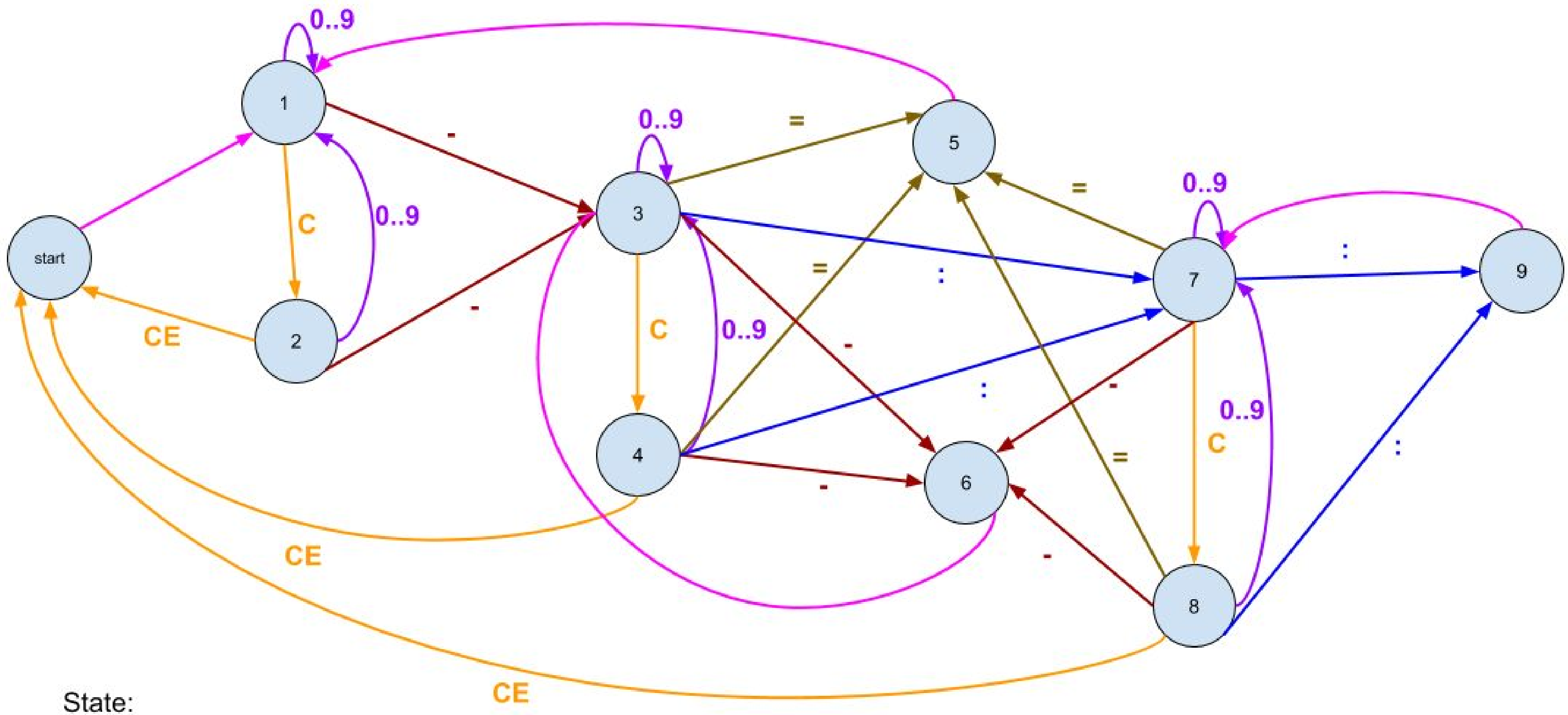
I will assume that I only use “=” to help me validate the result of the operation, but I will not test “=” as a function itself. I also won’t test “.” operation, so all number tested here will be integer, for the sole purpose of validating the required operation.

To make the problem easier to understand, I will define it as a finite state machine. I need to create 2 finite state machine, because of If the operation as follows:

$$x - y / z / a / b / c$$

The division will be processed first. This is why I need to divide into two finite state machine, one represent the case where minus is the first operation, and the other will represent the case where division is the first operation.

A. SUBTRACTION FIRST



State:

1: input **(first number)** on the operation

2: after input **(first number)** then **(CE button)** pressed, the **(first number)** on the operation will become 0

3: input **(second number)** on the operation

4: when input **(second number)** then **(CE button)** pressed, the **(second number)** on the operation will become 0

5: when **(equal button)** pressed, show result of **(first number)** - **(second number)**, and go back to state 1

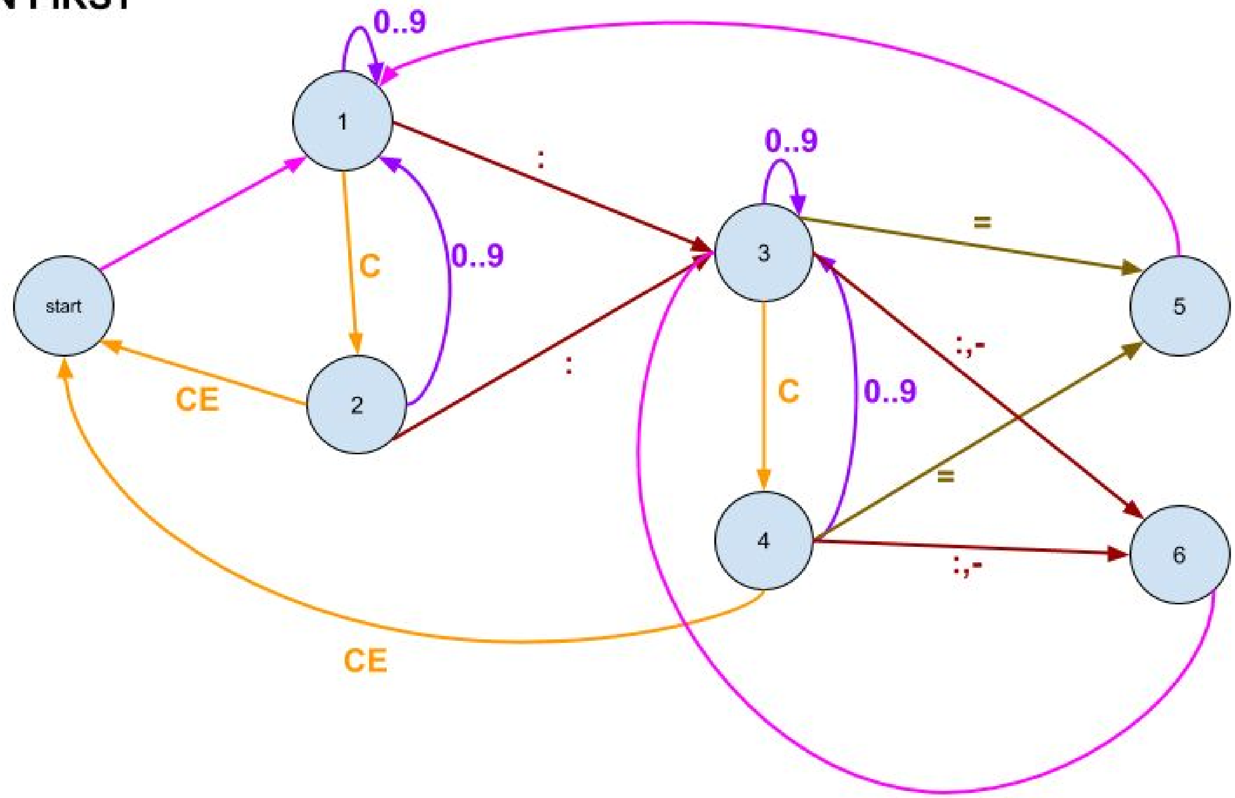
6: when **(minus button)** pressed, show result of **(first number)** - **(second number)**, treat the result as **(first number)**, and go to state 3 to input the **(second number)**

7: if the condition is **(first number)** - **(second number)** : **(third number)**, this is the state of third number inputted, with **(second number)** : **(third number)** will be processed before the minus. If (**equal button)** is pressed, will show the result, if **(minus button)** is pressed, will show the result and goes back to state 3.

8: after input **(third number)** then **(CE button)** pressed, the **(third number)** on the operation will become 0

9: if **(division button)** is pressed, will show the result and goes back to state 7, where the division will be processed first

B. DIVISION FIRST



State:
1: input **(first number)** on the operation
2: after input **(first number)** then **(CE button)** pressed, the **(first number)** on the operation will become 0
3: input **(second number)** on the operation after **(division button)** is pressed
4: when input **(second number)** then **(CE button)** pressed, the **(second number)** on the operation will become 0
5: when **(equal button)** pressed, show result of **(first number)** / **(second number)**, and go back to state 1
6: when **(minus button)** or **(division button)** pressed, show result of **(first number) (operation) (second number)**, treat the result as **(first number)**, and go to state 3 to input the **(second number)**

Couple things to note before we define the script:

Operator	Things to Note
Minus	Need to test boundary between positive and negative, the test case will involve <0, 0, >0
Division	Need to test division that return fraction Need to test zero division, any number divided by zero should return NaN (this will be our negative case)
CE	The display will become C if any current state is happening

Test Script

Given all the consideration on the premises, the test script will be as follows:

No.	Description	Precondition	Steps	Expected
Minus/Subtraction operation				
1	**FSM: A** **State: 1 -> 3 -> 6 -> 3 -> 4 -> 6** Cover: 1. all minus operation on state A1,A3,A4 2. Positive and negative boundary (<0, >0) 3. As a bonus (i) cover button CE	On clear state (state 1)	Press 9	Display is 9 CE become C
			Press (-)	Display still 9 CE still C (-) button color yellow
			Press 3	Display is 3 CE still C
			Press (-)	Display is 6,

	behavior on state 3 (ii) validate that the number cleared by CE is no longer affect the state			CE still C (-) button color yellow
			Press C	Display is 0, CE become CE
			Press 10	CE become C
			Press (-)	Display is -4 CE still C (-) button color yellow
2	**FSM: A** **State: 1 -> 2 -> 3 -> 5** Cover: 1. Minus operation on state A2 2. As a bonus (i) cover CE behavior on state 1 (ii) validate that the number cleared by CE is no longer affect the state	On clear state (state 1)	Press 9	Display 9 CE become C
			Press C	Display is 0 C become CE
			Press (-)	Display 0 CE become C (-) button color yellow
			Press 9	Display 9 CE still C
			Press (=)	Display -9 CE still C
3	**FSM: A** **State: 1 -> 2 -> 1 -> 3 -> 5** Cover: 1. Minus operation with result 0 2. As a bonus,	On clear state (state 1)	Press 1	Display 1 CE become C
			Press C	Display 0 C become CE (-) button color yellow

	(i) cover CE behavior on state 2 (iii) validate that the next calculation done have a fresh state		Press CE	Display 0 CE still CE
			Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C (-) button color yellow
			Press 9	Display 9 CE still C
			Press (=)	Display 0 CE still C
4	**FSM: A** **State: 1 -> 3 -> 7 -> 8 -> 1 -> 3 -> 7 -> 6** Cover: 1. Minus operation state A7 2. As a bonus (i) cover CE behavior on state 8 (ii) validate that the next calculation done have a fresh state	On clear state (state 1)	Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C (-) button color yellow
			Press 3	Display 3 CE still C
			Press (:)	Display 3 CE still C (:) button color yellow
			Press CE	Display 0 C become CE
			Press C	Display 0 CE still CE

			Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C (-) button color yellow
			Press 6	Display 6 CE still C
			Press (:)	Display 6 CE still C (:) button color yellow
			Press 3	Display 3 CE still C
			Press (-)	Display 7 CE still C (-) button color yellow
5	**FSM: B** **State: 1 -> 3 -> 6** Cover: 1. Minus operation on state B3 (note: state B4 is covered by division operation)	On clear state (state 1)	Press 8	Display 8 CE become C
			Press (:)	Display 8 CE still C (:) button color yellow
			Press 4	Display 4 CE still C
			Press (-)	Display 2 CE still C (-) button color yellow

6	**FSM: A** **State: 1 -> 3 -> 7 -> 8 -> 6** Cover: 1. Minus operation on state A8 2. As a bonus (i) cover CE behavior on state 7 (ii) validate that the number cleared by CE is no longer affect the state	On clear state (state 1)	Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C (-) button color yellow
			Press 6	Display 6 CE still C
			Press (:)	Display 6 CE still C (:) button color yellow
			Press 3	Display 3 CE still C
			Press C	Display 0 C become CE
			Press (-)	Display “Error” CE become C (-) button color yellow
Division operation				
7	**FSM: B** **State: 1 -> 3 -> 6 -> 3 -> 4 -> 3 -> 6** Cover: 1. all division operation on state B1,B3 2. Division that return integer and fraction	On clear state (state 1)	Press 12	Display is 12 CE become C
			Press (:)	Display still 12 CE still C (:) button color yellow
			Press 3	Display is 3 CE still C

	3. As a bonus (i) cover button CE behavior on state 3 (ii) validate that the number cleared by CE is no longer affect the state		Press (:)	Display is 4, CE still C (:) button color yellow
			Press C	Display is 0, CE become CE
			Press 3	Display is 3, CE become C
			Press (:)	Display is 1.3333333 CE still C (:) button color yellow
8	**FSM: B** **State: 1 -> 2 -> 3 -> 5** Cover: 1. Division operation on state B2 2. As a bonus, cover CE behavior on state 1, and validate that the number cleared by CE is no longer affect the state	On clear state (state 1)	Press 9	Display 9 CE become C
			Press C	Display is 0 C become CE
			Press (:)	Display 0 CE become C (:) button color yellow
			Press 9	Display 9 CE still C
			Press =	Display 0 CE still C
9	**FSM: A** **State: 1 -> 3 -> 7 -> 9 -> 7 -> 9 -> 7 -> 5** Cover:	On clear state (state 1)	Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C

	1. Division operation on state A7 2. Make sure if division operation kept being used, it will be looped in state A7 and A9 and the end subtraction result is correct			(-) button color yellow
			Press 8	Display 8 CE still C
			Press (:)	Display 8 CE still C (:) button color yellow
			Press 2	Display 2 CE still C
			Press (:)	Display 4 CE still C (:) button color yellow
			Press 2	Display 2 CE still C
			Press (=)	Display 7 CE still C
10	**FSM: B** **State: 1 -> 3 -> 4 -> 1 -> 3 -> 4 -> 6** Cover: 1. Division operation with 0 (state B4) as second operator 2. As a bonus, (i) cover CE behavior on state 4 (ii) validate that the next	On clear state (state 1)	Press 9	Display 9 CE become C
			Press (:)	Display 9 CE become C (:) button color yellow
			Press C	Display 0 C become CE (:) button color yellow
			Press CE	Display 0 CE still CE

	calculation done have a fresh state (iii) cover minus behavior on state B4		Press 9	Display 9 CE become C
			Press (:)	Display 9 CE still C (:) button color yellow
			Press C	Display 0 C become CE
			Press (-)	Display “Error” CE still CE
11	**FSM: A** **State: 1 -> 3 -> 7 -> 8 -> 9** Cover: 1. Division operation with 0 (state A8) as third operator 2. As a bonus: (i) cover CE behavior on state 7 (ii) validate that the number cleared by CE is no longer affect the state	On clear state (state 1)	Press 9	Display 9 CE become C
			Press (-)	Display 9 CE still C (-) button color yellow
			Press 8	Display 8 CE still C
			Press (:)	Display 8 CE still C (:) button color yellow
			Press 2	Display 2 CE still C
			Press C	Display 0 CE still C
			Press (:)	Display “Error” C become CE

				(:) button color yellow
CE operation				
**Note: for CE operation, all state behavior has been validated by the prior test cases (I assume CE behavior is same in FSM A and B), so there's no test case for this. As of why I include it in prior test case (subtraction and division), is because to test CE functionality, we need to validate that after we press CE, it will be a fresh start and following calculation is not affected by last calculation cleared anymore, therefore it's more efficient to include it inside the operation test case, because it will validate it in parallel with testing the operation functionality.*				
Other Test Case				
12	Test Max Digit	On clear state (state 1)	Press 9999999999 (10 of 9)	Value should be 999 999 998 (latest digit inputted is not included, because max digit is 9)
			Press (-)	
			Press 1	
			Press (=)	
13	Test Max Digit	On clear state (state 1)	Press 9999999999 (10 of 9)	Value should be 3003003 (latest digit inputted is not included, because max digit is 9)
			Press (:)	
			Press 333	
			Press (=)	

🟩: Positive test case
🟥: Negative test case

Implementation

There’s 2 technical issues here:

1. The calculator is on canvas
2. The result display cannot be extracted easily

To overcome problem 1, for the calculator button, I’m using coordinate to map each button, relatively to the canvas starting coordinate.

To overcome problem 2, I’m using Tesseract library to do OCR on the result, and do the validation.

Error are caught and will be logged on the report. Screenshot will be automatically capture if error happened based on default cucumber report setting.

If I have more time, I will clean some of the code, some are still hard coded and it’s quite messy.

Afternote

Due to the limited time constraint:

1. I won’t handle trivial requirement such as CE will clear yellow color mark on pressed operation, etc
2. Even though I put detailed expected results for every button pressed, I won’t check all of that in automation. This is assuming that the result justify all the prior calculation, therefore if end result is correct, all the process must be correct. There’s a risk on this assumption but I feel it’s very low and the efficiency of the automation will make up for the risk taken.