RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC211A
Course Name	Formal Languages and Automata Theory
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No	17ETCS002159
Semester/Year	04/2019
Course Leader/s	P. Padma Priya Dharishini

:

Declaration Sheet				
Student Name	Satyajit Ghana			
Reg. No	17ETCS002159			
Programme	B.Tech		Semester/Year	04/2019
Course Code	CSC211A			
Course Title	Formal Languages and Automata Theory			
Course Date		to		
Course Leader	P. Pamda Priya Dharishini			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Figure No. **Title of the figure** **Pg.No.**

Question No. 1

Solution to Question No. 1 Part A:

A 1.1 Introduction to control characters in games:

Finite state machines lend themselves to representing the behavior of computer-controller characters in video games. The states of the machine correspond to the character's behaviors, which change according to various events. These changes are modeled by transitions in the state diagram. State machines are certainly not the most sophisticated means of implementing artificially intelligent agents in games, but many games include characters with simple, state-based behaviors that are easily and effectively modeled using state machines.

At their simplest form, State Machines are an Observer pattern with 2 values they watch internally: An Entity and the Current State. Whenever someone, somewhere changes the internal state, it'll let the world know.

The high-level parts are:

1. States which contain/define the behavior for the Entity
2. Transitions which occur when one State changes to another
3. Rules which define which States can change to certain other States
4. Events which are internally or externally dispatched which trigger State Transitions

When creating State Machines, you usually know all the states up front, hence the word finite. Whether a list on paper or a bunch of boxes on a flow chart with arrows denoting where you can go from where. Some states can have parent child relationships. There can be a variety of rules that allow/prevent some states from changing to others can causing state transitions.

A 1.2 Discussion on role of FA to represent control characters in games:

Here we consider the classic game, Pac-Man. For those unfamiliar with the gameplay, Pac-Man requires the player to navigate through a maze, eating pellets and avoiding the ghosts who chase him through the maze. Occasionally, Pac-Man can turn the tables on his pursuers by eating a power pellet, which temporarily grants him the power to eat the ghosts. When this occurs, the ghosts' behavior changes, and instead of chasing Pac-Man they try to avoid him.

1. The ghosts in Pac-Man have four behaviors:
2. Randomly wander the maze
3. Chase Pac-Man, when he is within line of sight

4. Flee Pac-Man, after Pac-Man has consumed a power pellet
5. Return to the central base to regenerate

These four behaviors correspond directly to a four-state DFA. Transitions are dictated by the situation in the game. For instance, a ghost DFA in state 2 (Chase Pac-Man) will transition to state 3 (Flee) when Pac-Man consumes a power pellet.

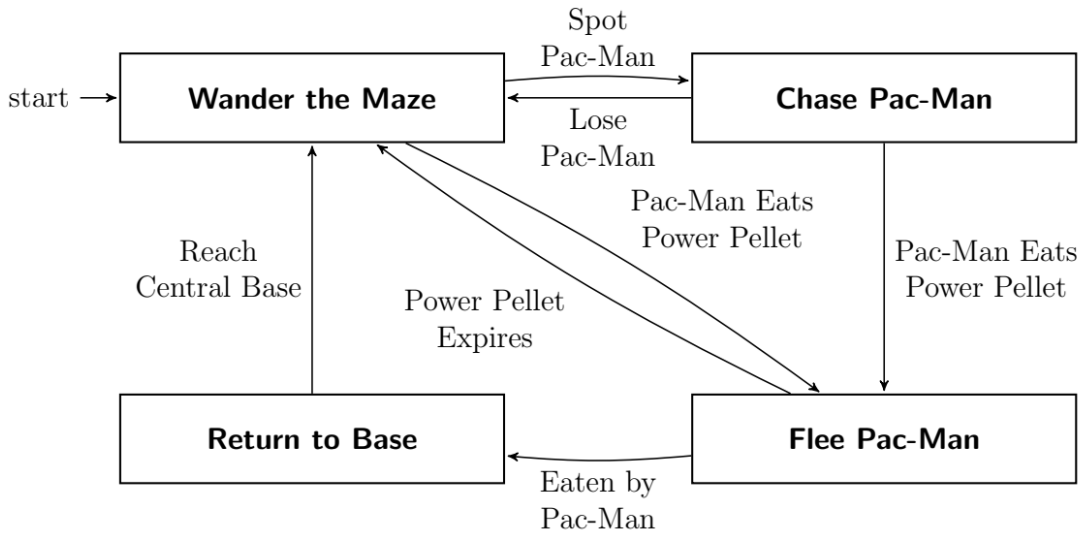


Figure 1 Behaviour of Pacman Ghost

(Eric Gribkoff, 2013)

A 1.3 Conclusion:

Finite-state machines are useful to implement AI logic in games. They can be easily represented using a graph, which allows a developer to see the big picture, tweaking and optimizing the final result.

The implementation of an FSM using functions or methods to represent states is simple, but powerful. Even more complex results can be achieved using a stack-based FSM, which ensures a manageable and concise execution flow without negatively impacting the code. We can use state machines to run different code based on state. We can use state machines to represent arbitrary meshes. These meshes can include things like maps and connectivity grids. We don't need to hard code the state machines. When you load them from data you can get substantially more use out of them.

Using State Machines allows you to more easily scale and organize behavior code for your Game Entities, especially for AI development. In games, they are most known for being used in AI, but they are also common in implementations of user input handling, navigating menu screens, parsing text, network protocols, and other asynchronous behavior.

Question No. 2

Solution to Question No. 1 Part B:

B 1.1 Introduction:

A Salad Vending machine is to be made that has to follow the given constraints,

- It has to display the menu of available fruits and vegetables
- It should allow the user to select the required fruits/vegetables for their salad
- After selecting fruits/vegetables the user has to press the submit button
- Based on the fruits/vegetables selected SVM has to display the price of the salad
- It should dispense the salad after accepting money from the user
- It should not allow the user to select both fruits/vegetables salad at the same time

The problem requires us to make either an NFA/DFA for the vending machine process. Since it's easier to design an NFA (Non-Deterministic Finite State Automata), we would be doing so in this problem.

B 1.2 Problem solving approach:

The Vending machine has to display the menu for the salad items, which is taken as one of the states, that can be triggered from the IDLE state, and then comes back to the IDLE state. Then the required items are selected as per the user's choice, proceeded by the payment for it, upon which the salad is dispensed.

The Payment design part of it has been taken as an inspiration from Moore-Mealy machines, in Mealy machine the output of the function is dependent upon the present state and also the current input, while in the Moore machine the output depends only on the present state.

Here is the approach for it in form of an Algorithm:
Step 1: IDLE
Step 2: Display Menu
Step 3: come back to IDLE
Step 4: START_ORDER
Step 5: Select either a fruit or a vegetable, this will set your salad type, and cannot be changed later, hence it restrict from mixing fruits and vegetables together.

Step 6: Select the second fruit or vegetable

Step 7: Select the third fruit or vegetable

Step 8: Press the Submit button to proceed to Payment.

Step 9: Display the total Amount to pay.

The payment process algorithm:

Step 1: Keep inserting coins of either 10 or 5, until the total amount

Step 2: The salad is dispensed after the payment is done.

Step 3: Return to IDLE state

Approach for Fruits/Vegetable Selection: Every fruit/vegetable has a price to it, when selecting this price is kept track of, since we have chosen 3 fruits and 3 vegetables the total number of combinations for the choice are 14, all these combinations have a combination of price, each of which is a different state, based on the total price, the displayed price is shown to the user.

Approach for Payment: The user can insert 10 and 5 rupees coins, upon insertion of these coins the current deposit amount is kept track using a Moore machine, if the user inserts 5, 5, coins, then the total balance is 10, or the user can insert a 10 rupee coin and go to state with balance 10, as soon as the price of the salad is met, the salad is dispensed to the user, this completes the payment process, and the vending machine then goes back to the IDLE state.

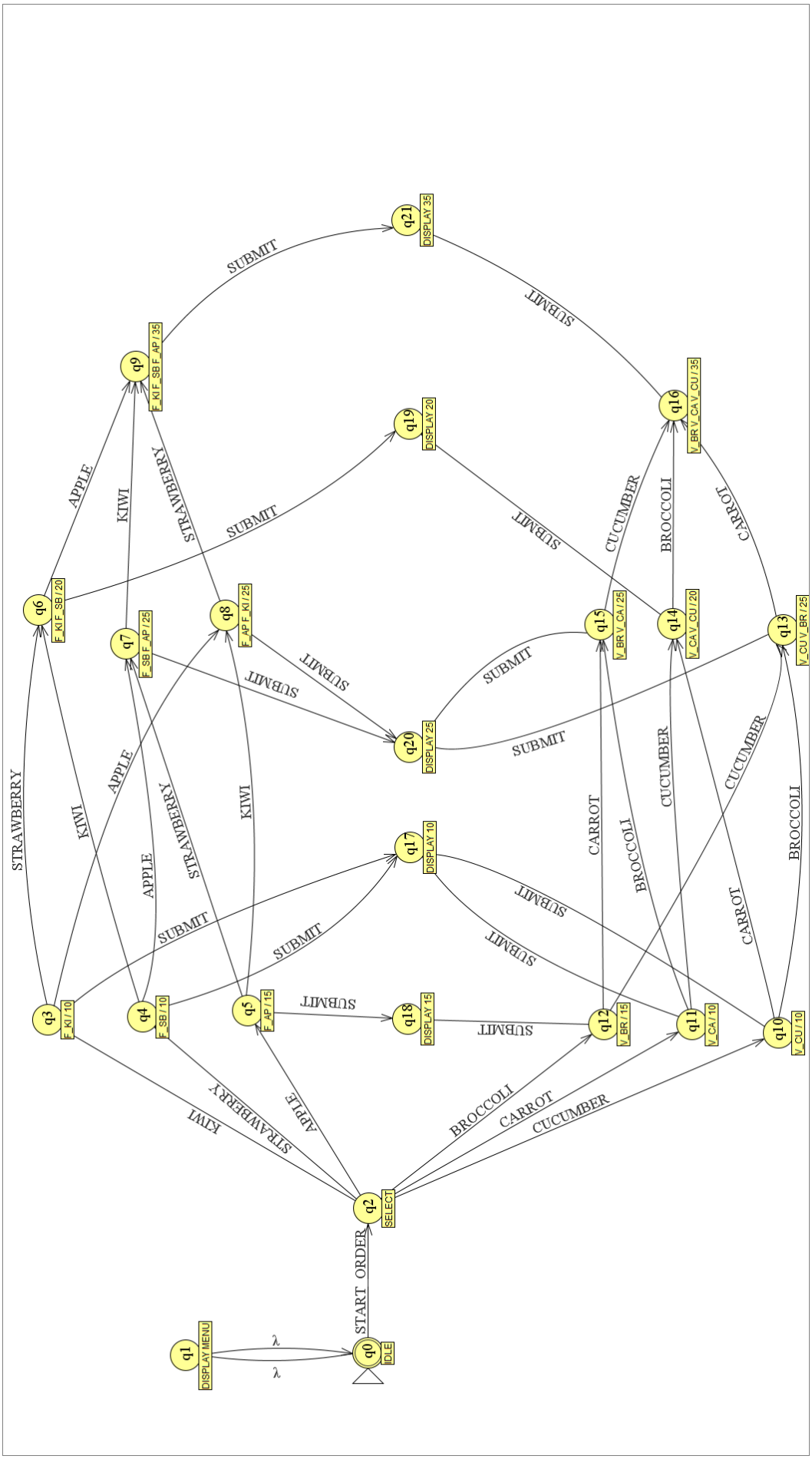
B 1.3 Design and Validation:

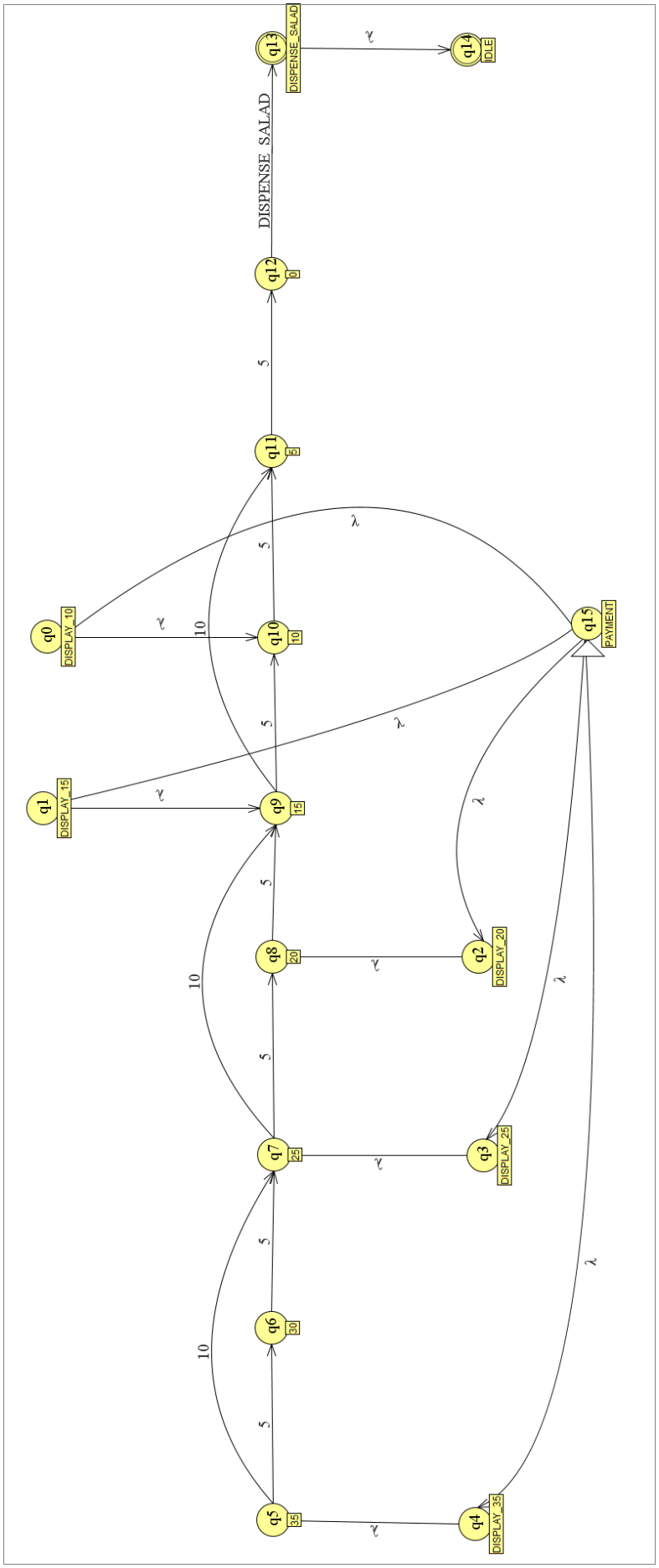
The design for the above approach is done in JFLAP.

The NFA is split into two parts for simplicity, the first part is the displaying of the menu and selection of the fruits or vegetables, then display the price, based on the price the second NFA processes the payment and dispenses the salad.

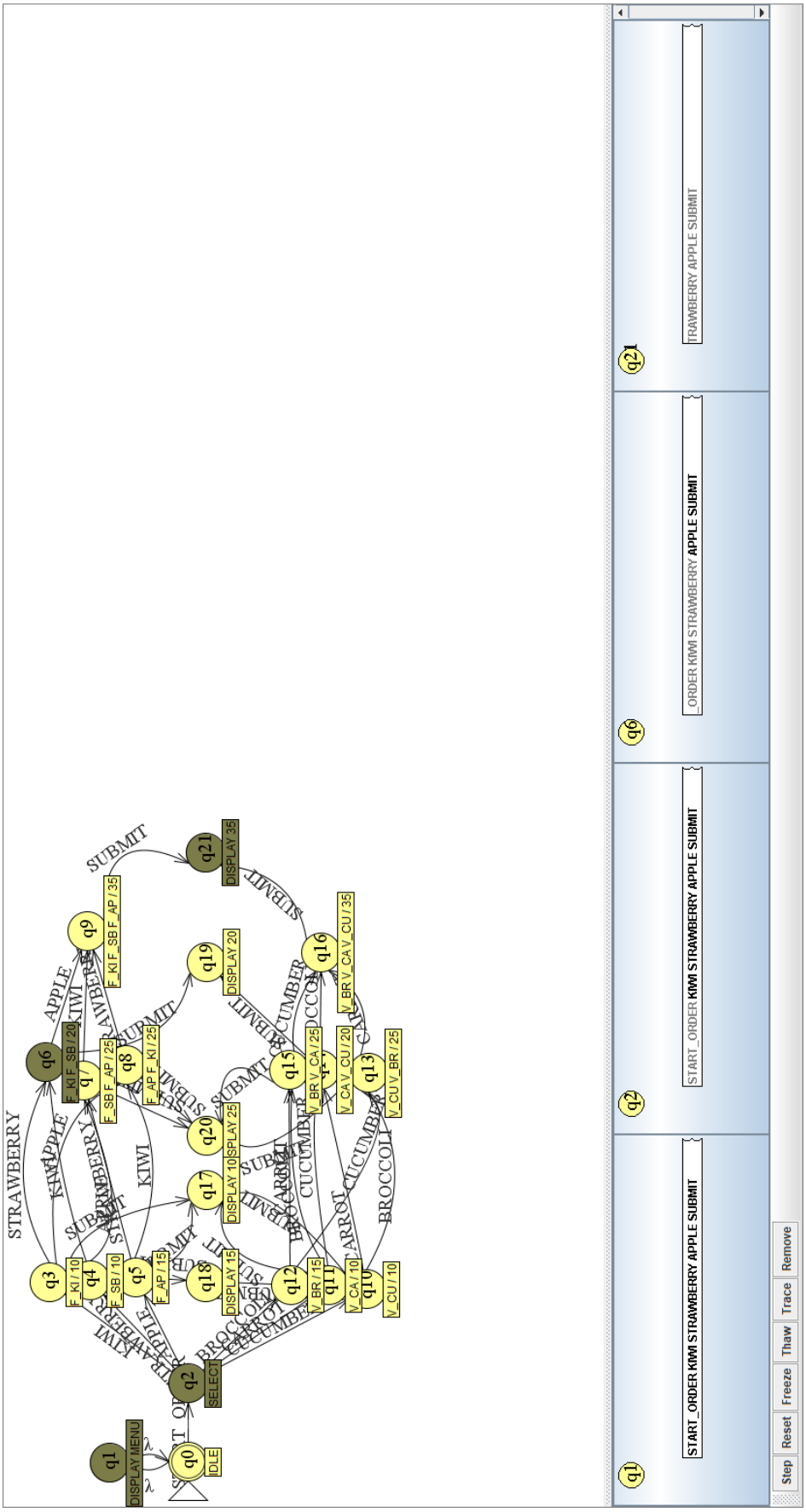
$$\Sigma = \{START\ ORDER,\ KIWI,\ STRAWBERRY,\ APPLE,\ BROCCOLI,\ CARROT,\ CUCUMBER,\ SUBMIT,\ \lambda\}$$

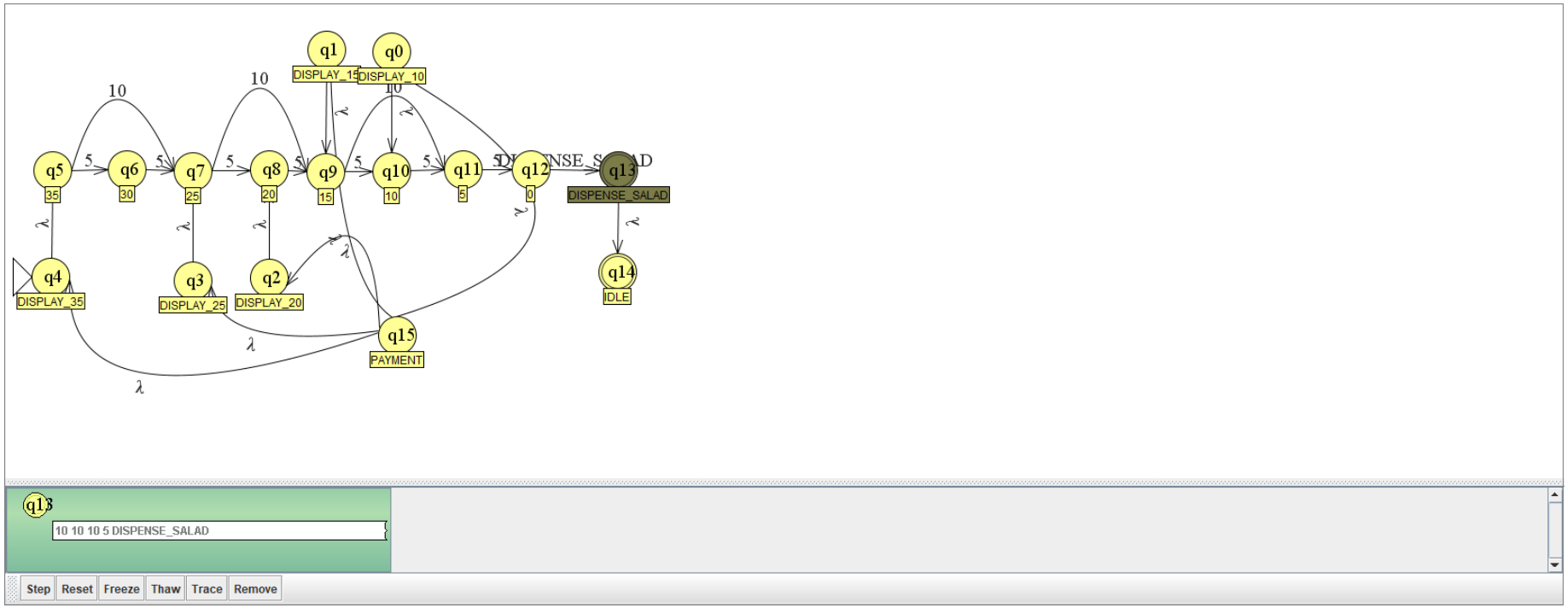
$$\Sigma = \{10,\ 5,\ DISPENSE\ SALAD,\ \lambda\}$$


q0
IDLE
q1
DISPLAY MENU
q2
SELECT
q3
F_KI / 10
q4
F_SB / 10
q5
F_AP / 15
q6
F_KI F_SB / 20
q7
F_SB F_AP / 25
q8
F_AP F_KI / 25
q9
F_KI F_SB F_AP / 35
q10
V_CU / 10
q11
V_CA / 10
q12
V_BR / 15
q13
V_CU V_BR / 25
q14
V_CA V_CU / 20
q15
V_BR V_CA / 25
q16
V_BR V_CA V_CU / 35
q17
DISPLAY 10
q18
DISPLAY 15
q19
DISPLAY 20
q20
DISPLAY 25
q21
DISPLAY 35
START ORDER
λ
λ
KIWI
STRAWBERRY
APPLE
BROCCOLI
CARROT
CUCUMBER
SUBMIT


q0
DISPLAY_10
q1
DISPLAY_15
q2
DISPLAY_20
q3
DISPLAY_25
q4
DISPLAY_35
q5
35
q6
30
q7
25
q8
20
q9
15
q10
10
q11
5
q12
0
q13
DISPENSE_SALAD
q14
IDLE
q15
PAYMENT
DISPENSE_SALAD
γ
λ
5
10

Validation:


q1
DISPLAY MENU
q0
IDLE
q2
SELECT
q3
F_KI / 10
q4
F_SB / 10
q5
F_AP / 15
q6
F_KI F_SB / 20
q7
F_SB F_AP / 25
q8
F_AP F_KI / 25
q9
F_KI F_SB F_AP / 35
q10
V_CU / 10
q11
V_CA / 10
q12
V_BR / 15
q13
V_CU V_BR / 25
q15
V_BR V_CA / 25
q16
V_BR V_CA V_CU / 35
q17
DISPLAY 10
q18
DISPLAY 15
q19
DISPLAY 20
q20
DISPLAY 25
q21
DISPLAY 35
STRAWBERRY
APPLE
KIWI
CUCUMBER
BROCCOLI
CARROT
SUBMIT
START_ORDER KIWI STRAWBERRY APPLE SUBMIT
START_ORDER KIWI STRAWBERRY APPLE SUBMIT
ORDER KIWI STRAWBERRY APPLE SUBMIT
STRAWBERRY APPLE SUBMIT
Step
Reset
Freeze
Thaw
Trace
Remove


q1
DISPLAY_15
q0
DISPLAY_10
10
10
10
q5
35
5
q6
30
5
q7
25
5
q8
20
5
q9
15
5
q10
10
5
q11
5
5
q12
0
q13
DISPENSE_SALAD
q4
DISPLAY_35
q3
DISPLAY_25
q2
DISPLAY_20
q15
PAYMENT
q14
IDLE
λ
q13
10 10 10 5 DISPENSE_SALAD
Step
Reset
Freeze
Thaw
Trace
Remove

B 1.4 Concluding Remarks:

In conclusion the required Salad Vending Machine was made using NFA, which was tested for some test inputs in JFLAP, which was successfully executed and ended up in the final state of dispense salad.

Limitations:

The number of fruits and vegetables is only limited to 6, also each fruit can only be selected once, multiple times of the same fruit cannot be selected, since the current implementation does not keep track of number of times the user selects the same fruit.

The current NFA does not work for currency in form of notes, it only accepts coins of denomination, also it doesn't keep track of the change, for example if the user has to do a payment of 35, and he inserts 4, 10 rupees coins, then he should receive a change of 5, but the machine would not accept 40 in the first place and only accepts exact change, if not then the salad is not dispensed.

Improvements:

To improve the current implementation the number of choices for the fruits and vegetables can be increased, and a feature to accept denominations greater than the payment price has to be added, which would require to tender change to the customer.

Instead of using a Normal NFA, NFA with Output can be used, which has:

```
Q - Set of states
S - Start state which is an element of Q
Sigma - Set of input symbols
Pi - Set of output symbols
Delta - Maps a state and a symbol from Sigma to a state and a symbol from Pi
```

Note: It has no set of final states but has set of output symbols.

Solution to Question No. 2 Part B:

B 2.1 Introduction:

Just as finite automata are used to recognize patterns of strings, regular expressions are used to generate patterns of strings. A regular expression is an algebraic formula whose value is a pattern consisting of a set of strings, called the language of the expression.

Operands in a regular expression can be:

- characters from the alphabet over which the regular expression is defined.
- variables whose values are any pattern defined by a regular expression.
- epsilon which denotes the empty string containing no characters.
- null which denotes the empty set of strings.

In this problem a regular expression has to be generated for the set of inputs that are valid for the automata made in the previous question.

B 2.2 Problem solving approach:

The regular expression for it can be designed by looking at all the possible inputs that can be given from the current state to go into the next state.

We'll separate the displaying the price and the payment process as two Regular Expressions for the sake of simplicity, the former design that was made in B1 is also similar where the SVM was split into two.

RE will have to start with START_ORDER, followed by the fruit/vegetable choice, these are in two separate OR cases, either only fruits or only vegetables can be selected, in fruits every unique combination of 1 fruit, 2 fruit or 3 fruit is taken, so that the regex can accept all combinations of input. Similar thing is done for the vegetables, where every unique combination of 1 vegetable, 2 vegetable or 3 vegetable is taken as a valid input. After these inputs the final input is SUBMIT button which then displays the price for the inputs chosen, that is taken care by the NFA. We cannot use wildcards like "$*$" for multiplicity because our number of fruits are fixed, we are not keeping track of the same fruit being taken multiple times. As it would increase the number of cases for the price, factorially high.

This RE also has to accept λ as an input to display the menu, which does not require any input, hence the menu is always displayed to the user.

RE for the payment process is designed by taking into account that there are only 5 states for the price, 10, 15, 20, 25, and 35. The machine only accepts coins of 5 and 10, which are our inputs. Now for each denomination there are only finite number of possibilities, taking example of 10, it can be made by taking 2, 5 rupee coins or 1, 10 rupee coin, similar strategy is used to compute the possibilities for the other denominations. Once again, we cannot use the wild card "*" because our number of states are fixed, and we cannot keep track if excess amount is paid, also we do not care for the amount paid if greater than what the user was supposed to pay, which would require the "dispense change" logic, that is not part of this NFA. The number of ways to complete the transaction is fixed. A soon as the required amount is got by the machine, the salad is dispensed.

B 2.3 Design and Validation:

Regular Expression for Selection and Display of the Price to Be Payed:

```
RE = (START_ORDER . (
(KIWI)+(STRAWBERRY)+(APPLE)+
(KIWI STRAWBERRY)+(KIWI APPLE)+(STRAWBERRY KIWI)+(STRAWBERRY
APPLE)+(APPLE STRAWBERRY)+(APPLE KIWI)+
(KIWI STRAWBERRY APPLE)+(KIWI APPLE STRAWBERRY)+(STRAWBERRY KIWI
APPPLE)+(STRAWBERRY APPLE KIWI)+(APPLE STRAWBERRY KIWI)+(APPLE KIWI
STRAWBERRY)
) + (
(BROCCOLI)+(CARROT)+(CUCUMBER)+
(BROCCOLI CARROT)+(CARROT CUCUMBER)+(CUCUMBER BROCCOLI)+
(BROCCOLI CARROT CUCUMBER)+(CARROT CUCUMBER BROCCOLI)+(CUCUMBER BROCCOLI
CARROT)
) . SUBMIT)
+ λ
```

Regular Expression for the Payment Process:

```
(5 5)+(10)+
(5 5 5)+(10 5)+(5 10)+
(5 5 5 5)+(10 5 5)+(5 10 5)+(5 5 10)+(10 10)
(5 5 5 5 5)+ (10 5 5 5)+ (5 10 5 5)+(5 5 10 5)+(5 5 5 10)+(10 10 5)+(5 10
10)+(10 5 10)+
```

(5 5 5 5 5 5 5)+(10 5 5 5 5 5)+ (10 5 5 5 5 5)+(5 10 5 5 5 5)+(5 5 10 5 5 5)+(5 5 5 10 5 5)+(5 5 5 5 10 5)+(5 5 5 5 5 10)+(10 10 5 5 5)+(5 10 10 5 5)+(10 5 10 5 5)+(5 5 10 10 5)+(5 10 5 10 5)+(10 5 5 10 5)+(5 5 5 10 10) (5 5 10 5 10)+(5 10 5 5 10)+(10 5 5 5 10)+ (10 10 10 5)+(5 10 10 10)+(10 5 10 10)+(10 10 5 10)
. DISPENSE_SALAD

Validation:

We use the same inputs as in our previous validation of the NFA,
INPUT : START_ORDER KIWI STRAWBERRY APPLE SUBMIT

Using this input in our regex, the first symbol START_ORDER is validated, we can also see that (KIWI STRAWBERRY APPLE) is a valid input for the regex, and the input ends with SUBMIT which is also valid as per the regex.

INPUT : λ
Since the input is empty, only the Menu is displayed to the user, this is also a valid input, and matches with the regex.

For the Payment option, since we have choose KIWI, STRAWBERRY, APPLE, the total price is 35, which has to be payed by the user,
Taking the same inputs for the NFA as previously
INPUT : 10 10 10 5 DISPENSE_SALAD

Using this input in our regex, the symbols (10 10 10 5) is part of the regex, hence it is validated, the input ends with the symbol DISPENSE_SALAD, which is also part of the regex, both these symbols are in order with the regex and hence it is validated.
Another case that can be taken for payment of 35 is
INPUT : 10 5 5 5 5 5 DISPENSE_SALAD

Here also, the input symbols (10 5 5 5 5 5) is part of the regex and ends with DISPENSE_SALAD, hence it is a valid input.

From the above test cases the regex is validated for the SVM.

B 2.4 Concluding Remarks:

For the NFA made for the SVM in B1 the regular expressions are made to validate the input from the user, since the number of ways of input is limited, as the number of fruits/vegetables is limited, the regex hence made is every possible combinations of those input. Even for the payment process the amount to be payed has fixed number of ways, keeping this in mind the regex is made.

Limitations:

Due to the fact that the regex is every possible combination of input, it's really large.

The regex does not work for the payment where the amount is more than that to be payed, basically it does not handle giving back the remaining change to the user.

Improvements:

Due to the limitations from the NFA that it was split into two parts, the regex is also split, to improve the regex both the NFA's can be combined to form a single regex for better validation of input.

Current regex does not allow the user to go back if multiple coins are inserted and the user wants to cancel the transaction, hence a cancel feature has to be implemented.

1. Jesterxl, 2012, Finite State Machines in Game Development, http://jessewarden.com/2012/07/finite-state-machines-in-game-development.html
2. Eric Gribkoff, Applications of Deterministic Finite Automata, UC Davis, Spring 2013