Q3

Would it be possible to exhaustively test the syntax of GLADIUS commands (that is, to write tests which have derivation coverage)? Why or why not? What about if we restricted ourselves to just the shop flight fare command? Give reasons for your conclusions.

(Maximum 500 word answer. You may use Markdown formatting in your answer, if you wish. Moodle will perform a basic word count, but won't reveal any marks.)

It is not possible to exhaustively test the syntax of the "air book request" command, because the command can have an infinite number of flight segment lines. The number of derivations is infinite. Therefore, it is not possible to exhaustively test the syntax of GLADIUS commands.

If we restrict ourselves to the “shop flight fare” command, then there are a finite number of derivations. We can calculate the number of derivations by inspecting the relevant grammar rules.

Let N(<a>) represent the number of derivations of the non-terminal <a>.

<airport> ::= [A-D] [A-D] [A-D]

N(<airport>) = 4\*\*3 = 64

<trip\_type> ::= "OneWay" | ("Return" <space> ("1"? [0-9]) | "20")

N(<trip\_type>) = 1 + 21 = 22

<cabin\_type> ::= "P" | "F" | "J" | "C" | "S" | "Y"

N(<cabin\_type>) = 6

<departure\_date> ::= [0-9] [0-9] [0-9] [0-9] "-" [0-9] [0-9] "-" [0-9] [0-9]

N(<departure\_date>) = 10\*\*8

<shop\_command> ::= "shop flight fares" <space> <airport> <space> <airport> <space> <trip\_type> <space> <cabin\_type> <space> <departure\_date>

N(<shop\_command>) = 64 \* 64 \* 22 \* 6 \* 10\*\*8 = 54,067,200,000,000 = 5.40672 \* 1013

If we assume each test takes 1μs, and the test must be able to complete on one core of one computer, the time it would take to test every derivation of the “shop flight fare” command in hours is 5.40672 \* 1013 \* 10-6 / 60 / 60 ≈ 150,187 hours. This is still infeasible to test exhaustively.

Q5

Describe in detail the preconditions and postconditions of the constructor for the ShopFlightFareCommand class, justifying your answer.

(Maximum 500 word answer. You may use Markdown formatting in your answer, if you wish. Moodle will check the word count, but won't reveal any marks.)

Pre-conditions:

* When tripType is RETURN, lengthOfStay must be non-null. Otherwise behaviour is undefined.

Postconditions

* Each parameter will be stored as an attribute of the ShopFlightFareCommand object, and can be accessed through the relevant getter methods.

Q6

Apply Input Space Partitioning (ISP) to the constructor for the SegmentSubcommand constructor, explaining in detail the steps you take and what characteristics and partitions you would use.

You should describe:

* eight characteristics that would be useful for testing the constructor.
* three test cases in detail, including all fixtures, test values and expected values.

You need not exhaustively describe all characteristics, partitions, test cases and values that would be needed, but should briefly discuss what more would be needed – beyond the eight characteristics and three test cases you have described – to achieve Base Choice Coverage, and how you would know when it was achieved.

(Maximum 1000 word answer. You may use Markdown formatting in your answer, if you wish. Moodle will check the word count, but won't reveal any marks.)

The first step in applying ISP to the constructor of SegmentSubcommand is to model it as a mathematical function. We can inspect its parameters and model it as the function:

SegmentSubcommand: (

String origin,

String destination,

String flightNumber,

LocalDate departureDate,

CabinType cabinType,

int numPeople,

Z

) -> Z

Where Z is the receiver parameter, which is the state of the SegmentSubcommand object being constructed.

Some useful characteristics for testing the constructor are:

1. whether origin is valid, syntactically invalid, or syntactically valid but semantically invalid,
2. whether destination is valid, syntactically invalid, or syntactically valid but semantically invalid,
3. whether origin is equal to destination,
4. whether flightNumber is valid, syntactically invalid, or syntactically valid but semantically invalid,
5. whether departureDate is before, equal to, or after today’s date,
6. whether numPeople is less than 1, between 1-10 inclusive, or greater than 10.

Using these characteristics, we could divide the input space into these partitions:

1. p1: origin is valid, p2: origin is syntactically invalid, p3: origin is syntactically valid but semantically invalid,
2. p1: destination is valid, p2: origin is syntactically invalid, p3: origin is syntactically valid but semantically invalid,
3. p1: origin is equal to destination, p2: origin is not equal to destination,
4. p1: flightNumber is valid, p2: flightNumber is syntactically invalid, p3: flightNumber is syntactically valid and semantically invalid,
5. p1: departureDate is before today’s date, p2: departureDate is equal to today’s date, p3: departureDate is after today’s date, and
6. p1: numPeople is less than 1, p2: numPeople is between 1-10 inclusive, p3: numPeople is greater than 10.

To achieve base test coverage, we first choose representative “base” choices for each characteristic. Then we loop through each characteristic and test the function using values from all partitions of that characteristic, while ensuring any other parameters satisfy the base choices of the other characteristics.

Base choices for each of the characteristics could be:

1. origin is valid,
2. destination is valid,
3. origin is not equal to destination,
4. flightNumber is valid,
5. departureDate is 5 days after today’s date, and
6. numPeople is 5.

We would then loop through the characteristics and test values from each partition. The total number of tests needed to achieve base coverage with the characteristics listed is 1 + (3-1) + (3-1) + (2-1) + (3-1) + (3-1) + (3-1) = 12, which corresponds to one test using all base choices, add one less than the number of partitions of each characteristic.

Three such cases could be:

Test case 1

origin = “AAA”

destination = “AAB”

flightNumber = “AA1”

departureDate = “2023-06-01”

cabinType = PremiumFirstClass

numPeople = 5

This test features the base choices of all characteristics. It represents normal, valid behaviour of the SegmentSubcommand constructor. It should not raise an exception.

Test case 2

origin = “AA”

destination = “AAB”

flightNumber = “AA1”

departureDate = “2023-06-01”

cabinType = PremiumFirstClass

numPeople = 5

This test features the base choices of all characteristics except for the first characteristic, because the origin is syntactically invalid. This test should raise a SyntacticError.

Test case 3

origin = “AAA”

destination = “AAA”

flightNumber = “AA1”

departureDate = “2023-06-01”

cabinType = PremiumFirstClass

numPeople = 5

This test features the base choices of all characteristics except for the “origin equal to destination” characteristic. This test should raise a SemanticError because the origin is the same as the destination.

To achieve base test coverage of the given characteristics, nine more test cases must be devised for the remaining partitions.

Q8

Describe a set of test cases for the CommandParser.parse() method which have production coverage of the grammar you specified as an answer to question 1. Explain why it is that your test cases satisfy this coverage criterion.

(Maximum 1000 word answer. You may use Markdown formatting in your answer, if you wish. Moodle will check the word count, but won't reveal any marks.)