Exercise 2: Testing with Input Space Partitioning

Cole Stankov (301295209)
Saqib Aziz Dhuka (301338669)
CMPT 473

Specification of the Program Under Test

For this assignment we decided to test a CLI application written in JavaScript for a CSV to Json converter (Application link: https://github.com/julien-f/csv2json).

CSV specification (input file format):

CSV file format reference: https://tools.ietf.org/html/rfc4180

While there are various implementations of CSV format there is no exact formal implementation, which allows for a wide variety of interpretations of CSV files. The following specifications will follow the format that is typically implemented:

- For the header there may be one or more fields and they are separated by a comma.
- Each line should contain the same number of fields
- Spaces are included within the field
- The last field in the record shouldn't be followed by a comma
- Fields may or may not be enclosed in double quotes, however, if they are enclosed then the field cannot contain double quotes.
- Fields containing line breaks, double quotes and commas should be enclosed in double-quotes
- If a double quote is used in a double quote enclosed field then the double quote must be escaped by preceding it with another double quote.

ABNF grammar for a CSV file appears as follows:

```
file = [header CRLF] record *(CRLF record) [CRLF]
header = name *(COMMA name)
record = field *(COMMA field)
name = field
field = (escaped / non-escaped)
escaped = DQUOTE *(TEXTDATA / COMMA / CR / LF / 2DQUOTE) DQUOTE
non-escaped = *TEXTDATA
COMMA = %x2C
CR = %x0D ;
```

JSON specification (output file format):

JSON file format reference: https://tools.ietf.org/html/rfc8259

- A JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).
- An object is considered as a collection of zero or more name/value pairs. The name is a string and the value is a string and the value is one of the primitive types mentioned previously.
- An array is an order of sequence of zero or more values.
- The literal names must be in lowercase (ex. true, false, null)

```
<Json> = <Object> || <Array>
<Object> = '{' '}' || '{' <Members> '}' || '{' <Members>, <Members> '}'
<Members> = <name: value>
<Array> = '[' ']' || '[' <Element> ']' || '[' <Element>, <Element> ']'
<Element> = <value>
<value> = string ||  number || boolean || null || <Array> || <Object>
```

Parameters for csv2json:

- For this assignment we decided to leave out tests that we felt weren't focused on the processing of a csv file to a json object.
- The program csv2json contains options -d (dynamically typed), -s (separator), -t (tab separator), -help and an input and output fields.
- We decided to exclude the -help for this assignment. The -help was removed due to it not focusing on processing the csv files.
- We will be testing with the -d, -s and -t options and input and output fields.
- For -s we will use " " (space) as the custom delimiter.
- Some tests will contain no options and solely the input and output fields.
- Invalid options do not concern the flags since the program is expected to crash.
- Although csv2json allows for disk and stdin input and output we will only be testing disk for the scope of this assignment.

Input CSV File:

- The input will be valid paths to valid csv files.
- The input file that will be tested will contain either an empty file, single line csv file or multi-lined csv file and will contain different delimiters as separators.
- We will not be testing for valid file extensions and it is assumed the input will be a .csv file.

Output Json File:

- The output will be valid paths to valid json files.
- A Json file will be generated by the csv2json program as output.
- The output generated will be compared to expected output files that we created.

Testing specifications:

The testing platform is a script in python 3.8.5 which will run the csv2json program with a series of tests that are labeled 1-9. The test will write to output json files and compare the results to expected output files. A message will also be written to the message files and be compared to expected message files. If both the message and output match then the test is considered passed.

- Command to run the test suite: python3 exercise2.py

Input Space Partitioning

We analyzed the -d, -s and -t option and the input and output field and their corresponding characteristics. Following what was discussed in class regarding input space partitioning we came to the following conclusion for our input domain model:

IO Environment
- Input_file : stdin, disk
- Output_file : stdin, disk
- Valid_input_path : true, false
- Valid_output_path : true, false

Command options
- Option_d : true, false
- Option_s : true, false
- Option_t : true, false

Input field
- Fields_separator : commas, tabs, none
- Contents_of_file_length : ZERO_REC, GREATER_THAN_ZERO_REC
- Numeric_and_boolean_vals_present : true, false

Constraints

1. Input type = “disk”
2. Output type = “disk”
3. option_t = true => Option_s = false
4. Option_s = true => option_t = false
5. Valid_input_path = false => Valid_output_path = true
6. Field_separator = "none" => (Valid_input_path = false || Valid_output_path = false || Contents_of_file_length = "ZERO_REC")
7. (Valid_output_path = false) => (Numeric_and_boolean_vals_present = false && Option_d = false && Option_s = false && option_t = false && Field_separator = "none")

8. (Valid_input_path = false) => (Numeric_and_boolean_vals_present = false && Option_d = false && Option_s = false && option_t = false && Field_separator = "none")
9. Contents_of_file_length = "ZERO_REC" => Numeric_and_boolean_vals_present = false && Option_d = false && Option_s = false && option_t = false && Valid_input_path = true && Valid_output_path = true && Field_separator = "none"

Constraints 1 and 2 are needed to limit the input and output type to only disk due to us limiting the testing to only disk and removing stdin from the input space. Constraints 3 and 4 are used due to an overlap in characteristics for -t and -s since they both deal with separating the file by either a given delimiter or tab. Constraint 5 is to remove redundancy in test cases since we don't need to see how the program handles if both input and output paths are invalid at the same time. 6 is constraining the fact that the only way a file should have a “none” separator is if either the input or output file path is invalid or if the record is empty. Constraints 7 and 8 are used to limit the redundant test cases since if the input and output file paths are invalid the other options are no longer relevant. Lastly constraint 9 is used to remove all other characteristics that specify contents of a document or options that act on those contents for empty files.

Combinatorial

Test Generation using ACTS

The ACTS tool was used to generate each testframe characteristics for a given number of tests. It was then converted to actual tests created from scratch in python. The ACTS file can be found within the csv2json.xml file.

Test Results

The ACTs generated 9 test cases which all worked as expected. The two test cases for invalid input and output were expected to show error messages since paths were invalid, but apart from that all test cases passed successfully. Without pair-wise testing we assume there would have been 96 total test cases using conventional software testing methods. These are a lot of test cases and although we would’ve been able to go through each test case and remove redundancy, it would have been nearly impossible to filter out these 9 test cases that we got through ACTs. By working with functionality based techniques we were able to reveal much more important characteristics of parameters themselves. So for example, for our code, one parameter was the input file, and instead of looking at the input of the file as a simple input file path (as in interface based approach), we were able to explore its characteristics like the contents of the files which are quite important in testing such programs. Despite this

advantage, pairwise testing does have a drawback of not showing how more than three parameters together can affect the program's behaviour and the result. So for programs that work with only three parameters, pairwise testing would perhaps be more suited.

Reflection

We found this assignment pretty straightforward, especially the ACTS software after understanding the features. After we found our test cases using ACTS we found that setting up the python script from scratch was not that difficult. However, the difficulty arose when we had to decide on the input space partitioning variables. We found ourselves correcting and adjusting the variables and their constraints quite often as we had to make sure that the completeness and disjoint property was maintained. The other surprisingly more difficult than we thought it would be was finding proper documentation for csv and json file specification. Although the ACTS software was easy to get used to, an in class working demo or a video reference would have cleared up some of the confusion we had.