CMPT 473 - A2 - Assignment Report

Patrick San Juan

Jordan Ho

https://github.com/pjsanjuan/CMPT473-Asn2-ACTS

Specification of the Program Under Test

For this assignment we tested a CLI application for a json2csv converter (https://github.com/jehiah/json2csv) written in Go. An example use of the program can be found in the GitHub repository. For the program to work correctly, we've found that each JSON object must be in its own line and we've taken this into consideration when creating the input domain model.

Parameters of json2csv command:

- Since the assignment requirements focuses on processing input files, we decided to leave out tests that didn't focus on processing input files.
- The json2csv program can take input from 'stdin', but this was not the focus of the assignment requirements so we exclude this functionality from testing.
- We also decided to exclude the '--version', '-p' and '-h' command line arguments from out testing for the same reasons as mentioned above.
- In addition, we will be testing with all the '-i', '-k' and '-o' options included and ignoring the program without usage of any options.

Input Json File:

- input file will be valid a path to host operating system file system
- We will not be testing for valid file extensions as it will be assumed it is using a .json file extension
- The input files we will test will contain either an empty file, valid JSON or invalid JSON syntax

Output Csv File:

- output file will be valid a path to host operating system file system
- The json2csv program will generate a file to the output file path
- We have created static expected output csv files to compare them with the generated csv file from running the json2csv command

Output Messages and Expected Messages:

- We have created a file structure to our assignment to store expected and generated output messages

Testing Harness:

- We have written a Python script to run the json2csv program with a series of tests specified by the ACTs program

- The Python program is written in version 2.7.12

- The Python program will:
    - test and compare the generated output csv files to expected output csv files using the diff command with os system calls and prints its outcome to stdout
    - save execution of the commands of json2csv and diff command with the outcome of the diff command to a message.txt file that will be stored in a output message folder
    - compare the generated output message with an expected message file with the diff command and prints its outcome to stdout

- The results of the test harness can are saved to the “TestReport.txt” file in root file path of the project folder.
- To run the testing harness, run the command in the project root directory to display test results:
    - Python runTests.py

Input Space Partition

To arrive at our results, we followed the 5 step process outlined in slides 77 - 92 of the Input Space Partitioning slide deck to create a functionality based input domain model. By analysing each parameter (command line arguments) to the program (-k, -i, -o) and trying to elicit each of it's characteristics, we arrived at the following input domain model:

- Fields Parameter (-k)
    - FIELDS_EXISTS_IN_JSON_OBJ
        - True
        - False [If JSON_PATH_FALSE] [If FILE_CONTENTS_EMPTY]
- Input Parameter (-i)
    - VALID_JSON_PATH
        - True
        - False [Property JSON_PATH_FALSE]
    - FILE_CONTENTS (Contents of the input file)
        - EMPTY [If JSON_PATH_FALSE] [Property FILE_CONTENTS_EMPTY]
        - MULTIPLE_ROWS
        - SINGLE_ROW
    - ALL_VALID_JSON_OBJECTS (According to JSON specifications)
        - True
        - False [If JSON_PATH_FALSE] [If FILE_CONTENTS_EMPTY]
    - EACH_ROW_HAS_ONE_JSON_OBJECT (Program requires that each JSON object must be in it's own row to work correctly. May or may not be valid JSON)
        - True
        - False [If JSON_PATH_FALSE] [If FILE_CONTENTS_EMPTY]
- Output Parameter (-o)
    - File Path to output CSV file
        - VALID
        - INVALID

Notes:

- all options are expected to have input after them, ignoring the empty case
- all input are expected to NOT have special characters that require escape sequences

Additional Comments

Excluded testing on:

- --version
  - View Version option
- -h
  - help option
- -p
  - print csv header row option
- No options

Included testing only on:

- -k
  - Selects Fields, and nested fields of csv
- -i
  - Path to input json file
- -o
  - Path to generated output csv file

Constraints

The following constraints can be seen in the attached XML file called ‘JSON2CSV.xml’.

Note:

- All testing is done with the options -k, -i, -o including an input parameter (ignores empty case)
- All input are expected to NOT have special characters that require escape sequences

Constraint 1

```
(VALID_JSON_PATH = false) => (FILE_CONTENTS = "EMPTY" &&
ALL_VALID_JSON_OBJECTS = false && EACH_ROW_HAS_ONE_JSON_OBJECT = false
&& FIELDS_EXISTS_IN_JSON_OBJ = false)
```

Constraint 2

```
(FILE_CONTENTS = "EMPTY") => (ALL_VALID_JSON_OBJECTS = false &&
EACH_ROW_HAS_ONE_JSON_OBJECT = false && FIELDS_EXISTS_IN_JSON_OBJ =
false)
```

Explanation of Constraints

Since `FILE_CONTENTS != EMPTY` and `VALID_JSON_PATH = true` are prerequisites that must be satisfied before `FIELDS_EXISTS_IN_JSON_OBJ, ALL_VALID_JSON_OBJECTS, EACH_ROW_HAS_ONE_JSON_OBJECT` are even applicable to the test, we have decided to default the values of these last 3 characteristics (referred to as post-requisites) to false if the prerequisites are not satisfied. Constraint 1 says that the 3 post-requisites must be false if the `VALID_JSON_PATH = FALSE`. Similarly constraint 2 says that the 3 post-requisites must be false if `FILE_CONTENTS = EMPTY`.

Test Results

The ACTs generated 9 tests and all tests passed. Test results are saved to the "TestReport.txt" file in root file path of the project folder. If the ACTs had not been used, then we assume there would've been 48 tests ($2^4 * 3$) if we were to test every possible combination of the parameters and characteristics specified by the input domain model. Although we would've likely identified the redundant cases in the 48 tests, we probably wouldn't have been able to bring the test frames to as low a number as 9. Following a functionality based approach to create input domain partitioning took more effort and time than interface based partitioning. However, the functionality based approach revealed important test frames we could've missed if we simply tried to eliminate the redundant cases manually. For instance, our the program we tested takes in a file path as a parameter. An interface based approach would only see this as just a file path, but the functionality based approach takes into account the contents of the file -- characteristics that are more important to workings of the program than the file path itself. Pairwise testing may not reveal how multiple parameters (more that 3) interacting with each other can influence the results. The json2csv program under test for this assignment had 3 command line arguments (fields, input file path, output file path) but other program in this category may only accept one parameter. Perhaps pairwise testing is sufficient for these types of programs.

Brief Reflection on Experience

Although the assignment was pretty straightforward. The ACTs tool is easy to use and it was well documented. We struggled to elicit characteristics from the input parameters. We felt the process of eliciting characteristics and constraints was somewhat subjective because throughout the assignment we drafted multiple iterations of the input domain model before finally arriving at one we believe was representative of the program. Thus, we had difficulty determining if certain characteristics and constraints we discovered were important or if they were redundant.

We think programs that satisfy the assignment requirements tend to be simple programs. Unfortunately, the simplicity of these programs makes it difficult to elicit the characteristics of the parameters based on the program specification. We particularly had difficulty creating the constraints because we initially thought that the characteristics of our program were straightforward (true/false characteristics). We believe creating the input domain model for a complex program would have resulted in a better assignment experience because we would have an easier time discovering characteristics from the programs specification. However, its possible we encountered these issues because we chose a simple program and that more complex programs exists that satisfy the assignment requirements.

We also had trouble with implementing a test harness for Json2csv due to the specified assignment structure. The complexity of the project folders containing an expected output files, output files, input files, output message files and expected output message files was slightly confusing and time consuming to implement. Particular for implementing the test Harness with test cases of the program crashing due to invalid inputs, we had to implement a solution to save a test message in the event the program crashes. Luckily we’re allowed to use the diff command to compare files from the CLI. However, we had to use the function called fileCmp in compare files in python to save a message based on the output if its the same or different.