Testing with Input Space Partitioning

Junchen Li & Wenqing Liu & Ningyi Ke

February 19, 2022

1 Environment Set-up Instructions

The open-source program we selected for this exercise requires external commands to execute so please make sure running the following set-up commands before executing python program.

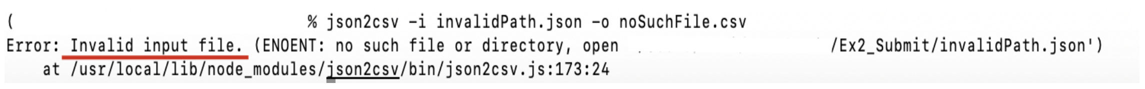
1. Install the json2csv program: *$ npm install*
   - For Mac & Linux: *$ sudo npm install -g json2csv*
   - For Windows: *$ pip npm install -g json2csv*
2. Validate if the installation is successful: *$ json2csv -V* (output the version number)
3. Go to project directory
4. Run python program with: *$ python3 exer_ tests_ code.py*

2 Specification of the Program Under Test

For this exercise, we tested an open-source program https://github.com/zemirco/json2csv that aims to convert file format from JavaScript Object Notation (JSON) into Comma-Separated Values (CSV) which is written in JavaScript. The input JSON file must be in the valid JSON format with column titles and proper line endings. Thus, all JSON files in TestData folder are valid and ready to be passed into the terminal.

Input

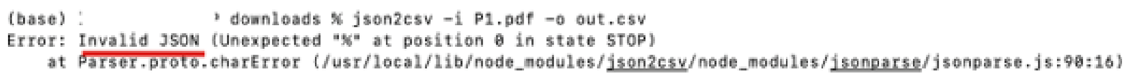
In order for the program to be run correctly, we applied certain restrictions on input files. The first thing we take into consideration is the valid input file path. Since we pass the input argument through Command-Line Interface (CLI), the test data must be located in the same directory as the program. As Figure 1 shows, when a file with invalid path is passed in, the program is terminated due to an invalid input file error.



```
(                                % json2csv -i invalidPath.json -o noSuchFile.csv
Error: Invalid input file. (ENOENT: no such file or directory, open                    /Ex2_Submit/invalidPath.json')
    at /usr/local/lib/node_modules/json2csv/bin/json2csv.js:173:24
```

Figure 1: Invalid file path

Secondly, this testing program only accepts JSON files as the input so we decided to exclude other file extensions. When the incorrect file extension is passed into the terminal, such as .pdf, it leads to 'Invalid JSON' error as shown below.

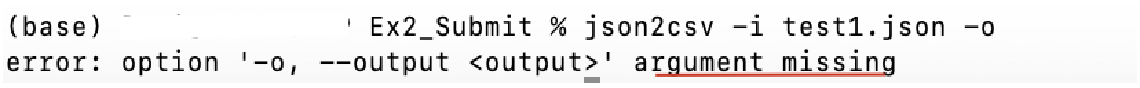


```
(base)                ' downloads % json2csv -i P1.pdf -o out.csv
Error: Invalid JSON (Unexpected "%" at position 0 in state STOP)
    at Parser.proto.charError (/usr/local/lib/node_modules/json2csv/node_modules/jsonparse/jsonparse.js:90:16)
```

Figure 2: Invalid file extension

Output

In order for the program to be run correctly, we apply certain restrictions on output files. Based on the README.md on the GitHub repo, the command that is used to run this program takes two arguments in total. Thus, we assume there are two arguments that must be passed to "json2csv -i <jsonFile> -o <csvFile>". Error message shown in Figure 3. After running this command, the output file is generated and placed in the same working directory.



```
(base) [redacted] Ex2_Submit % json2csv -i test1.json -o
error: option '-o, --output <output>' argument missing
```

Figure 3: Missing argument

3 Input Space Partitioning

In order to construct an input domain model, we considered characteristics from two aspects, file content and additional commands passed into the command line.

In terms of the file content, we divided it into two main categories, empty and non-empty. If the input JSON file is not empty, we consider whether the content contains an empty JSON object, backslash, and quote mark.

We also analysed the -Q, -q, -e, -d commands to specifically check whether the program can correctly handle some special characters, such as space, slash, end-of-line, and quotes. These optional commands need to be used after the normal output command (-o output.csv).

File Content

1. EMPTY_FILE (File is empty)

   – True
   – False

2.BACK_SLASH_AT_END (File content is not empty and contains backslash at the end)

   – True
   – False

3.INCLUDE_EMPTY_ROW (File content is not empty and includes at least one empty row)

   – True
   – False

4.HAS_QUOTE (File content is not empty and includes at least one quote mark)

   – True
   – False

Additional Commands

5.CMD_Q_ESCAPED_QUOTE (-Q)

- Additional command that is used to replace all escaped quote (\") in the file with customized character(s), default to a double 'quote', ' "" '

   – True
   – False

6.CMD_q_QUOTE (-q)

- Additional command that used to replace all quote mark (") in the file with customized character(s), default to, ' " '

  - True
  - False

7.CMD_e_EOL (-e)

- Additional command that used to replace all delimiter in the file with customized character(s), default to, ',’
  - True
  - False

Input Space Domain Model

Characteristics	B1	B2
EMPTY_FILE	True	False
BACK_SLASH_AT_END	True	False
INCLUDE_EMPTY_ROW	True	False
HAS_QUOTE	True	False
CMD_Q_ESCAPED_QUOTE	True	False
CMD_q_QUOTE	True	False
CMD_e_EOL	True	False
CMD_d_DELIMITER	True	False

Constraint1:

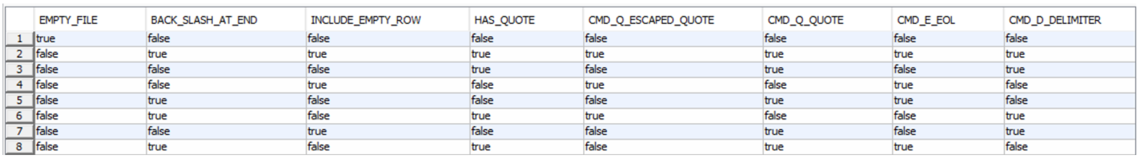
‘EMPTY_FILE = True’ means ‘BACK_SLASH_AT_END = False’ and ‘INCLUDE_EMPTY_ROW = False’ and ‘HAS_QUOTE = False’ and ‘CMD_Q_ESCAPED_QUOTE = False’ and ‘CMD_q_QUOTE = False’ and ‘CMD_e_EOL = False’ and ‘CMD_d_DELIMITER = False’

Explanation

If the input JSON file is empty, there is no need to check for its content anymore. In other words, BACK_SLASH_AT_END, INCLUDE_EMPTY_ROW, HAS_QUOTE are automatically assigned False when EMPTY_FILE is assigned True. Since the empty file always output an empty CSV file no matter what additional commands are passed in, it is the same as not passing any additional commands. Thus, we also assigned False to all CMD_X_XXX parameters when EMPTY_FILE is assigned True.

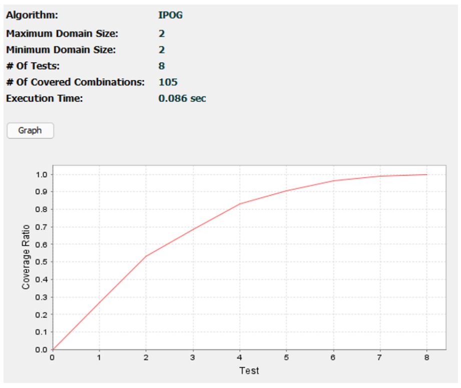
4 Combinatorial Test Generation using ACTS

ACTS is a powerful tool for generating t-way combinatorial test sets based on custom parameters, constraints, and strength. According to what we had from the previous section, we entered these eight parameters along with their boolean values and applied the constraint for the test generation process to effectively reduce the number of test cases. In addition, based on the slide decks for Lec6-Slide 11, the test strength with a value greater than two often does not help too much for the test results. Thus, we decided to use 2-way combinatorial test sets for this exercise. The detailed ACTS file can be found in e2_ACTS.xml. We successfully generated 8 test cases (see Figure 4) over 105 combinations in total and these 8 test cases reached a coverage ratio of about 1.0 (see Figure 5). Test sets generated from ACTS were converted to actual executable tests in the python script exer_tests_code.py.



	EMPTY_FILE	BACK_SLASH_AT_END	INCLUDE_EMPTY_ROW	HAS_QUOTE	CMD_Q_ESCAPED_QUOTE	CMD_Q_QUOTE	CMD_E_EOL	CMD_D_DELIMITER
1	true	false	false	false	false	false	false	false
2	false	true	true	true	true	true	true	true
3	false	false	false	true	false	true	false	true
4	false	false	true	false	true	false	true	false
5	false	true	false	false	false	true	true	false
6	false	true	false	true	true	false	false	true
7	false	false	true	false	false	true	false	true
8	false	true	false	true	false	true	true	false

Figure 4: Tables generated by ACTS

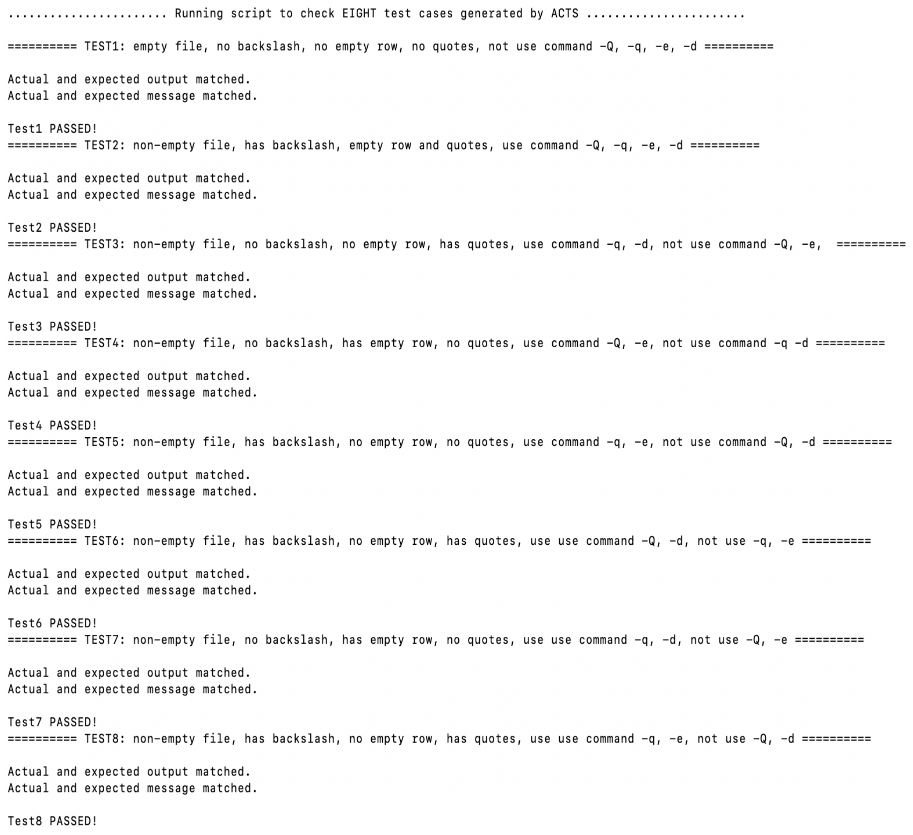


```
Algorithm:                      IPOG
Maximum Domain Size:            2
Minimum Domain Size:            2
# Of Tests:                     8
# Of Covered Combinations:      105
Execution Time:                 0.086 sec
```

Figure 5: Tables generated by ACTS

5 Test Results

ACTS generated 8 test cases based on eight input parameters by using pairwise testing. From the test results we got from running the python script, all test cases passed (see Figure 6). As mentioned above, we excluded the cases of invalid input/output file paths. When we constructed the command in the script, we assumed a valid file path was passed to the terminal through 'os.system()'. In this way, we only focused on comparing the actual CSV file content and the expected one. If these two corresponding files have exactly identical content, we write a message into a message text file and compare the actual message file with the expected one again. Hence, if a test case can generate the correct output file and message file, we will treat it as a passed test case. Those 8 test cases covered combinations of different JSON file content with different numbers of additional command(s). As shown in the Figure 6, each test case had a concrete description of what kind of combination it is testing. Pair-wise testing provides an alternative to reduce the number of test cases and execution time but it still can increase the test coverage effectively.



```
....................... Running script to check EIGHT test cases generated by ACTS .......................

========== TEST1: empty file, no backslash, no empty row, no quotes, not use command -Q, -q, -e, -d ==========

Actual and expected output matched.
Actual and expected message matched.

Test1 PASSED!
========== TEST2: non-empty file, has backslash, empty row and quotes, use command -Q, -q, -e, -d ==========

Actual and expected output matched.
Actual and expected message matched.

Test2 PASSED!
========== TEST3: non-empty file, no backslash, no empty row, has quotes, use command -q, -d, not use command -Q, -e,  ==========

Actual and expected output matched.
Actual and expected message matched.

Test3 PASSED!
========== TEST4: non-empty file, no backslash, has empty row, no quotes, use command -Q, -e, not use command -q -d ==========

Actual and expected output matched.
Actual and expected message matched.

Test4 PASSED!
========== TEST5: non-empty file, has backslash, no empty row, no quotes, use command -q, -e, not use command -Q, -d ==========

Actual and expected output matched.
Actual and expected message matched.

Test5 PASSED!
========== TEST6: non-empty file, has backslash, no empty row, has quotes, use use command -Q, -d, not use -q, -e ==========

Actual and expected output matched.
Actual and expected message matched.

Test6 PASSED!
========== TEST7: non-empty file, no backslash, has empty row, no quotes, use use command -q, -d, not use -Q, -e ==========

Actual and expected output matched.
Actual and expected message matched.

Test7 PASSED!
========== TEST8: non-empty file, has backslash, no empty row, has quotes, use use command -q, -e, not use -Q, -d ==========

Actual and expected output matched.
Actual and expected message matched.

Test8 PASSED!
```

Figure 6: Tables generated by ACTS

If the pair-wise testing is not selected for the generation process, there will be 256 test cases (2^8=256) without constraint which is a huge number to be covered and we probably need to spend much more time and energy to execute all test suites.

On the other hand, pair-wise testing has its disadvantages as well. According to its definition, pair-wise testing considers all possible combinations of parameters for each pair of input parameters of the system. Hence, it is very unlikely to provide an accurate relationship between a group with more than three parameters and result. Although the total number of test cases is proportional to test strength, it is still worth exploring the implication of how different numbers of parameters interact with each other and impact the result. If a system only contains three parameters, pair-wise testing is good enough to cover most possible combinations.

6 Reflection

In this assignment, we learned how to use ACTS to generate test cases and construct corresponding tests by using Python.

Although this was our first time to use ACTS, the software was really straightforward and convenient to use after spending some time on reading the manual. It helped us to generate a small number of test cases but also provided a high coverage ratio.

However, in the process of doing the project, we also encountered some problems. For example, in order to search for a qualified open-source program with high correctness, we spent about three days on GitHub to create a list of possible programs. We analysed these programs through its correctness, code complexity and ease of use, so it was a great challenge to select the most suitable one. Moreover, since we misunderstood the purpose of the assignment at the very beginning, we did not come up with the suitable characteristics, which resulted in the failure of our test to execute properly. After trial and error, we found the appropriate parameters and successfully constructed test cases with ACTS.

Through this assignment, we gained a better understanding of input space partitioning and acquired hands-on experience of how to select appropriate characteristics to create input space models.