Testing with Input Space Partitioning

Note: This is a converted markdown document.
*To view original file with correct format, see **README.md** on GitLab.*

Contents

Program Specification

Program under test: **csv2json**.
Description: convert CSVs into JSON documents.

> https://github.com/apolitical/csv2json

Usage

```
$ csv2json [FLAGS] [OPTIONS] --in <FILE>
```

```
$ csv2json --in file.csv > file.json
```

Flags

Flag	Use
`--remove-empty-objects`	Do not output empty objects
`--remove-empty-strings`	Do not output empty strings
`-a`, `--arrays`	Indicate the CSV file contains arrays
`-h`, `--help`	Show program help
`-V`, `--version`	Show program version

Options

Option	Specifies
`-i`, `--in <FILE>`	Input **File** *(required)*
`-o`, `--out-dir <DIR>`	Output **Directory**
`-f`, `--out-name <TEMPLATE>`	**Template** name for multiple output files
`-d`, `--delimiter <DELIMITER>`	**Delimiter** character
`-D`, `--dimensional-separator <SEPARATOR>`	**Separator** character for deeper objects
`-b`, `--boolean <COLUMN>...`	**Columns** which contain boolean values
`-n`, `--numeric <COLUMN>...`	**Columns** which contain numeric values

Input Space Model

To obtain the input domain model, we followed the steps described in the course notes on input space partitioning.

Program Input and Output

Data Formats

Input

- Definition of the CSV Format [RFC4180].
    - File = [header CRLF] record *(CRLF record) [CRLF]
    - Header = name *(, name)
    - Record = field *(, field)
    - Name = field
    - Field = (escaped / non-escaped)
    - Escaped = " *(_text_ / , / CR / LF / "") "
    - Non-escaped = *_text_

Output

- JSON Grammar [RFC8259].
- **csv2json** output specification.

Analysis

We came up with the following, after thoroughly analyzing specifications of the program and CSV format: *

Component: core functionality of **csv2json** program. * Parameters: Input file; Contents of the input file; and User input.

Input Space Partition

Characteristics

Input File

- The input file which the program will read is a required parameter user specifies using optior `--in`. The program is expected to output JSON objects when provided plaintext files. Otherwise, the program is expected to output an error message.

```
input_file_type (enum) : plaintext,
                         non_plaintext,
                         multiple,
                         directory,
                         non_existant
```

- We will test how well the program handles more workload by defining a size characteristic. The difference in sizes between files will be relative to the size of the input file.

```
input_file_size (enum)  : small,
                          medium,
                          large,
                          none
```

File Contents

- The contents of the input file are**records** consisting of delimited **fields**. The program should be able to work correctly with any records with valid CSV syntax shown above.

```
 csv_syntax_is_valid (Boolean)
records_contain_same_number_of_fields (Boolean)
record_contains_trailing_comma (Boolean)
record_field_contains_space (Boolean)
record_field_escaped (Boolean)
record_field_contains_comma (Boolean)
record_field_contains_new_line (Boolean)
record_field_contains_double_quotes (Boolean)
record_field_data (enum) :  only_ascii,
                            only_non_ascii,
                            contains_both
```

- However, the contents need not comply with defined CSV syntax because **csv2json** allows users to change delimiter character or specify other program-specific options unrelated to CSV data format.

```
contents_contain_arrays (Boolean)
contents_contain_empty_objects (Boolean)
contents_contain_empty_strings (Boolean)
contents_contain_nested_objects (Boolean)
delimited_by_specified_character (Boolean)
dimensionally_separated_by_specified_character (Boolean)
```

- Other possibilities of file contents include special cases when the content is empty or unknown because user is not allowed to read the file.

```
contents_empty (Boolean)
no_read_permission (Boolean)
```

User Input

- We only consider cases when the input file is specified because it is a required argument to generate testable program output. `js input_file_specified (Boolean) : true`
- JSON output of the program depends on which options and flags are used.
    - Users can use *three flags*:

```
arrays_flag_set (Boolean)
remove_empty_objects_flag_set (Boolean)
remove_empty_strings_flag_set (Boolean)
```

    - Users can use *six options*, and each option requires a value:

```
output_directory_specified (Boolean)
output_template_name_specified (Boolean)
dimensional_separator_specified (Boolean)
delimiter_character_specified (Boolean)
boolean_columns_specified (Boolean)
numeric_columns_specified (Boolean)
```

- Users can redirect output to a file or to input of another command. *(Implementation of Test Harness could not save and test results of pipelined output.)*

```
output_redirected (enum) : to_standard_output,
                           to_file,
                           to_command
```

Excluded Characteristics

Some characteristics were not considered in our tests because they do not provide valuable test results or we were unable to correctly test them.

```
no_read_permission
output_redirected
```

Constraints

Using our knowledge and intuition, we identified and assigned constraints among the characteristics. By doing this, we were able to increase the quality of the tests and also decrease the number of tests that were required.

The constraints in our test plan can be grouped into five main sets:

1. *Empty or nonexistent file* constraint set.
   Various constraints related to when the file is empty or nonexistent.

   ```
   (input_file_type = "non_existent") => (contents_empty = true)
   (input_file_type = "non_existent") => (input_file_size = "none")
   (input_file_size = "none") => (contents_empty = true)
   (contents_empty = true) => (input_file_size = "none")
   ```

2. *Double-quote enclosure* constraint set.
   According to the CSV definition, "fields containing line breaks (CRLF), double quotes, and commas should be enclosed in double-quotes." To enforce this rule, this second constraint set was established.

   ```
   (record_field_contains_comma = true) => (record_field_escaped = true)
   (record_field_contains_new_line = true) => (record_field_escaped = true)
   (record_field_contains_double_quotes = true) => (record_field_escaped = true)
   ```

3. *CSV file* constraint set. Various characteristics related to the file given to the program. When any of the listed characteristics are true, then it is implied that the CSV must have valid syntax. Reversing the constraint, if a file has valid syntax, then the file cannot be empty.

   ```
   (csv_syntax_is_valid = true) => (contents_empty = false)
   (records_contain_same_number_of_fields = true) => (csv_syntax_is_valid = true)
   (record_contains_trailing_comma = true) => (csv_syntax_is_valid = true)
   (record_field_contains_space = true) => (csv_syntax_is_valid = true)
   (record_field_escaped = true) => (csv_syntax_is_valid = true)
   (contents_contain_empty_objects=true) => (csv_syntax_is_valid = true)
   (contents_contain_nested_objects=true) => (csv_syntax_is_valid = true)
   (delimited_by_specified_character=true) => (csv_syntax_is_valid = true)
   (dimensionally_separated_by_specified_character=true) => (csv_syntax_is_valid = tru
   (contents_contain_empty_strings=true) => (csv_syntax_is_valid = true)
   (dimensionally_separated_by_specified_character = true) => (csv_syntax_is_valid = t
   (contents_contain_arrays=true) => (csv_syntax_is_valid = true)
   (delimited_by_specified_character = true) => (csv_syntax_is_valid = true)
   ```

4. *Input file type* constraint set. Various constraints related to the different input file types. Files coming from a directory or is of a non-plaintext type, would be considered to have invalid CSV syntax.

```
(input_file_type = "directory") => (csv_syntax_is_valid = false)
(input_file_type = "non_plaintext") => (csv_syntax_is_valid = false)
```

5. *Input file option arguments* constraint set. If a file is delimited or dimensionally separated by a specified character, then its corresponding program option must also be set to true.

```
(delimited_by_specified_character = true) => (delimiter_character_specified = true
(dimensionally_separated_by_specified_character = true) => (dimensional_separator_s
```

Test Plan

Step 4 in the course notes was about creating tests based on the chosen characteristics bounded by the constraints, and removing redundant cases; however, as we were using the ACTS tool for our pairwise test generation, all we had to do was enter the parameters and constraints into the tool, and out came a test plan that had no redundant cases and contained tests that sastified constraints. The generated XML file is located in the TestPlan directory. Test Suite document describes each test configuration.

To create valuable tests, we did not consider capabilities of**csv2json** unrelated to its primary task: conversion from CSV to JSON. For that reason, flags `--help` and `--version` were excluded from our test plan.

Using interface and functionality based approaches, we developed a model which allowed us to test the program's configurable options and its compatibility with common commands that a typical user may use. We attempted to create test cases that can correctly determine whether the program can correctly produce output for a variety of input files, diverse file contents, and enabled program options.

Each test configuration represents a possible valid or invalid input the program could receive, including all of: well-formatted CSV files; HTML, XML, markdown, and other plaintext files; images, styled text, binary executable non-plaintext files; paths to files which do not exist, or inclusion of more than one file; and directories with or without files.

All input files have a corresponding text file with options and flags that will be enabled for that test case. Using combinations of options, our test suite determines if **csv2json** correctly produces output for multiple kinds of delimiters, output options, specified boolean/numeric columns, and invalid or not present options.

We discovered one bug while testing**csv2json** program. It occurs when the input is a directory instead of a file. The program stops responding and no error message is printed. The command will continue running until it is interrupted.

Test Infrastructure

Test Harness is a Z-shell script we developed to automatically and quickly iterate over the input files, generate output, and determine test results. Test configurations placed in the Input folder will be included automatically.

The program successfully automates generation of files and messages for different input files and options of the program under test, as well as comparison of each file with expected outcomes. Its options can set the level of detail of generated test reports (`-b` , `-n` , `-q`) or partially automate renaming of all files related to a particular test case (`-r`).

However, we were unable to automate generation of expected output files or messages. All expected files and messages were created manually based on what we thought the output should look like.

Usage

Default Settings

To run the program under test for each input file, enter the following command from the *project's root* directory:

```
$ zsh bin/testharness
```

Custom Settings

Alternatively, you can use extended syntax to specify path to program binary which is not in the `$PATH` , run *Test Harness* from any directory, or enable brief output option.

```
$ testharness [-h] [-b -n -q] [-d /dir/TestData] [-p /bin/csv2json]
    -h, --help                          - Print script usage.
    -b, --brief                         - Output only whether files differ.
    -n, --nopts                         - Do not show process invocation options.
    -q, --quiet                         - Suppress most output.
    -d, --data /dir/TestData            - Path to TestData directory.
    -p, --program /bin/csv2json         - Path to program under test.
    -r, --rename 'old_name' 'new_name'  - Rename test files from 'old_name' to 'new_name
```

> *Recommended shell: **zsh**.*

Automation

Test Harness is a script designed for automation of the test infrastructure:

1. Program reads CSV input files from **TestData/Input/Files** and program options and flags (if any) from the corresponding text files in **TestData/Input/OptionsAndFlags**.
2. Then it passes the file and all parameters to the program under test. The resulting JSON output files are saved to **TestData/Output/Files** and messages outputted by the program are written to

TestData/Output/Messages directory.

```
> Generating output...
    01-empty
    02-instructions --remove-empty-objects --remove-empty-strings
    03-person --boolean person.old person.alive --numeric person.age person.id --di
    04-all_types --boolean LikesChocolate LikesCookies --numeric BirthYear LikesCho
    05_spaces --remove-empty-objects
> Generated 5 files and 5 messages.
```

3. After the output files for each test plan are generated, *Test Harness* begins to compare and report differences between the produced output and the expected output located in **TestData/Expected**.
4. A test is considered to be failed if there exists any difference between the expected and outputted files *or* messages. By default, *Test Harness* prints all differences of compared files `--side-by-side`.
5. A summary of identical/different files and a short test summary is displayed when all tests have been completed.

Sample Test Summary

```
> File comparison summary
    Input File:                Kind:                      File:          Message:
    ✔ 01-S-invalid-syntax      UTF-8 Unicode text         identical   |  identical
    ✘ 02-M-mixed_data          UTF-8 Unicode text         different   |  different
    ✔ 03-L-invalid-syntax      ASCII text                 identical   |  identical
    ✘ 04-empty-text-file       File not found             different   |  different
    ✘ 05-S-Binary              Mach-O 64-bit executab     different   |  identical
    ...
> Compared 58 files: 7 files and 12 messages are different from expected.

> Number of tests ran: 29
    15 out of 29 (52%) tests PASSED ✔
    14 out of 29 (48%) tests FAILED ✘

Done.
TestHarness exited with code 0.
```

Report and Analysis

Generated Tests

The ACTS tool generated 25 tests based on our chosen set of characteristics.
Additionally, we included four other test cases not included in the original suite.

The number of passed and failed tests depended on operating system and shell. When we ran tests on different systems, some test cases produced same error messages with minor differences.
Some test results may have been incorrectly written or read to output files by the automation script when using a shell other than **zsh**.

Running on Ubuntu 18.04 and Z-shell:
15 out of 29 (52%) tests passed
14 out of 29 (48%) tests failed

However, running the same tests on macOS and bash:
9 out of 29 (32%) tests passed
20 out of 29 (68%) tests passed

Pair-wise Testing

By choosing pair-wise testing, we likely missed some non-trivial test cases. While a portion of the missed tests may not reveal any bugs present in the program, a good number may. One example of a test missed in our test plan is the case when the file is of a plaintext type, medium file size, and contains invalid CSV syntax. We can predict that this case would not cause any issues to our program, but this shows that tests can be missed.

If instead of pair-wise we used **Each Choice** testing, we would need to utilize every value of our characteristics we determined above in at least one test. That comes down to as few tests as 5 given the number of choices for the input_file_type characteristic.

On the other hand, with **All Combinations** testing we would need 3,019,898,880 tests to check for the behavior of every possible configuration of our input model domain.

Reflection

What was simple

After deciding what program to use, the task of using and understanding the program was relatively simple as the program we chose had sufficient documentation. Same goes for the ACTS tool; although a bit troublesome to use, the well-written manual made the tool easy to learn.

Developing Test Harness script to automate our test cases was a straightforward process because of the standalone structure of the program. The script was easy to write for the reason that we knew the test process will not change regardless of the chosen program. General test infrastucture was already completed before by the time we fully agreed on which program we will test. Although it was simple to develop, using an existing framework might have been a better choice to avoid compatibility issues.

What was *not* simple

Before the testing process, our first task was to find a program that would be used in our tests. Finding the appropriate program was a challenge because one, many of the programs did not have adequate documentation; this made identifying program properties a problem. And secondly, for some programs that did have good documentation, there were compatibility issues or they were just non-functional.

After selecting our program came the task of deciding the characteristics and constraints to be used in the ACTS tool. The process of deciding what characteristics and constraints to include was not a completely straightforward, unambiguous process; our decisions were partly based on specifications, but also partly based on our own opinions and feelings. What we initially thought of to be the characteristics, characteristic names and types, and constraints for the program were modified again and again before a final set was decided on. This was a very time consuming and frustrating process. Then followed the next issue.

As mentioned, the ACTS tool was easy to learn but troublesome to use. Every time we modified the characteristic names or types, also known as the parameters in the ACTS tool, not only did the parameter have to be removed and re-entered, but also, any associated constraint would also have to be removed and re-entered. There was simply no option to update the parameter name or type in a convenient way.

What could be better

If more programs had better documentation and if more programs that actually do what they say they would do, then the program selection process would have been a lot quicker and less work.

Furthermore, if the ACTS tool had support for parameter name and type changing, and a more automated interface, in a sense that whenever a minor change was made to a parameter, the associated constraint would automatically get updated, then the test creation process would have been somewhat less time consuming.

Extra guidance on how to determine a good and effective set of characteristics would have been a lot of help. Our generated tests are mainly influenced by the set of characteristics we have decided upon. Some of the tests generated by ACTS were not the most effective at gauging the quality of the program under test. While others felt redundant and too similar to previous tests. It is clear after working on this assignment that a set of correctly chosen characteristics is key for the strengths of pairwise testing to shine.

Contributors

Maxim Puchkov
Alan Ou
Kevin Chen