Testing Methodology

The input specifications were tested using partition testing. For every input field, we divided the space of possible inputs into partitions which represented common failure points for such programs (e.g., trailing whitespace). For each of these, we tested: that invalid input errors were only thrown iff the input was truly invalid, that python runtime errors were never thrown, and that invalid output behaviour was never generated.

For oracles, we primarily coded small python functions that checked the assertions described in the previous paragraph (for instance, the return code of the program was used to tell if the program was returning an error message). We also printed the output of the program, as this was a useful tool/heuristic for identifying odd behaviour and possible failure points. Some more complicated oracles were created through python functions (such as for testing what the correct riding-level outcomes were for a specific input).

We also used boundary value testing for the riding_results.csv requirement where the program had a decision to make based on a numerical inequality (R17). Here, the relevant boundary points were the numerical turning points for “Uncontested”, “Plurality”, and “Recount”. We did on point tests for these, along with standard cases (as there is not really a single salient “off point” for these, since the variable we are manipulating is real-valued, so there is no “next value”). The test performed on the on point that separates plurality and recount yielded a bug, as specified below.

Boundary value testing and partition testing were used for the federal_results.csv requirements. Partition value testing was used for R25, where inputs are classified according to the case where there zero, one, or more than one entry of a given party in the input. When numerical values are expected like in R26 - R28, boundary value testing was used to test values of 0, the on point, and greater than 0, the in point. Testing the off point in this case was not needed because the values could never go below 0. For R29, again boundary value testing was used. An example of a boundary value for a majority government could be a case where a party wins two seats and two other parties each win one seat. This is a boundary because by adding another party with one seat, we would get a minority government. The cases for other values are similar.

Failures Identified

type_csv.csv:
When the input file contains one “independent” candidate, the program produces a TypeError. This violates R11, as the program does not terminate gracefully, and R12, as this is a valid input which produces an error.

This error was caught when doing partition testing on federal_results.csv. The partition tested is when the input file includes only “independent” candidates.

number_with_space_before_riding.csv:
When a leading space is inserted into a RidingNum entry, i.e. “ 1”, the program does not catch the invalid input and furthermore recognizes it as a different entry from “1”. In

particular, either “ 1” is not supposed to be interpreted as an integer (and thus the fact that it is accepted violates R4) or it is supposed to be interpreted as an integer (and thus the fact that it is not equivalent to “1”, despite standing for the same integer, violates R8, as here the riding with number “ 1” has a name that differs from that of the riding with number “1”).

This input was generated as part of partition testing the possible inputs to the riding number. Here, we meant to take a representative of the set of riding numbers that contained trailing or leading whitespace. We wanted to see if this would allow two strings that intuitively “stand for the same integer” be associated with two different riding names (and thus violating R8, which is what happened), but we were also checking that, when invalid inputs were caught, the program was terminating gracefully.

one_candidate_caps.csv:
The program incorrectly allows for two rows with Party entries equivalent up to capitalization with the same Riding and RidingNum entries. This violates either R13, or R8.

This input was generated in a way equivalent to the previous one: we were testing possible inputs to party names. In particular, when we were testing how the program handled whitespace in the names, we realized that capitalization was being added to party names automatically (so “Democrat party” became “Democrat Party” in the output). This prompted the creation of this test.

outcome_only_recount_on_point.csv:
The program does not correctly handle the boundary values of the Outcome in riding_results.csv. The program outputs plurality instead of recount, violating R17.

This was generated directly through boundary value testing, as the plurality vs. recount decision is a natural point for on/off testing. The on point generated the error.

riding_clash_bad_integer.csv:
When leading zeros are inserted into the RidingNum entry, they are treated as different entries even though the number is the same (thus allowing the existence of different ridings with numbers “01” and “1”). This violates R8.

This was found through partition testing the possible inputs to the riding num.

riding_clash_caps.csv:
The program does not correctly handle spaces in the middle of Riding entries. Two entries differing only in the capitalization of the first letter behind a space are treated as two different entries, which violates either R13, or R8.

This issue was found through the same methodology as one_candidate_caps.

same_riding_same_party_space.csv:

Two entries of Party differing only in trailing whitespace are not treated as the same entry. This violates R10, rows with the same Riding and RidingNum can have equivalent Party entries.

This error was caught when doing partition testing of R13. We partitioned the inputs into classes differing only by capitalization, by whitespace, and by both.

minority_indep_equals_main_party.csv:
The boundary value between a result that generates government (majority) in the federal_results.csv, and government (minority) does not give the correct result. When the number of seats won by a party is equal to the number of seats won by independents, the program outputs a tie when R29 states that independents should not be factored into the calculation.

This error was found by boundary value testing of R29 on the number of seats won by each party.

bad_l_name.csv:
The program does not detect invalid characters in the LastName field. This violates R5.

This error was found using partition testing of R5. The input space was partitioned into entries with alphanumerical characters only, alphanumerical characters with spaces, entries containing special symbols “.”, ““, “-”, and entries containing invalid characters.

bad_fname_bad_error_message.csv:

When an invalid input is given in the FirstName column, the program correctly detects the existence of the error, but the row number given is incorrect. This violates R11.

This error was found during the partition testing of R5, where it was noted that no matter where the error was planted, the row number given in the error messages did not change.

vote_total_no_votes.csv:
A ValueError is thrown when all entries for Votes are 0 for all rows. This violates R12, as the input is valid.

This error was found using boundary value testing on the boundary point 0.