Integrating a Category-Partition Testing Tool with Combinatorial Interaction Testing Tool To Produce T-Way Adequate Test Frames

Andrew Graff

January 22, 2021

1 Introduction

(Start with stating that software has defects and those defects can be costly. That is, first you need to make a point why it is important to have defect/bug free software.) Software is the beating heart of a lot of technology today, and defects in that software can be costly. Some simply print funny characters to the terminal, while some can bring your production to a halt for hours or even days costing thousands and even millions of dollars. Or perhaps a security breach which exposes your intellectual property which could kill your business altogether. That is why testing is such an important step in the process of creating defect free software. Testing software can be a difficult task depending on the complexity of the software itself, what kind of inputs it takes, and how many different functions it can perform. And there are several different approaches in how to do the actual testing: black-box, white-box, static analysis to name a few. We will be focusing on black-box testing, and more specifically generating test cases that find the greatest amount of defects with a realistically executable number of test cases.

Using several testing methods, or combining them, will help to find the most defects possible. (Not sure how to tie this in. I was thinking maybe just focus on black-box testing and test case generation? Because in the next part, part of the problem statement is that TSL and CASA by themselves aren't good enough for a good selection of test cases) (This statement is too general. Instead it should bring up the problem you are trying to address. You can say that

various testing technique are complementary in their approaches and it would be beneficial to combine/integrate those components together.)

2 Problem Statement and Proposed Solution

(Here you can state the main point of what you are proposing – combining TSL (testing specification language), i.e., the front-end of the category partitioning method with combinatorial interaction testing, which has no useful front-end) It is difficult to engineer a set of tests that adequately tests a particular piece of software in an acceptable amount of time. Category partitioning is a useful black-box testing method that helps to test systematic design with test cases. It allows the user to divide the inputs of the software into categories of choices. TSL (testing specification language) is a tool that offers a good front-end for the category partitioning method and a formal descriptive language to specify the categories and choices. In addition, TSL will generate tests cases in an output file. However, it's weakness is that it will generate all possible permutations of program input as test frames, which may not run in a feasible amount of time.

Combinatorial interaction testing addresses the problem of the brute force approach of running all possible inputs by offering a method for analyzing and selecting a subset of test cases.

(Here you can start that combinatorial interaction testing addresses this problem by finding test cases where only certain category choices appear together) Similarly, combinatorial interaction testing is useful for defining a subset of tests that satisfy a t-way pairwise interaction using a model and constraints file. (Give three to four more sentence about how CASA tool does it)(Then talk about its front end – in fact its front-end is quite confusing that might introduce errors especially when expressing constraints. You might also state that in general an industry tester would not be able to use such tool — how many would be able to express a constraint in a conjunctive normal form? Or decode the resulting output file.)

(Here you should talk how you propose to solve this problem. The goal of this project is to ...) These two methodologies and tools(you never mentioned any tools – mention them in their corresponding paragraphs) are separate pieces of software that require the user to generate the input to both tools. This project takes on the task of combining(the user-friendly input/output components of one and integrate them into powerful test selection algorithm of another)

these two powerful methods and tools so that the user only needs to generate the category partition test specification, and an adequate coverage set of test frames is generated eliminating the need for the engineer to create the model or constraints for the combinatorial interaction testing.

3 Background

(Have a better transition, e.g., This section presents background information on both methodologies and their corresponding tools. In particular, ...) This project builds on the work of two teams that have come up with solutions to two different problems. Here we will explain each of those solutions and the tools created to support them.

The *category-partition method* uses a formal test specification language to generate test case descriptions. These test case descriptions would then be used to create an executable software test. So how do you generate the formal test specifications? Depending on the size of the software system to test, the engineer may need to break the program up into smaller testable blocks. For the purposes of this project, we will use a simple example of a zip command. The test specification file can be seen in Fig. 1.(I have not given much of feedback here since it does need substantial information on

- The idea of black-box testing – test selection based on the input space of a program
- Partition of the input space into equivalent classes
- Category partition method as a systematic way multi-level way of partition the input space
- TSL as a formal language to express those categories and paritions
- Existence of constraints between choices of different categories
- Resulting frames
- Refinement and coarsening of the spcification
- The tool description - lines of code and such

Also the example on Figure 1 is too large - try create something smaller, similar what we use in class – new link in a browser setting) (For next week, I will probably dig up the browser tab example and use that instead. I will have to take new screenshots of the resulting .tsl file, .citmodel file, and .constraints file)

```
Parameters:
        Function:
           compress.                [property Comp]
           decompress.              [property Decomp]

        Info:
           help.                    [single]
           license.                 [single]
           version.                 [single]
           checking.                [single]

        Ramblings:
           none.
           quiet.
           verbose.

        Compression:
          normal.                   [if Comp]
          fast.                     [if Comp]
          slow.                     [if Comp]
          n/a

        Suffix:
          no-suffix.
          myGZ.                     [property myGZ]

        Outputs:
          regular.                  [if !myGZ || !Decomp]
          stdout.

        Input file name:
          good file name.
          no such file.             [error]

        File state:
          file compr.               [if Decomp]
          file uncompr.             [if Comp]
          incorrect format.         [error]
```

Figure 1: Category partition input format

Combinatorial Interaction Testing - (Make sure you address the following

- The premise of t-way testing (defects occurs when few choices of categories interact, e.g., two choices)
- An overall approach of CIT test case selection.
- CASA tool and its input and output format examples, emphasize that it is not user-friendly. Moreover, a compound constraint among choices should

be converted into a conjunctive normal form (CNF), i.e., "AND" of "ORs" between choices

- Have some metric of this tool - lines of code and such

)

4 Preliminary Work

Some preliminary work has been done on this project in order to determine the feasibility of the idea to combine the *tsl* and *casa* tools into a single, easy to use tool. A goal of the project is to simplify and reduce the effort a user needs to put in to use the new tool. Source code was acquired for both tools and some initial analysis of the code was done in order to understand how they work and where some work may be needed to combine them. As it turns out, the *tsl* tool is written in basic C and *casa* is written in C++. Converting *tsl* to C++ would be beneficial for combining the code bases into a single program. So *tsl* was updated to support compilation with the g++ compiler.

As part of the simplification of the tool, only the category partition test specifications should need to be created by the user as a *.tsl* file. The tool will do the rest to generate the adequate coverage set of test frames. Since *casa* takes as input a *.citmodel* file, this preliminary version of *tsl* generates one based on the specification file passed in. The generation of the *.constraints* file will be created at a later time as part of the proposed work. In order to create the *.citmodel*, we need to be able to keep track of the parsed information from the specifications file.

(Reworded everything above in Preliminary Work with your suggestions. Still working on this part below. Having difficulty re-arranging the ideas.) (I think below you have all the information written well. However it would be better if instead of starting a paragraph talking about *what you've done and then* why you did it, you switch the order. State first what needs to be done: (1) establishing the map between categories and their choices and casa input values; (2) invoking casa on the translated input; (3) interpreting casa's output to produce frames)

In order to achieve this, a new struct was added to *tsl/structs.h* called *container* that keeps track of the list of non-single Choices as a vector of Choice struct pointers.(Make sure you explain in your background section what is Choice - by the way why is it capitalized?) This vector is used to lookup the

choices and categories later for printing the test frames. In addition, a *parent* pointer was added to the Choice struct so that the Category information can be referenced when performing the lookup into the Choice* vector.

Rather than output directly all possible test frames, a new function called *make_citmodel()* was written in *tsl/output.c* to write a *.citmodel* file used as an input to *casa*. The function *generator(Flag flags)* was also modified to call *make_citmodel()*, and then make a system call to casa passing in the *.citmodel* file for the input. Then, another function created called *process_output_file(string filename)* processes the file output by *casa* to generate the test frame final output. Ideally there should be no file io required, but this is a preliminary draft of the final solution and will be addressed in the final version of the project.

5 Proposed Remaining Work

Arguably the bulk of the work will be to translate the *properties* and *constraints* defined in the category partition test specifications into the constraints file required by *casa* to properly generate the adequate set of test frames rather than all possible combinations. (Maybe one of those days we can talk about how we can outline this approach in general)