ValidAlloy

A tool for validating a git alloy specification using test-case generation

José Pinheiro Tiago Guimarães

Métodos Formais em Engenharia de Software
Departmento de Informática
Universidade do Minho, Portugal

June 27, 2013

Overview

What is git?

- Git is a famous distributed revision control and source code management system;
- Branching is lightweight;
- Complete history and full revision tracking capabilities;
- Porcelain commands;
- Plumbling commands;
- Git data structures.

Some angry git comments

What a pity that it's so hard to learn, has such an unpleasant command line interface, and treats its users with such utter contempt.

10 things I hate about Git

Some angry git comments

*The man pages are one almighty "f*ck you". They describe the commands from the perspective of a computer scientist, not a user.*

10 things I hate about Git

Some angry git comments

(...) there are lots of "cheat sheets" floating around about how to use git tools - I think this is evidence that [...] many people have to write "GIT for mortals" pages...

Perl Mailing List

Git problems

- Git is complex;
- Git manual is obscure;
- Git commands are multi-purpose;
- Sometimes it doesn't work as expected.

Our solution: Create a **formal git model** to alleviate some of these problems.

Why a model?

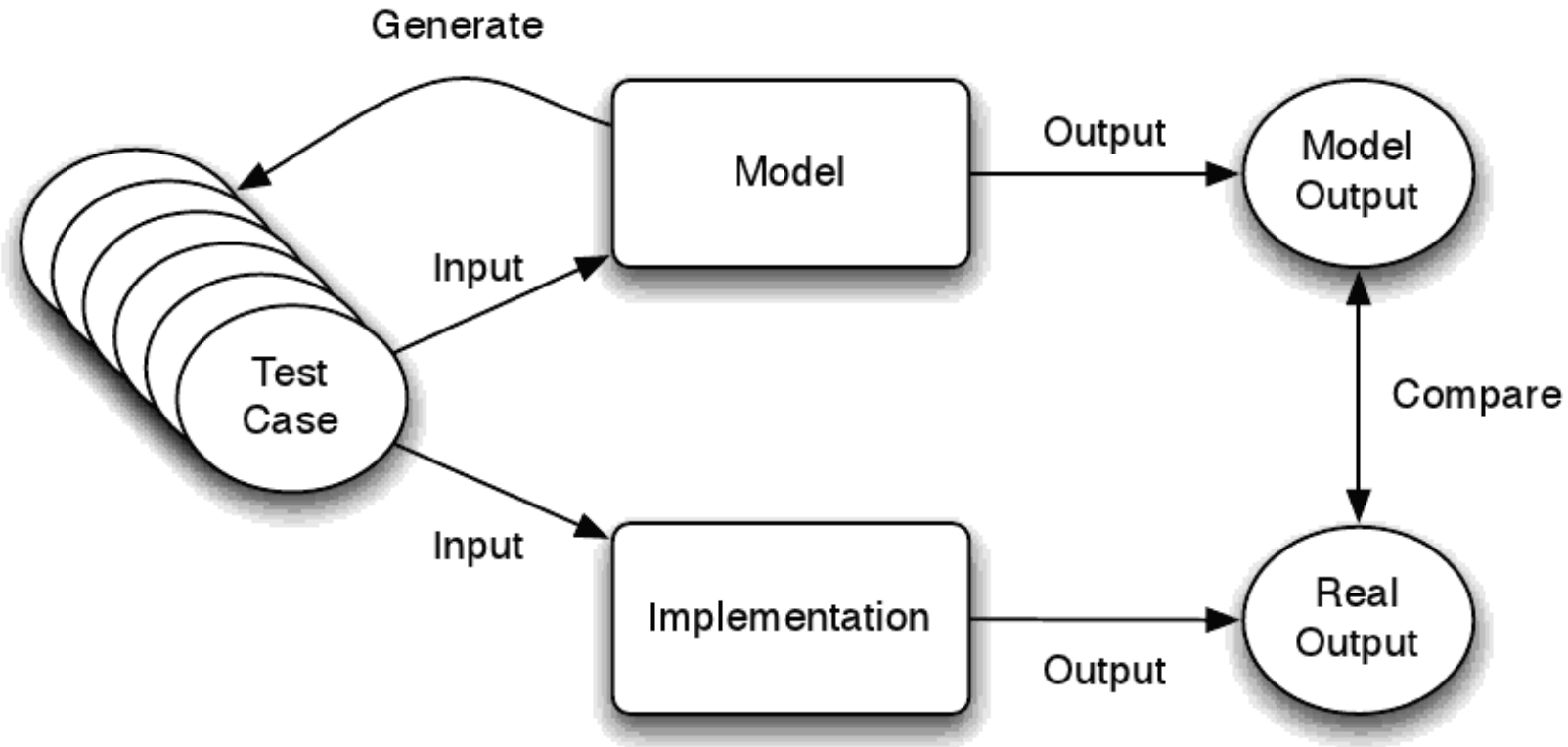
- Predict system behaviour;
- Verify properties of the system;
- Find Errors and bugs.

Main problem: Model accuracy.

How to get an accurate model?

- It is hard to validate a formal model;
- There are several solutions:
  - Derive model from code;
  - By trial and error;
  - Test case generation(our approach).

Model Validation Through Test Case Generation



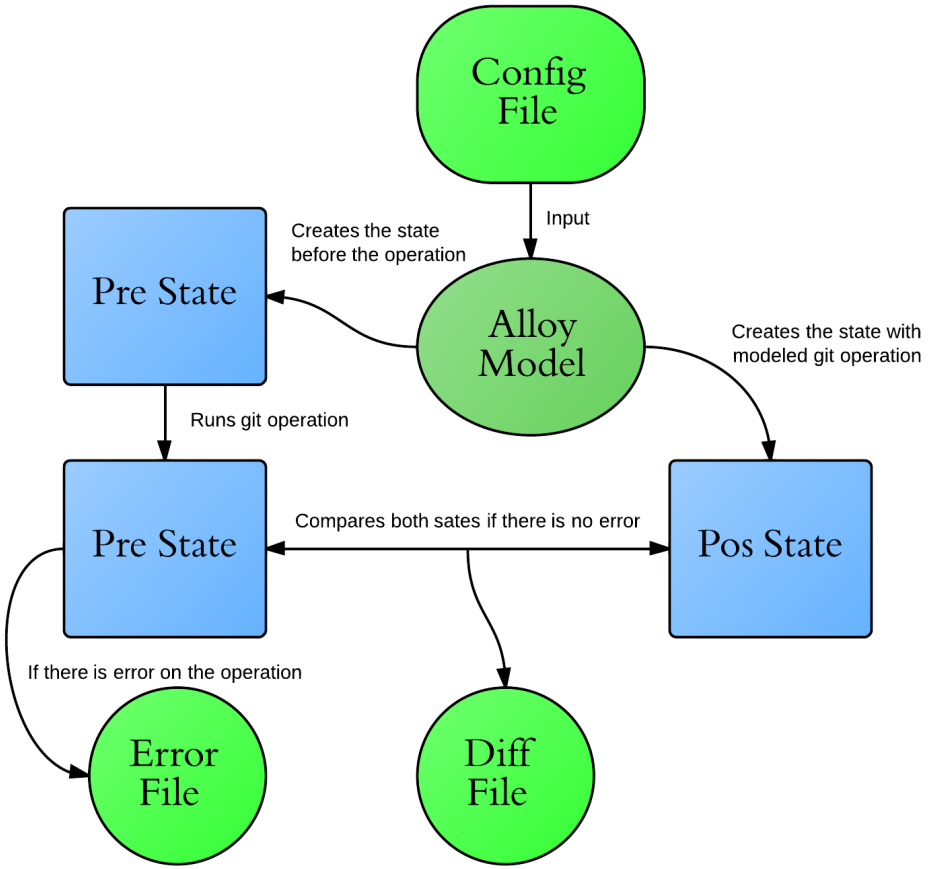
Case Study: Git

- Our goal was to create a testbench that implements model validition through test case generation;
- We started with a legacy Git model, and aimed to validate and improve it;
- Our testbench has **two main goals**:
  - **Test git itself;**
  - **Improve the git alloy model.**

Features

- Creates a git repo from the modelled instance;
- Associates any git modelled command with actual command;
- Compares the modelled version with the git command;
- Generic configuration file;
- Extensive logs;
- Handles git errors.

ValidAlloy


Config File
Input
Creates the state before the operation
Pre State
Alloy Model
Creates the state with modeled git operation
Runs git operation
Pre State
Compares both sates if there is no error
Pos State
If there is error on the operation
Error File
Diff File

Implementation Details

- ValidAlloy is implemented in Java 7;
- Uses the Alloy API;
- It depends on the git alloy model, this implies:
  - If you change some parts of the model, you have to change the testbench;
  - It only supports predicates written on that model.
- It requires as input a configuration file.

Demo

DEMO

How it Works

- Through Alloy API the testbench gets the solution generated from a run in the model;
- Use of Git plumbing commands to generate the Git objects;
- Creates a number of different runs from that solution into actual Git repositories.

Configuration File

- Use of a config file, to configure the testbench to generate the instances from the wanted commands;
- Use of Antlr(Another tool for language recognition), to generate a parser to parse the config file;
- Associates a given predicate that exists in the model, to an actual git command;
- Can be parametrized by a list of expected errors.

Example Configuration File

```
#p : path
pred rmNOP[#p]
scope for 4 but exactly 2 State
cmd git rm #p
errors ("has changes staged in the index")

runs 100
```

Unix Diff

How do we compare the pre and pos states?

- Using unix diff;
- When it finds differences it detects where and what where the differences in the filesystem;
- As result, our testbench only works on unix plataforms.

Problems Encountered

We encountered some problems during the development of validalloy:

- Timestamps in commit objects;
  - We resolved this problem, by fixing the same timestamp in all commit objects.
- Index is binary file and difficult to compare;
  - Instead of comparing the index file, we ignore it, and compare the output from git ls-files –staged instead.

Problems Encountered

- Errors that are expected while running the git operations;
  - We added to the config file a list of expected errors.
- Git doesn't report some errors as errors, but to standard input;
  - We could not find a solution to this problem, but you can manually check the logs, and see if the output from the operation was actually an error.

Git Model

One of the main purposes of the testbench is to help develop a git model, although we didn't get a full model, we still achieved some results:

- git add;
- git remove;
- git commit;
- git branch;

Git add

- We found no differences between the pre state and post state.
- We believe that with the current model we will not find any differences.

Git remove

- What does git remove do?
  - Removes the entry of the file from the index;
  - Deletes it from the file system;
- Unmentioned in the man pages:
  - Deletes empty directories.

Git remove

- Git remove is supposed only to work with these pre-conditions:
  - The files being removed have to be identical to the tip of the branch;
  - No updates to their contents can be staged in the index.

Git remove

- Our approach to git rm revolved around negating some of git rm man pages pre-conditions:
  - Run git rm with both pre-conditions holding true;
  - Negated one alternatively while the other held true;
  - Test with both pre-conditions being false.
- This will create lots of errors that are expected because of negating the pre-conditions;
- Our solution was to implement the list of expected errors in the configuration file.

Git remove

We actually found a bug with this command and will now show you a trace of this bug.

Git remove

- We asked at git@vger.kernel.org, if this was the correct behaviour from git rm;
- In the first reply, they suggested a patch for it.

Conclusion

- Git is not nearly perfect;
- Model validation through test case generation works:
  - To validate a model;
  - To create tests to the actual software.
- But there are still some problems:
  - Model depth limits the obtained results.

Possible future Work

- Finish Model:
  - git merge;
  - git commit;
  - git branch.
- Update documentation;
- Create a generic framework, to work with any alloy model.