Project 4: Cause-Effect Chains

In this project, you will write a tool that isolates failure-inducing program states. Your task is to implement the tool and apply it to the XMLProc parser from Project 1[1].

Isolating Cause-Effect Chains (60% of the points)

Overview

Create a tool that isolates cause-effect chains by comparing a passing and a failing program run. You will implement a simplified version of the HOWCOME tool (see Zeller, "Isolating Cause-Effect Chains from Computer Programs" (ICSE 2002)[2]). The main file of your tool should be called `howcome.py` and it should be callable as follows:

```
python howcome.py UNIT_TEST PASSING_TEST FAILING_TEST LOCATIONS_TXT
```

`UNIT_TEST` is the name of the test case class that contains the passing and failing tests to be compared (see the Python's `unittest` framework for details[3]). `PASSING_TEST` is the name of a passing test from the unit test above and `FAILING_TEST` is the name of a failing test from the unit test above. `LOCATIONS_TXT` is the name of the file containing locations in the test runs at which the two program states should be compared and failure-inducing state differences should be isolated. Each line in this file should have the following format:

```
RELATIVE_FILE_NAME:LINE_NUMBER:EXECUTION_NUMBER
```

where `RELATIVE_FILE_NAME` is the path to the file relative to the directory, from which your tool was called, `LINE_NUMBER` is the line number of the line, at which to isolate the state differences, and `EXECUTION_NUMBER` is the number specifying, during which execution of the given line to isolate the state differences.

A sample call to isolate the differences between two runs of the XMLProc parser looks as follows:

```
python howcome.py XMLProcTests.TestXMLProc testPass testFail locs.txt
```

This means that there is an `XMLProcTests.py` file in the current directory and that it contains a test case called `TestXMLProc`. This test case contains two test methods: `testPass` and `testFail`. `locs.txt` specifies the locations, like in this example:

[1]http://www.st.cs.uni-saarland.de/whyprogramsfail/code/xmlproc-0.70a.zip
[2]http://www.st.cs.uni-saarland.de/papers/fse2002/p201-zeller.pdf
[3]http://docs.python.org/library/unittest.html

```
xmlproc/xml/parsers/xmlproc/xmlutils.py:185:1
xmlproc/xml/parsers/xmlproc/xmlproc.py:85:4
```

This locations file specifies that failure-inducing state differences should be isolated at two points in time during the program executions:

- The *first* time the execution reaches the line 185 of the file `xmlproc/xml/parsers/xmlproc/xmlutils.py`.
- The *fourth* time the execution reaches the line 85 of the file `xmlproc/xml/parsers/xmlproc/xmlproc.py`.

Your tool should output the failure-inducing state differences on the standard output. You have freedom in choosing the output format, but be sure to output all the locations and their respective failure-inducing state differences (deltas) in a human-readable form.

Test Cases

Use your tool on the `middle.py` program to do the first steps and make sure everything works as expected. The simplicity of this program should make debugging your tool easier.

Apply your finished tool on the XMLProc parser from Project 1. Your test case should contain two tests: one passing, with XMLProc being invoked on the input `<?m?><!DOCTYPEl PUBLIC""""><a>&#10;</a>`, and one failing, with XMLProc being invoked on the input `<?m?><!DOCTYPEl PUBLIC""""><a>&#xa;</a>`. Invoke XMLProc using a similar approach to the one you used in the 1st Project. However, do not write the inputs to files and call a `parse_resource` method, but rather use methods `reset`, `feed`, and `flush` to pass a string to the XMLProc parser. This facilitates isolating failure-inducing state differences. If you have trouble with this, ask your tutor for help. Use the following locations file to isolate failure-inducing state differences in the two test runs specified above:

```
xmlproc/xml/parsers/xmlproc/xmlutils.py:185:1
xmlproc/xml/parsers/xmlproc/xmlproc.py:85:4
xmlproc/xml/parsers/xmlproc/xmlproc.py:391:1
xmlproc/xml/parsers/xmlproc/xmlproc.py:400:1
```

What are the failure-inducing state differences? Are they helpful in locating the part of the input that causes the defect? Are they helpful in locating the defect? If so, what is the fix that will make the XMLProc parser work correctly on both of the inputs above?

Approaching the Problem

You can use the website `http://www.whyprogramsfail.com/project3.php` to *guide* you in implementing the tool specified above, but be sure to take into account the following things:

- Your tracing function should operate on specific *lines*, not *functions*. Moreover, remember to take the execution number into account (see the specification of the locations file above).
- To copy state, use the function `copy.deepcopy`. Only if the value to be copied is a module or if this function raises an exception should you simply return the original value without copying it. Be sure not to use `copy.copy`. Read the specification of the `copy.deepcopy` function carefully: make sure you understand how to make it treat aliases appropriately when copying many values in a row. If you have trouble with this, ask your tutor for help.
- Make sure your tool works as specified above. Do *not* simply implement a `debug` function to be called from outside. This also means that you do not have to provide a `README` file.
- Do *not* implement any of the advanced methods described by the website.
- You do not have to run your tool on the test cases described by the website, but keep in mind that it may help you make sure your tool works correctly.

Some parts of this project are tricky and can be difficult to get done right. Be sure to start sufficiently early and contact your tutor if you are stuck somewhere during this process.

Finding Compatible States (30% of the points)

Extend your tool such that it collects states at all locations during the executions of the passing and failing test and discovers pairs of compatible states. Two states from different runs are compatible if it is possible to apply isolating failure-inducing state differences on them. More precisely, two states are compatible if their backtraces have the same length and locations of all respective frames in the backtraces are identical. Your extended tool should be runnable just like the base version, but it should additionally look for all pairs of compatible states and output them to the file `compatible_states.txt`. Each line in this file should have the following format:

```
PASSING_RUN_LOCATION == FAILING_RUN_LOCATION
```

where both `PASSING_RUN_LOCATION` and `FAILING_RUN_LOCATION` are locations with the same format as those in the locations file. Sample output snippet looks like this:

```
xmlproc/xml/parsers/xmlproc/xmlapp.py:146:1 == xmlproc/xml/parsers/xmlproc/xmlapp.py:146:1
xmlproc/xml/parsers/xmlproc/xmlapp.py:169:1 == xmlproc/xml/parsers/xmlproc/xmlapp.py:169:1
```

Identifying Cause Transitions (10% of the points)

How can compatible states calculated in the part above be used for automatic identification of cause transitions, as described in the paper of Cleve and Zeller, "Locating

Causes of Program Failures" (ICSE 2005)[4]? Sketch an algorithm for doing this. You do not have to implement it.

Submission

Submit your solution as a **.zip** file to `wasylkowski@cs.uni-saarland.de` with `[Project 4]` in the subject. Provide your full name and matriculation number in the body of the email. Be sure to adhere **exactly** to the input and output specification as given above. Include the answers to the questions stated in the assignment in the body of the email. If you have any questions or need clarification regarding the assignment, send an e-mail to `wasylkowski@cs.uni-saarland.de`.

The deadline is **2009-02-13 18:59**.

[4] http://www.st.cs.uni-saarland.de/papers/icse2005/p88-cleve.pdf