1 Tool usage

The eVolCheck can be run from a command line as well as using the Eclipse plug-in. Its Linux binaries, benchmarks used for evaluation, a tutorial explaining how to use eVolCheck and explanation of the most important parameters are available on-line for other researchers[1].

The following shows the example of usage of eVolCheck from a command line [2]:

1. Create a model of the base version of the program by running the goto-cc compiler. Choose one of the *_orig.c files in examples directory, for example by typing:

```
~/evolcheck$ ./goto-cc examples/valid/change_valid_orig.c
   -o examples/valid/change_valid_orig.out
```

The file examples/valid/change_valid_orig.c is the input source code, while examples/valid/change_valid_orig.out is the resulting goto-binary. Note, that the upgrade checking environment should be prepared for analysis by cleaning the repository with ~/evolcheck$ rm __summaries __omega before performing this step.

2. Run eVolCheck to perform the initial bootstrapping check of the program (parameter --init-upgrade-check):

```
~/evolcheck$ ./evolcheck --init-upgrade-check --unwind 10
   examples/valid/change_valid_orig.out
```

Note that the parameter --unwind <N> is optional and specifies the maximal number of unwindings of each loop.

3. Check the eVolCheck outputs. The following message at the end of the eVolCheck output indicates either that the program is safe:

```
ASSERTION(S) HOLD(S).
```

or that the program is buggy:

```
ASSERTION(S) DO(ES)N'T HOLD.
A real bug found.
```

In the latter case, a corresponding error trace manifesting the bug is part of the output as well. After a successful bootstrapping check, the summaries and their mapping the calltree are created and stored for the subsequent upgrade checks, files __summaries and __omega respectively.

4. When the program is upgraded, goto-binary model of the new version is created again using the goto-cc compiler. Run it for the file corresponding *_upgr.c file chosen in Step 2:

```
~/evolcheck$ ./goto-cc examples/valid/change_valid_upgr.c
             -o examples/valid/change_valid_upgr.out
```

5. With the goto-binary model of the new version of the program, the actual upgrade check is performed (parameter --do-upgrade-check <file>) as follows:

[1] www.verify.inf.usi.ch/evolcheck

[2] the running example can be found at http://www.inf.usi.ch/phd/fedyukovich/evolcheck.lin32.tar.gz

```
~evolcheck$ ./evolcheck --do-upgrade-check examples/valid/change_valid_upgr.out
            --unwind 10 examples/valid/change_valid_orig.out
```

6. Check the `eVolCheck` output. There are several possible cases. Either the two programs have identical models, i.e., no or only simple syntactical changes occurred (examples for this case are located in the `examples/ident` folder), resulting in the following output:

```
The program models are identical.
```

Or the upgraded program really changed but it remains correct (examples are located in the `examples/valid` folder), resulting in the following message for each checked function summary:

```
... summary was verified.
```

Or the upgraded program is buggy (examples are located in `examples/not_valid`). The corresponding output contains the following message for the summary of the function `main`: ‘

```
Old summary is no more valid.
...
summary cannot be renewed. A real bug found.
```

7. Additional information about the usage of the tool can be found simply by typing

```
~/evolcheck$ ./evolcheck --help
```

Eclipse plug-in. Nowadays, IDEs form an essential part of software development tool chains. Therefore, we integrated `eVolCheck` into `Eclipse`, which is one of the most widely used IDE. Our plug-in hides some of the implementation details and provides much more comfort compared to the command line tool. As expected, the actual use of the plug-in follows the command line scenario.

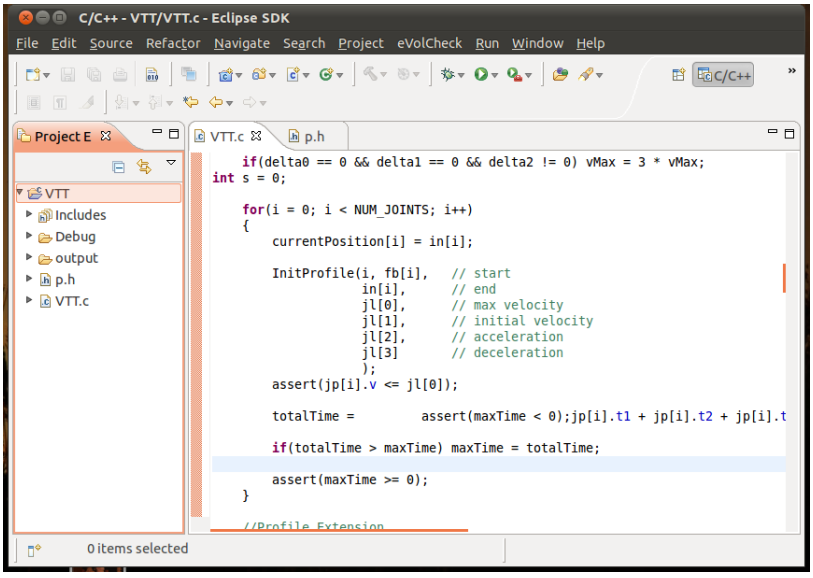

Figure 1: Eclipse developing prospective

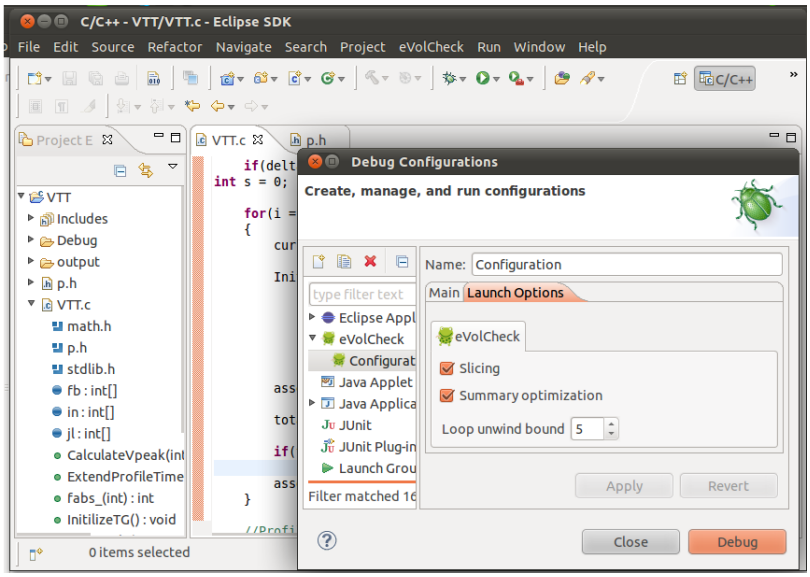

Figure 2: eVolCheck configuration window

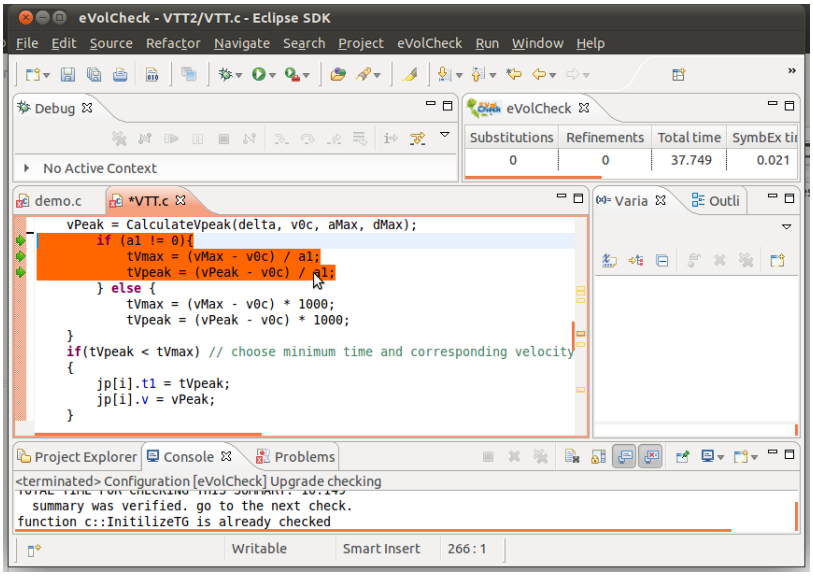

Figure 3: eVolCheck invokes goto-diff (changed lines are highlighted)

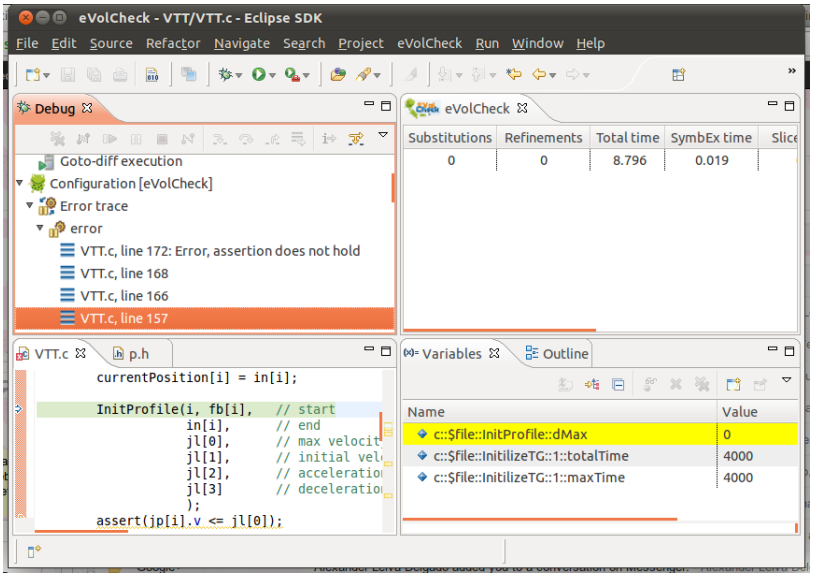

Figure 4: eVolCheck error trace

1. The user develops a base version of the program. In order to specify properties, the assertions should be placed in the code (Fig. 1) or generated automatically by the tool. The examples of of the default properties are division by zero, pointers dereferencing, array out-of-bounds checks.

2. The user opens the *Debug Configurations* window and chooses the file(s) to be checked and specifies the unwinding bound (Fig. 2). `Eclipse` then automatically creates the model (goto-binary) from the selected source files and keeps working with it.

3. The plug-in searches for the last safe version of the current program (goto-binary created from the same selection of source files). If no such a version is found, it performs the initial bootstrapping check. Otherwise, plug-in restores the summaries and outdated goto-binary from the subsidiary storage. `eVolCheck` then identifies the modified code by comparing call trees for both the current and the previous versions. The modified lines of code are marked (Fig. 3) for the user review.

4. Then the localized upgrade check is performed. If it is unsuccessful, the plug-in reports violation to the user and provides an error trace (Fig. 4). The user can traverse the error trace line by line in the original code and see the valuation of all variables in all states along the error-trace. If desired, the user fixes the reported errors and continues from Step 3.

5. In case of successful verification, the positive result is reported (Fig. 5). The plug-in stores the set of valid and new summaries and the goto-binary in the subsidiary storage. In addition, graphical visualization of the change impact in the form of a colored call-tree is available (Fig. 6).

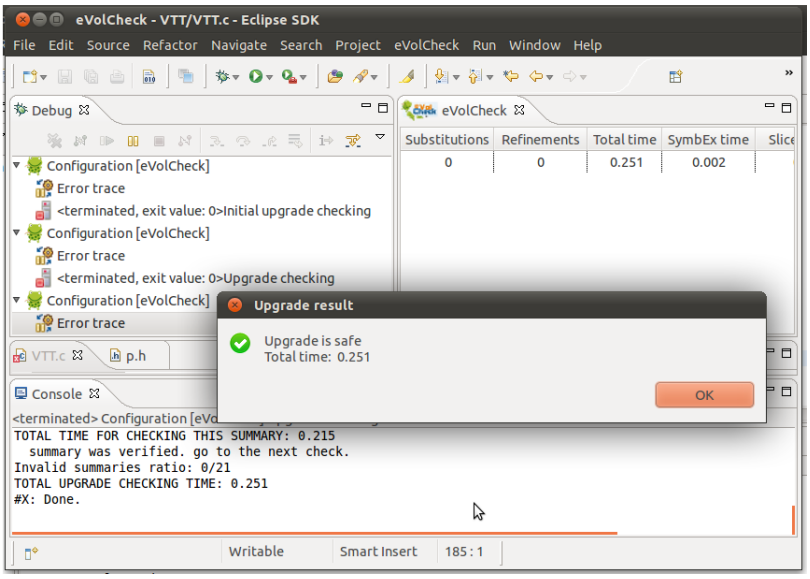

Figure 5: eVolCheck successful verification report

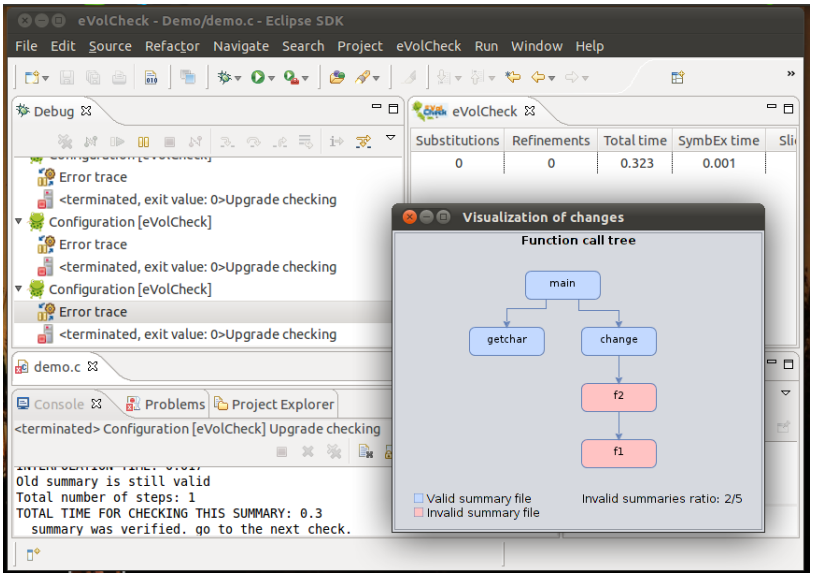

Figure 6: eVolCheck change impact