KT Advance User Manual

Kestrel Technology, LLC

November 24, 2017

Contents

1 Quick Start **3**

1.1 System Requirements 3

1.1.1 Platform: MacOSX or Linux 3

1.1.2 Utility Programs 3

1.1.3 Other Dependencies 4

1.2 Organization 4

1.3 General Use Guidelines 4

1.4 Getting Started 5

2 Running Test Cases **6**

2.1 Kendra 6

2.1.1 Organization 6

2.2 Running the Tests 6

1 Quick Start

1.1 System Requirements

1.1.1 Platform: MacOSX or Linux

The KT Advance C Analyzer consists of three components that may be run on different platforms:

1. **Parser:** A Mac/Linux executable (parseFile) that takes as input a preprocessed C source file and produces a set of xml files that precisely represent the semantics of the C source file. This program is an extension of the CIL parser front end, developed by George Necula, at UC Berkeley, an now maintained by INRIA in France. Except for very simple programs (typically programs that do not include any standard libraries), it is recommended to run the parser on a Linux platform, as the parser pulls in definitions from the standard header files resident on the system.
2. **C Analyzer:** A Mac/Linux executable (ktadvance) that takes as input the semantics files produced by the parser as well as analysis results files, if available, and produces a set of xml files that hold analysis results. This executable will be wrapped in a license manager to protect the contained intellectual property. The C Analyzer operates solely on the semantics files produced by the parser without any other dependencies on the local system, and thus it can be run equally well on Mac or Linux.
3. **PyAdvance:** Python code, provided as source code to licensed users, that performs linking and provides various analyzer invocation, integration, and reporting services. All reporting scripts (scripts whose name typically start with `chc_report_` or `chc_show_` rely only on python code and thus can (theoretically) be run on any platform that has python installed, including Windows platforms. Thus if analysis has been performed on a server and the results saved, the analysis results can then be viewed and queried by anyone with access to these results.

1.1.2 Utility Programs

The front-end parser makes use of the utility `bear` to record and reply the actions performed by the Make file when compiling an application. This utility is usually available via a package manager.

1.1.3 Other Dependencies

The analyzer and python scripts make use of jar files; being able to extract these requires a working Java installation.

1.2 Organization

The `ktadvance` repository has three top directories:

1. `advance:` python scripts and programs to run the analysis and view the results; Section **??** describes these scripts and programs in more details.
2. `doc:` KT Advance User Manual (this document) and a Reference Manual that explains the analysis approach and describes in detail the data representation of all intermediate data artifacts and analysis results data.
3. `tests:` regression tests and other test cases, several of which have been pre-parsed and are ready for analysis. The reason for having pre-parsed applications is to provide reference applications that enable longitudional study of analyis performance. Several of the test directories have dedicated scripts for parsing and analysis, as described in Section 2.

1.3 General Use Guidelines

The analysis consists of three phases that may be performed on different platforms.

1. **Parsing:** This phase takes as input the original source code, a Makefile (if there is more than one source file), and, in case of library includes, the library header files resident on the system. This phase produces as output a set of xml files that completely capture the semantics of the application, and are the sole input for the Analysis phase.
   Because of the dependency on the resident system library header files it is generally recommended to perform this phase of the analysis on a Linux system, because of its more standard library environment than MacOSX (the CIL parser also may have issues with some of the Darwin constructs on MacOSX).
   For several of the test cases in `tests/sard/kendra` and for all of the test cases in `tests/sard/zitser` and `tests/sard/juliet_v1.3` the parsing step has already been performed (on Linux) and the resulting artifacts are checked in in files named `semantics_linux.tar.gz`. These gzipped tar files contain all xml files necessary for the analysis, and thus to analyze these files the parsing phase can be skipped altogether.
2. **Analysis:** This phase takes as input the xml files produced by the parsing phase. As long as the source code is not modified, the analysis can be run several times without having to repeat the parsing step. The Analysis step can be run on either MacOSX or

Linux, independently of where the parsing step was performed, as it operates solely on the xml files produced and is not dependent on any external programs or library headers.

3. **Viewing Results:** All analysis results are saved in a directory with the name `semantics` in the analysis directory. Various reporting scripts are provided to process and view these results. These scripts rely only on python code and thus can be run on any platform once the analysis results have been produced.

1.4 Getting Started

All interactions with the KT Advance C Analyzer are performed via python scripts from the command line. All scripts have been tested to work with python 2.7. An effort has been made, however, to have all python code also compliant with python 3.x.

All scripts to interact with the analyzer are in the directory `ktadvance/advance/cmdline`. This directory has a few subdirectories with scripts dedicated to some of the test sets in the tests directory, as follows:

- **kendra:** scripts to analyze and report on the test cases in `tests/sard/kendra`;
- **zitser:** scripts to analyze and report on the test cases in `tests/sard/zitser`;
- **juliet:** scripts to analyze, score, and report on the test cases in `tests/sard/juliet_v1.3`.

Two other subdirectories have scripts to parse, analyze, and report on any c file or c application:

- **sfapp:** scripts to parse and analyze an application that consists of a single c file that be compiled directly with gcc (without a Make file).
- **mfapp:** scripts to parse and analyze an application that comes with a Makefile. It is expected that the Makefile exists (that is, a configure script has already been run, if necessary).

Section 2 provides a detailed description of the scripts available in the test-specific cmdline directories and Section **??** describes the scripts for the general files and applications. In general, user scripts start with the prefix `chc_` followed by some verb that indicates the action performed; most scripts have a –help command-line option that describes the arguments expected.

2 Running Test Cases

2.1 Kendra

The `ktadvance/tests/sard/kendra` directory contains a collection of very small test programs retrieved from the NIST Software Assurance Reference Dataset (`samate.nist.gov`). These programs are a subset of the collection of test cases developed by Kendra Kratkiewicz []. These test cases serve as a good first illustration of the KT Advance analysis approach and presentation of results. Section 2.2 has step-by-step instructions how to run these tests and some comments on particular cases.

2.1.1 Organization

The test cases are organized in groups of four related test cases, where the first three test cases have a given vulnerability with varying magnitude of overflow and in the fourth case that vulnerability is fixed (or absent). The names of the tests refer to the sequence numbers in the SARD repository, and the name of the group refers to the sequence number of the first test. For example, the test group id115Q contains the test cases id115.c, id116.c, id116.c, and id117.c .

The kendra tests are also used as regression tests for generating and discharging proof obligations. Each test directory has a reference file [testname].json (e.g., id115Q.json) that lists all proof obligations and their expected proof status, against which the analysis results are checked after each test run.

2.2 Running the Tests

Set the `PYTHONPATH` environment variable (or adapt for a different location of the ktadvance directory):

```
> export PYTHONPATH=$HOME/ktadvance
```

To see a list of test sets currently provided in the `tests/sard/kendra` directory:

```
> cd ktadvance/advance/cmdline/kendra
> python chc_list_kendratests.py
```

To run the analysis of a test set (staying in the cmdline/kendra directory):

```
> python chc_test_kendra_set.py id115Q
```

(or any other of the test set names provided in the list displayed earlier.) This will print the summary results for the four test programs included:

```
File          Parsing        PPO Gen        SPO Gen      PPO Results    SPO Results
-------------------------------------------------------------------------------------
id115.c          ok             ok             ok             ok             ok
id116.c          ok             ok             ok             ok             ok
id117.c          ok             ok             ok             ok             ok
id118.c          ok             ok             ok             ok             ok
```

indicating that all stages ran without error, all proof obligations were correctly generated and the analysis results are as expected. To see more of the output while running the test, add –verbose as a command-line option.

To see which proof obligations are included in the test cases, with line numbers and expected proof status:

```
> python chc_show_kendra_test.py id115Q

id115.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id116.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id117.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      violation
      56  cast                   safe
id118.c
  main
      56  index-lower-bound      safe
      56  index-upper-bound      safe
      56  cast                   safe
```

When a test case has been analyzed the analysis results are saved in the `semantics/ktadvance` directory and are available for inspection and to reporting scripts. To see a full report, including code, proof justifications, and summary for an individual test file (note the filename):

```
> python chc_report_kendratest_file.py id115.c

Function main
-------------------------------------------------------------------------------
50  int main(int argc, char *argv[])
-------------------------------------------------------------------------------
Api:
  parameters:
    int[] argc
    ((char[] *) *) argv

  -- no assumptions

  -- no postcondition requests

  -- no postcondition guarantees

  -- no library calls
-------------------------------------------------------------------------------
Primary Proof Obligations:
-------------------------------------------------------------------------------
51  {
52    char buf[10];
53
54
55    /*  BAD  */
56    buf[4105] = 'A';
-------------------------------------------------------------------------------
<S>    1     56  index-lower-bound(4105) (safe)
                  index value 4105 is non-negative
<*>    2     56  index-upper-bound(4105,bound:10) (violation)
                  index value 4105 violates upper bound 10
<S>    3     56  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
-------------------------------------------------------------------------------

Primary Proof Obligations
functions    stmt   local     api    post  global    open   total
-----------------------------------------------------------------
main          3       0       0       0       0       0       3
-----------------------------------------------------------------
total         3       0       0       0       0       0       3
percent  100.00    0.00    0.00    0.00    0.00    0.00


Proof Obligation Statistics for file id115
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Primary Proof Obligations
                              stmt   local     api    post  global    open   total
-------------------------------------------------------------------------------------
cast                             1       0       0       0       0       0       1
index-lower-bound                1       0       0       0       0       0       1
index-upper-bound                1       0       0       0       0       0       1
-------------------------------------------------------------------------------------
total                            3       0       0       0       0       0       3
percent                     100.00    0.00    0.00    0.00    0.00    0.00
```

For each line of code that has associated proof obligations the report shows the proof obligation predicate, whether it is valid (safe, indicated by <S>) or violated (indicated by <*>), and the reason for the assessment. For the safe case, id118.c, the output shows that the proof obligation for the upper bound is indeed safe.

```
> python chc_report_kendratest_file.py id118.c

....

-------------------------------------------------------------------------------
Primary Proof Obligations:
-------------------------------------------------------------------------------
51  {
52    char buf[10];
53
54
55    /*  OK  */
56    buf[9] = 'A';
-------------------------------------------------------------------------------
<S>    1     56  index-lower-bound(9) (safe)
                  index value 9 is non-negative
<S>    2     56  index-upper-bound(9,bound:10) (safe)
                  index value 9 is less than bound 10
<S>    3     56  cast(chr('A'),from:int[],to:char[]) (safe)
                  casting constant value 65 to char
-------------------------------------------------------------------------------

....
```

For a list of all proof obligation predicates and their meaning, please see the KT Advance Reference Manual in this directory.

To analyze all kendra test cases run

```
> python chc_test_kendra_sets.py
```

If all of them complete all analysis results are available for inspection and reporting. Below we highlight some special cases.