Features Pricing owner/repository English **Sign up** Log in

khooyp
Otter

ACTIONS

Fork

NAVIGATION

Overview

Source

Commits

Branches

Pull requests

Issues

Downloads

Overview

HTTPS https://bitbucket.org/khooyp/otte

Last updated 6 Branches

5 Tags

Language

Access level 1 Fork

12 Watchers

Unlimited private and public hosted repositories. Free for small teams!

Sign up for free

Recent activity

khooyp/Otter
Repository watched
twinofmunin · 2013-04-25

1 commit
Pushed to khooyp/Otter
cf06305 Quote versionString.stamp che…
Yit Phang Khoo · 2013-04-08

khooyp/Otter
Repository watched
zhuyue · 2013-03-15

khooyp/Otter
Repository watched
zero out · 2013-01-26

Introduction

Otter is a pure, source-level symbolic executor for C that can be used to test programs. Using Otter, a program may be given *symbolic inputs*, and Otter will explore all possible executions of the program for any value of those inputs to find bugs such as invalid pointer dereferences, buffer overflows, and other run time errors as well as verify assertions in the source code.

Quick-start

```
make
make test-otter
otter/otter.pl <file to test>
```

Building and running Otter

Detailed instructions for building, running, developing and testing Otter are documented in `otter/README.md`.

To build Otter for the first time, run `make` in the directory this file is in. This will configure all the sub-directories correctly and build them in the correct order.

Subsequently, after making any changes to the source code in any directory, you should run `make` from this directory again, and the build system should automatically reconfigure and/or rebuild each sub-directory as necessary.

You may also run `make` from individual sub-directories, but note that only that sub-directory will be reconfigured/rebuilt.

Running `make clean` and `make distclean` in this directory will recursively run those commands in the sub-directories to delete built products as well as configuration files respectively.

Testing Otter

To run Otter's test suite, run `make test-otter` in this directory. Alternatively, run `make test` from the `otter` directory.

`80f9f0c` - Add NSF grant CNS-090541…
Commit pushed to khooyp/Otter
Yit Phang Khoo · 2013-01-20

`80f9f0c`
Changeset stripped from khooyp/Otter
Run `hg strip 80f9f0cce741` on your local c
opy
Yit Phang Khoo · 2013-01-20

`80f9f0c` - Add NSF grant CNS-090541…
Commit pushed to khooyp/Otter
Yit Phang Khoo · 2012-12-03

`c9cc2f2` - Fix build failure due to new O…
Commit pushed to khooyp/Otter
Yit Phang Khoo · 2012-08-27

`f6024ce` - Add missing subdirectories f…
Commit pushed to khooyp/Otter
Yit Phang Khoo · 2012-07-05

`ea39678` - Reorder library link flags to s…
Commit pushed to khooyp/Otter
Yit Phang Khoo · 2012-07-05

Running Otter

To run Otter, use the command `otter/otter.pl [flags] <input files>`.

For more information, please run `otter/otter.pl --help` and refer to `otter/README.md`.

Installing Otter

It is not currently possible to install Otter; please run it from the source directory.

Directory layout

The source code to Otter is in the `otter` directory.

Otter uses customized versions of a number of third-party libraries, which are included in the following directories:

- `cil` - CIL analysis framework for C
- `stp` - STP theorem prover
- `ocamlgraph` - graph library for Ocaml
- `delimcc` - delimited continuations library for Ocaml
- `newlib-1.19.0` - standard C library

The `newlib-1.19.0` directory also contains a POSIX implementation for Otter under `newlib-1.19.0/otter`.

The `experiments` directory contains various experiments that have been run to evaluate Otter.

The `cilqual` directory contains CilQual and Mixy, which is another program analysis tool that uses Otter as a library. *Note that CilQual/Mixy does not build with the current revision of Otter; a working revision may be found in the source code repository under the tag `PLDI2010`.*

Mailing list

If you have any questions about Otter, please send an email to otter-dev@cs.umd.edu.

Publications

Confererence papers

Directed Symbolic Execution. Kin-Keung Ma, Khoo Yit Phang, Jeffrey S. Foster, and Michael Hicks. In Eran Yahav, editor, The 18th International Static Analysis Symposium (SAS), volume 6887 of Lecture Notes in Computer Science, pages 95–111, Venice, Italy, September 2011. Springer-Verlag Berlin / Heidelberg.

Mixing Type Checking and Symbolic Execution. Khoo Yit Phang, Bor-Yuh Evan Chang, and Jeffrey S. Foster. In Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 436–447, Toronto, Canada, June 2010.

Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems. Elnatan Reisner, Charles Song, Kin-Keung Ma, Jeffrey S. Foster, and Adam Porter. In Proceedings of the 32nd International Conference on Software Engineering (ICSE), pages 445–454, Cape Town, South Africa, May 2010.

Technical reports

MultiOtter: Multiprocess Symbolic Execution. Jonathan Turpie, Elnatan Reisner, Jeffrey S. Foster, and Michael Hicks. Technical Report CS-TR-4982, Department of Computer Science, University of Maryland, College Park, August 2011.

Directed Symbolic Execution. Kin-Keung Ma, Khoo Yit Phang, Jeffrey S. Foster, and Michael Hicks. Technical Report CS-TR-4979, Department of Computer Science, University of Maryland, College Park, April 2011.

Mixing Type Checking and Symbolic Execution (Extended Version). Khoo Yit Phang, Bor-Yun Evan Chang, and Jeffrey S. Foster. Technical Report CS-TR-4954, Computer Science Department, University of Maryland, College Park, March 2010.

Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems. Elnatan Reisner, Charles Song, Kin-Keung Ma, Jeffrey S. Foster, and Adam Porter. Technical Report CS-TR-4946, Computer Science Department, University of

Maryland, College Park, December 2009.

Acknowledgements

The development of Otter was supported by the following grants:

- DARPA ODOD.HR00110810073;
- NSF CCF-0541036, CCF-0915978, CCF-0346982, CCF-0430118, CCF-0524036, CCF-0811284, CCR-0205265, and CNS-0905419.