# KLEE-Reach: a KLEE version for Guiding Symbolic Execution with A-star
## User Manual

Guilhem Ardouin
guilhem.ardouin@enseirb-matmeca.fr

March 31, 2024

## Contents



LABORATOIRE BORDELAIS DE RECHERCHE EN INFORMATIQUE LaBRI

# 1 Introduction

Symbolic execution is an exploration technique designed to analyse a program to determine the inputs that lead to a specific part of a program. This approach usually doesn't scale on real-size programs, hence the order of exploration is important. In response to this problem, it is possible to guide symbolic execution using a strategy based on A-star [2]. This approach has been implemented in KLEE [1] and allows the user to use two new search heuristics for reachability analysis.

This document presents how to build this addition to KLEE and how to use the implementation of these new exploration strategies directly in KLEE. For implementation details, please refer to the developer report.

# 2 Installation

This section explains how to build a version of KLEE with the new search heuristics and reachability analysis tools.

## 2.1 Requirements

To install KLEE-Reach, you need to satisfy all the KLEE dependencies: `http://klee-se.org/build-llvm13/`. However, there are addional dependencies:

- Boost C++ library: `https://www.boost.org/`
- Python 3.8 and pip installed

## 2.2 Building and installing KLEE with A-star module

To install KLEE-Reach, simply follow the usual instructions for building KLEE. Instead of following these steps on the official KLEE repository, use our repository:

`https://github.com/gardouin/klee-reach`

In addition to that, execute `make` in `klee-reach-utils`. This will install the Python module for calculating distances.

# 3 Using A-star exploration strategy in KLEE

In this part, we will focus on how to run KLEE with the new search algorithms. We assume the reader is familiar with the use of KLEE and has installed the A-star module (otherwise, please refer to the section 2).

If the reader wishes to understand how to use the basic functions of KLEE, they can refer to KLEE documentation: `http://klee-se.org/docs/`.

## 3.1 Overview

Running KLEE with the A-star exploration strategy can be done in two ways:

- by using the supplied `klee-reach.sh` script (available in `klee-reach-utils`)
- by manually executing all script steps

### 3.1.1 Using the script

The provided script performs all the steps required to use the new exploration strategies. Please note that the safest way to use an A-star exploration strategy is to use this script.

The only requirement, after having correctly build the new version of KLEE according to section 2, is to specify the path of the KLEE executable as shown in Listing 1.

```
1  #!/usr/bin/env bash
2
3  ##########################################
4  #             USER CONFIGURATION         #
5  ##########################################
6
7  # Path to KLEE executable (built with ASTAR module)
8  klee=""
9  # Specify absolute path to klee-reach-utils
10 kreachdist="."
```

Listing 1: First lines of `klee-reach.sh`

Once this condition has been met, the user simply needs to execute the script as if using KLEE. For instance, if the user wishes to execute the following command:

```
klee --max-instructions=1000 --posix-runtime foo.bc --sym-arg 10,
```

he just needs to run instead

```
./klee-reach.sh --max-instructions=1000 --posix-runtime foo.bc --sym-arg 10.
```

It is possible to select the search heuristic by specifying script options:

- `-a`: use `AStar`
- `-A`: use `AStar2`
- `-k`: use KLEE's default search (you can then override the choice by using `--search=<searcher>`)

The default search heuristic is `AStar2`.

For more details on the different steps carried out by the script, see section 3.1.2.

### 3.1.2 Manual execution

Running KLEE with A-star search heuristic can be done by carrying out the following steps:

1. Use KLEE to generate the LLVM file by indicated *all* command line arguments that will be used in KLEE final execution:

   ```
   klee --max-instructions=1 --output-dir=__dist__ [args] foo.bc
   ```

2. In `klee-reach-utils`, execute the Python distance script to generate the `.dist` file:

   ```
   python3 kreachdist/compute_distance.py __dist__/assembly.ll
   ```

   Please note that *exactly* one target has to be defined in the source code by using the instruction `klee_reach()` (more details in section 3.4). Otherwise, there is no distance to calculate. In the last case, A-star approach loses all interest.

   For more details on the computation of the distances, see section 3.3.

3. Run KLEE with a AStar heuritic and previous command line options:

   ```
   klee --search=astar --input-distance-file="foo.dist" [args] foo.bc
   ```

   ```
   klee --search=astar2 --input-distance-file="foo.dist" [args] foo.bc
   ```

   This command will launch KLEE with the selected algorithm. The user has to specify the path of the precomputed file. If the parsing of the distance file failed, the user will be notified by KLEE.

The A-star approach has been proposed for reachability analysis. Consequently, halting KLEE when the target has been found should interest the reader. It can be done by asking KLEE to stop whether this situation occurs:

```
klee --exit-on-error-type=Reach [astar options] foo.bc
```

If such path exists, KLEE generates a `test[0-9]+.reached` file in the `klee-last` folder. Information relating to this path can be found by identifying the test number with the `.reached` file.

Following sections will explain more thoroughly the different steps to use A-star as an exploration strategy in KLEE.

## 3.2 Command line options

Various command line options have been added to this KLEE extension. This section explains each of these options.

Please note: all these options are listed in the "AStar options" category (see `klee --help`). They therefore have no impact on KLEE's operation when using native searchers.

### 3.2.1 Debug option

Using the command line option `--debug-print-worklist` will output on `stderr` various information regarding worklist states, such as the priority of a state or the depth in the branch. Other information, such as the value of $g$ or $\mu$ for `AStar2` searcher, is available.

Be advised: as useful as displaying the worklist may be, this process is cumbersome and slows down KLEE considerably. That is to say, if you are hoping to get a result within a reasonable amount of time, or get a result at all, don't use this option.

### 3.2.2 Input distance file option

The distances used in the AStar algorithm have to be computed before the execution of KLEE. The command line option `--input-distance-file="file.dist"` is used to provide KLEE with the file containing the distances (more details on 3.3).

If KLEE is unable to read the distance file, for instance because the file format is incorrect or the path is wrong, KLEE will inform the user that it has been unable to parse the file correctly.

Forgetting to input the distance file or giving the wrong path will *not* prevent KLEE from using the AStar searcher. However, all distances from the target will be considered infinite.

## 3.3 Computing distances

As described in the presentation of the A-star exploration strategy [2], the priority function that defines the exploration order takes into account the distance between the target in the code and an LLVM instruction. These distances must be pre-calculated with a Python script following the instructions in section 3.1.1.

To use KLEE's A-star strategy on any `foo.bc`, one may use the `klee-reach.sh` script. Although it is easy to obtain an LLVM file from a bytecode file, we can't be sure that this file will be exactly the same as the one KLEE will use beforehand. This is why, before running the Python script on an LLVM file, the script uses KLEE on the bytecode file to generate the "correct" LLVM file. Running KLEE with the exact same options as for symbolic execution generates the correct LLVM file:

```
klee [options] --max-instructions=1 foo.bc
```

Using `--max-instructions=1` ensures that the correct LLVM file is generated and that KLEE is terminated quickly. This LLVM file is generated in the `klee-last` folder under the name `assembly.ll`. Running the Python script on it generates the `.dist` file.

## 3.4 Reachability

KLEE is intended for use in generating high-coverage tests for programs. However, using KLEE's symbolic execution engine for reachability analysis is also an appropriate use of this software. In view of the new exploration strategies, certain additions to KLEE had to be made and are explained here.

### 3.4.1 `klee_reach()` instruction

It is possible to use KLEE for reachability analysis. However, the A-star heuristic has to compute the distances between an LLVM instruction and a target. A clear representation of the target in LLVM is therefore required, and none of KLEE's tools seem relevant in this respect. Consequently, a special instruction has been implemented to designate the target in code: `klee_reach()`[1].

The listing 2 illustrates the original code that a user would like to execute symbolically to see if the target is reachable.

```
1  int main()
2  {
3      int x;
4      klee_make_symbolic(&x, sizeof(x), "x");
5      if (x == 0) {
6          // target here
7          return 0;
8      }
9     return 1;
10 }
```

Listing 2: Example without using `klee_reach()`

To make the target explicit in the listing 2, the new instruction must be used following the example in listing 3.

```
1  int main()
2  {
3      int x;
4      klee_make_symbolic(&x, sizeof(x), "x");
5      if (x == 0) {
6          klee_reach();
7          return 0;
8      }
9     return 1;
10 }
```

Listing 3: Example using `klee_reach()`

When the Python script computes the distances, it searches for the target defined by the `klee_reach()` instruction. Without this instruction, the Python script will not be able to compute any distance.

[1] This solution is inspired by the `klee_assert()` feature, however implementing a new instruction has enabled us to define specific behaviour for KLEE.

### 3.4.2 Asking KLEE to stop whether the target is hit

Another feature that will be useful for someone trying to do reachability analysis with KLEE will be to stop the program when a path to the target has been found.

This functionality already exists in KLEE: it is possible to ask KLEE to stop whether a state in KLEE encounters an error case by specifying `--exit-on-error` on the command line. It also possible to specify the type of error by using `--exit-on-error-type=<error type>` instead.

For reachability, it possible to ask KLEE to stop when the target has been hit using: `--exit-on-error-type=Reach`. When it reaches the target, by executing the instruction `klee_reach()`, KLEE triggers a termination signal to the current state. This signal is the "error of type Reach", thus KLEE will stop immediately.

Please note that the term "error" is the generic term used by KLEE to designate a case of program stoppage. Indeed, reaching the target is not an error, but it was more convenient for the developer to adapt an existing feature of KLEE.

# References


[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: Unassisted and automatic generation of High-Coverage tests for complex systems programs. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08)*, San Diego, CA, December 2008. USENIX Association.

[2] Theo De Castro Pinto, Antoine Rollet, Grégoire Sutre, and Ireneusz Tobor. Guiding symbolic execution with a-star. In Carla Ferreira and Tim A. C. Willemse, editors, *Software Engineering and Formal Methods*, pages 47–65, Cham, 2023. Springer Nature Switzerland.