* Install K: <https://github.com/kframework/k/releases>

**Introduction**

The K Framework is a tool for designing and modeling programming languages and software/hardware systems. At the core of the K Framework is a programming, modeling, and specification language called K. The K Framework includes tools for compiling K specifications to build interpreters, model checkers, verifiers, associated documentation, and more.

**Quick Start**

If you are not a K developer, but just want to get started using K, we provide a streamlined installation process for any system that supports [Nix](https://nixos.org/download.html):

bash <(curl https://kframework.org/install)

kup install k

For more information on the kup tool and other packaged releases of K, please refer to our [installation notes](https://github.com/runtimeverification/k/blob/master/k-distribution/INSTALL.md).

**Preface**

This is a readme file for *K developers*. Users should feel comfortable using the command line, as we do not provide GUI tools at this time.

*K-based tool users* should:

1. Consult their tool documentation for build/installation instructions.
2. If needed, download a [packaged release](https://github.com/runtimeverification/k/releases/) of the K Framework as part of their tool setup process.

If you are interested in quickly trying out the K Framework without building from source, please see our [packaged release installation guide](https://github.com/runtimeverification/k/blob/master/k-distribution/INSTALL.md).

The rest of this file assumes you intend to build and install the K Framework from source.

Note that the K Framework can only be built on (x86-64) Linux-like systems, e.g., this also includes macOS/brew (x86-64) as well as the Windows Subsystem for Linux. All 32-bit systems are **not supported**. See the [installation notes](https://github.com/runtimeverification/k/blob/master/k-distribution/INSTALL.md) for details about supported configurations and system setup.

**Contents**

1. [Prerequisite Install Guide](https://github.com/runtimeverification/k#prerequisite-install-guide)
2. [Build and Install Guide](https://github.com/runtimeverification/k#build-and-install-guide)
3. [IDE Setup](https://github.com/runtimeverification/k#ide-setup)
4. [Running the Test Suite](https://github.com/runtimeverification/k#running-the-test-suite)
5. [Changing the KORE Data Structures](https://github.com/runtimeverification/k#changing-the-kore-data-structures)
6. [Building the Final Release Directory/Archives](https://github.com/runtimeverification/k#building-the-final-release-directoryarchives)
7. [Compiling Definitions and Running Programs](https://github.com/runtimeverification/k#compiling-definitions-and-running-programs)
8. [Installing Python Support](https://github.com/runtimeverification/k#installing-python-support)
9. [Troubleshooting](https://github.com/runtimeverification/k#troubleshooting)

**Prerequisite Install Guide**

Before building and installing the K Framework, the following prerequisites must first be installed.

**The Short Version**

On Ubuntu Linux 22.04 (Jammy):

git submodule update --init --recursive

sudo apt-get install build-essential m4 openjdk-11-jdk libfmt-dev libgmp-dev libmpfr-dev pkg-config flex bison z3 libz3-dev maven python3 python3-dev cmake gcc g++ clang-14 lld-14 llvm-14-tools zlib1g-dev libboost-test-dev libyaml-dev libjemalloc-dev

curl -sSL https://get.haskellstack.org/ | sh

On Ubuntu Linux 20.04 (Focal):

**Note**: the installation process is very similar to the above, the only difference is that clang, lld and llvm-tools have to be version 12.

git submodule update --init --recursive

sudo apt-get install build-essential m4 openjdk-11-jdk libfmt-dev libgmp-dev libmpfr-dev pkg-config flex bison z3 libz3-dev maven python3 python3-dev cmake gcc g++ clang-12 lld-12 llvm-12-tools zlib1g-dev libboost-test-dev libyaml-dev libjemalloc-dev

curl -sSL https://get.haskellstack.org/ | sh

On Arch Linux:

git submodule update --init --recursive

sudo pacman -S git maven jdk-openjdk cmake boost fmt libyaml jemalloc clang llvm lld zlib gmp mpfr z3 curl stack base-devel base python

If you install this list of dependencies, continue directly to the [Build and Install Guide](https://github.com/runtimeverification/k#build-and-install-guide).

On macOS using [Homebrew](https://brew.sh/):

git submodule update --init --recursive

brew install bison boost cmake flex fmt gcc gmp openjdk jemalloc libyaml llvm@15 make maven mpfr pkg-config python stack zlib z3

**Note**: in this case llvm@15 is required.

**The Long Version**

The following dependencies are needed either at build time or runtime:

* [bison](https://www.gnu.org/software/bison/)
* [boost](https://www.boost.org/)
* [cmake](https://cmake.org/)
* [flex](https://github.com/westes/flex)
* [fmt](https://fmt.dev/)
* [gcc](https://gcc.gnu.org/)
* [gmp](https://gmplib.org/)
* [jdk](https://openjdk.java.net/) (version 11 or greater)
* [libjemalloc](https://github.com/jemalloc/jemalloc)
* [libyaml](https://pyyaml.org/wiki/LibYAML)
* [llvm](https://llvm.org/) (We require version 10 or greater for clang, lld, and llvm-tools. On some distributions, the utilities below are also needed and packaged separately.)
  + [clang](http://clang.llvm.org/)
  + [lld](https://lld.llvm.org/)
* [make](https://www.gnu.org/software/make/)
* [maven](https://maven.apache.org/)
* [mpfr](http://www.mpfr.org/)
* [pkg-config](https://www.freedesktop.org/wiki/Software/pkg-config/)
* [python](https://www.python.org/)
* [stack](https://docs.haskellstack.org/en/stable/README/)
* [zlib](https://www.zlib.net/)
* [z3](https://github.com/Z3Prover/z3) (on some distributions libz3 is also needed and packaged separately) Note that you need version 4.8.15 of Z3, which may require you to build and install from source if your package manager does not supply it. Other versions are known to have bugs and performance regressions likely to cause issues in the K test suite.

Typically, these can all be installed from your package manager. On some system configurations, special installation steps or post-installation configuration steps are required. See the notes below.

**Installation Notes**

1. Java Development Kit (required JDK11 or higher)
   * Linux: Download from package manager (e.g. sudo apt-get install openjdk-11-jdk).
   * macOS/brew: Download from package manager (e.g. brew install java).

To make sure that everything works you should be able to call java -version and javac -version from a terminal.

1. LLVM
   * macOS/brew: Since LLVM is distributed as a keg-only package, we must explicitly make it available for command line usage. See the results of the brew info llvm command for more information on how to do this. Additionally, the default version of LLVM supplied by Homebrew is newer than the version supported by K. The formula llvm@15 should be used instead of llvm.
2. Flex / Bison
   * macOS/brew: The versions of these packages supplied by the OS are too old, and are not compatible with the K build. You must ensure that the Homebrew-installed versions are first on your PATH when building K (i.e. which flex is **not** /usr/bin/flex).
3. Apache Maven
   * Linux: Download from package manager (e.g. sudo apt-get install maven).
   * macOS/brew: Download it from a package manager or from <http://maven.apache.org/download.cgi> and follow the instructions on the webpage.

Maven usually requires setting an environment variable JAVA\_HOME pointing to the installation directory of the JDK (not to be mistaken with JRE).

You can test if it works by calling mvn -version in a terminal. This will provide the information about the JDK Maven is using, in case it is the wrong one.

1. Haskell Stack

To install, go to <https://docs.haskellstack.org/en/stable/README/> and follow the instructions. You may need to do stack upgrade to ensure the latest version of Haskell Stack.

**Build and Install Guide**

**Building with Maven**

Checkout the project source at your desired location and call mvn package from the main directory to build the distribution. For convenient usage, you can update your $PATH with <checkout-dir>/k-distribution/target/release/k/bin (strongly recommended, but optional).

You are also encouraged to set the environment variable MAVEN\_OPTS to -XX:+TieredCompilation, which will significantly speed up the incremental build process.

# KEVM: Semantics of EVM in K

In this repository, we provide a model of the EVM in K.

## Fast Installation

* bash <(curl https://kframework.org/install): install [kup package manager](https://github.com/runtimeverification/kup).
* kup install kevm: install KEVM.
* kup list kevm: list available KEVM versions.
* kup update kevm: update to latest KEVM version.

**NOTE**: The first run will take longer to fetch all the libraries and compile sources. (30m to 1h)

## Documentation/Support

These may be useful for learning KEVM and K (newest to oldest):

* [K, KEVM and Foundry Integration overview](https://www.youtube.com/watch?v=9PLnQStkiUo)
* [Jello Paper](https://jellopaper.org/), a nice presentation of this repository.
* [20 minute tour of the semantics](https://www.youtube.com/watch?v=tIq_xECoicQNov) at [2017 Devcon3](https://archive.devcon.org/archive/watch?edition=3&order=desc&sort=edition).
* [KEVM 1.0 technical report](http://hdl.handle.net/2142/97207), especially sections 3 and 5.
* [KEVM Paper at CSF'18/FLoC](https://fsl.cs.illinois.edu/publications/hildenbrandt-saxena-zhu-rodrigues-daian-guth-moore-zhang-park-rosu-2018-csf).

To get support for KEVM, please join our [Discord Channel](https://discord.com/invite/CurfmXNtbN).

If you want to start proving with KEVM, refer to [VERIFICATION.md](https://github.com/runtimeverification/evm-semantics/blob/master/VERIFICATION.md).

## Repository Structure

The following files constitute the KEVM semantics:

* [network.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/network.md) provides the status codes reported to an Ethereum client on execution exceptions.
* [json-rpc.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/json-rpc.md) is an implementation of JSON RPC in K.
* [evm-types.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/evm-types.md) provides the (functional) data of EVM (256-bit words, wordstacks, etc...).
* [serialization.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/serialization.md) provides helpers for parsing and unparsing data (hex strings, recursive-length prefix, Merkle trees, etc.).
* [evm.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/evm.md) is the main KEVM semantics, containing EVM’s configuration and transition rules.

These additional files extend the semantics to make the repository more useful:

* [buf.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/buf.md) defines the #buf byte-buffer abstraction for use during symbolic execution.
* [abi.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/abi.md) defines the [Contract ABI Specification](https://docs.soliditylang.org/en/v0.8.1/abi-spec.html) for use in proofs and easy contract/function specification.
* [hashed-locations.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/hashed-locations.md) defines the #hashedLocation abstraction used to specify Solidity-generated storage layouts.
* [edsl.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/edsl.md) combines the previous three abstractions for ease-of-use.
* [foundry.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/foundry.md) adds Foundry capabilities to KEVM.

These files are used for testing the semantics itself:

* [state-utils.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/state-utils.md) provides functionality for EVM initialization, setup, and querying.
* [driver.md](https://github.com/runtimeverification/evm-semantics/blob/master/include/kframework/driver.md) is an execution harness for KEVM, providing a simple language for describing tests/programs.

## Building from source

There are two backends of K available: LLVM for concrete execution and Haskell for symbolic execution. This repository generates the build-products for each backend in .build/usr/lib/kevm.

### System Dependencies

First install the following tools:

* [git](https://git-scm.com/)
* GNU [Bison](https://www.gnu.org/software/bison/), [Flex](https://github.com/westes/flex), and [Autoconf](http://www.gnu.org/software/autoconf/).
* GNU [libmpfr](https://www.mpfr.org/) and [libtool](https://www.gnu.org/software/libtool/).
* [Z3](https://github.com/Z3Prover/z3) version 4.12.1

#### Installing Z3

KEVM requires Z3 version 4.12.1, which you may need to install from a source build if your package manager supplies a different version. To do so, follow the instructions [here](https://github.com/Z3Prover/z3#building-z3-using-make-and-gccclang) after checking out the correct tag in the Z3 repository:

git clone https://github.com/Z3Prover/z3.git

cd z3

git checkout z3-4.12.1

python scripts/mk\_make.py

cd build

make

sudo make install

On macOS, it is easiest to install Z3 from Homebrew. If you wish to install from the source, install it to an appropriate prefix (e.g. /usr/local on Intel machines).

#### Ubuntu

On Ubuntu >= 22.04 (for example):

sudo apt-get install --yes \

autoconf bison clang-12 cmake curl flex gcc jq libboost-test-dev \

libcrypto++-dev libffi-dev libgflags-dev libjemalloc-dev libmpfr-dev \

libprocps-dev libsecp256k1-dev libssl-dev libtool libyaml-dev lld-12 \

llvm-12-tools make maven netcat-openbsd openjdk-11-jdk pkg-config \

protobuf-compiler python3 python3-dev python3-pip rapidjson-dev time \

zlib1g-dev libfmt-dev

On Ubuntu < 18.04, you'll need to skip libsecp256k1-dev and instead build it from source (via our Makefile):

make libsecp256k1

#### Haskell Stack (all platforms)

* [Haskell Stack](https://docs.haskellstack.org/en/stable/install_and_upgrade/#installupgrade). Note that the version of the stack tool provided by your package manager might not be recent enough. Please follow installation instructions from the Haskell Stack website linked above.

To upgrade stack (if needed):

stack upgrade

export PATH=$HOME/.local/bin:$PATH

### Build Dependencies

#### K Framework

The Makefile and kevm will work with either a (i) globally installed K or a (ii) K submodule included in this repository. For contributing to kevm, it is highly recommended to go with (ii) because some of the build scripts might not work otherwise. Follow these instructions to get and build the K submodule:

git submodule update --init --recursive -- deps/k

make k-deps

If you don't need either the LLVM or Haskell backend, there are flags to skip them:

make k-deps SKIP\_LLVM=true SKIP\_HASKELL=true

On an Apple Silicon machine, an additional flag to make is required to correctly build the Haskell backend:

make k-deps APPLE\_SILICON=true

#### Blockchain Plugin

You also need to get the blockchain plugin submodule and install it.

git submodule update --init --recursive -- deps/plugin

make plugin-deps

### Building

You need to set up a virtual environment using Poetry with the prerequisites python 3.8.\*, pip >= 20.0.2, poetry >= 1.3.2:

make poetry

Finally, you can build the semantics.

make build

You can build specific targets using build-llvm, build-Haskell, or build-foundry. For more information, refer to the [Makefile](https://github.com/runtimeverification/evm-semantics/blob/master/Makefile).

## Running Tests

To execute tests from the [Ethereum Test Set](https://github.com/ethereum/tests), the submodule needs to be fetched first.

git submodule update --init --recursive -- tests/ethereum-tests

The tests are run using the supplied Makefile. Run make build-prove to generate tests from the markdown files.

The following subsume all other tests:

* make test: All of the quick tests.
* make test-all: All of the quick and slow tests.

These are the individual test-suites (all of these can be suffixed with -all to also run slow tests):

* make test-vm: VMTests from the [Ethereum Test Set](https://github.com/ethereum/tests).
* make test-bchain: Subset of BlockchainTests from the [Ethereum Test Set](https://github.com/ethereum/tests).
* make test-proof: Proofs from the [Verified Smart Contracts](https://github.com/runtimeverification/verified-smart-contracts).
* make test-interactive: Tests of the kevm command.

When running tests with the Makefile, you can specify the TEST\_CONCRETE\_BACKEND (for concrete tests), or TEST\_SYMBOLIC\_BACKEND (for proofs).

## For Developers

If built from the source, the kevm executable will be in the .build/usr/bin directory. To make sure you are using the correct kevm, add this directory to your PATH:

export PATH=$(pwd)/.build/usr/bin:$PATH

Alternatively, if you work on multiple checkouts of evm-semantics or other semantics, you could add the relative path .build/usr/bin to your PATH. Do note, however, that this is a security concern.

You can call kevm help to get a quick summary of how to use the script.

Run the file tests/ethereum-tests/LegacyTests/Constantinople/VMTests/vmArithmeticTest/add0.json:

kevm run tests/ethereum-tests/LegacyTests/Constantinople/VMTests/vmArithmeticTest/add0.json --schedule DEFAULT --mode VMTESTS

To enable the debug symbols for the llvm backend, build using this command:

make build-llvm KEVM\_OPTS=--enable-llvm-debug

To debug a conformance test, add the --debugger flag to the command:

kevm interpret tests/ethereum-tests/BlockchainTests/GeneralStateTests/stExample/add11.json --backend llvm --mode NORMAL --schedule MERGE --chainid 1 --debugger

### Keeping in mind while developing

Always have your build up-to-date.

* If using the kup package manager, run kup install kevm --version . to install the local version.
* If building from source:
  + make poetry needs to be re-run if you touch any of the kevm-pyk code.
  + make build needs to be re-run if you touch any of this repos files.
  + make deps needs to be re-run if there is an update of the K submodule (you did git submodule update --init --recursive -- deps/k and it actually did something).
  + If both deps and build need to be re-run, you need to do deps first.
  + make clean is a safe way to remove the .build directory, but then you need to re-run make deps (should be quick this time) and make build.