MicroTESK 2.5

Installation Guide

ISP RAS

Version 2.5.0, January 9, 2020

System Requirements

Being developed in Java, MicroTESK can be used on Windows, Linux, macOS, and other platforms with the following software installed:

- JDK 11+ (https://openjdk.java.net);
- Apache Ant 1.8+ (https://ant.apache.org).

To generate test data based on constraints (if required), MicroTESK needs an SMT solver such as Z3 or CVC4 (see Installing SMT Solvers).

Installation

Installation Steps

1. Download from http://forge.ispras.ru/projects/microtesk/files and unpack a distribution package (the latest `.tar.gz` file). The destination directory will be further referred to as `<INSTALL_DIR>`.
2. Set the `MICROTESK_HOME` environment variable to the `<INSTALL_DIR>` path (see Setting Environment Variables).
3. Add the `<INSTALL_DIR>/bin` path to the `PATH` environment variable.
4. If required, install SMT solver(s) to the `<INSTALL_DIR>/tools` directory (see Installing SMT Solvers).

Setting Environment Variables

Windows

1. Start the `Control Panel` dialog.
2. Click on the following items:
   - `System and Security`;
   - `System`;
   - `Advanced system settings`.
3. Click on `Environment Variables`.
4. Click on `New…` under `System Variables`.
5. Specify `Variable name` as `MICROTESK_HOME` and `Variable value` as `<INSTALL_DIR>`.
6. Click `OK` in all open windows.
7. Reopen the command prompt window.

Linux and macOS

Add the command below to `~/.bash_profile` (Linux) or `~/.profile` (macOS):

```
export MICROTESK_HOME=<INSTALL_DIR>
```

Changes will be applied after restarting the terminal.

Installation Directory Structure

<INSTALL_DIR> contains the following subdirectories:

Directory	Description
arch	Microprocessor specifications and test templates
bin	Scripts for model compilation and test generation
doc	Documentation
etc	Configuration files
gen	Generated code of microprocessor models
lib	JAR files and Ruby scripts
src	Source code
tools	SMT solvers

Installing SMT Solvers

To generate test data based on constraints, MicroTESK uses external SMT solvers. There are supported Z3 and CVC4. Solver executables should be placed to <INSTALL_DIR>/tools.

Specifying paths

If solvers are already installed, MicroTESK can find them by using the Z3_PATH and CVC4_PATH environment variables.

NOTE	Each _PATH variable specifies the path to the corresponding executable.

Installing Z3

- Build Z3 as it is described in https://github.com/Z3Prover/z3.
- Move the executable file to the following path (or create a symbolic link):
  - <INSTALL_DIR>/tools/z3/windows/z3.exe (Windows);
  - <INSTALL_DIR>/tools/z3/unix/z3 (Linux);
  - <INSTALL_DIR>/tools/z3/osx/z3 (macOS).

Installing CVC4

- Build CVC4 as it is described in https://github.com/CVC4/CVC4/.
- Move the executable file to the following path (or create a symbolic link):
  - <INSTALL_DIR>/tools/cvc4/windows/cvc4.exe (Windows);

- `<INSTALL_DIR>/tools/cvc4/unix/cvc4` (Linux);
- `<INSTALL_DIR>/tools/cvc4/osx/cvc4` (macOS).

Usage

Model Compilation

To compile a microprocessor model from its specification, run the following command:

- `sh compile.sh <SPECIFICATION>` (Linux and macOS);
- `compile.bat <SPECIFICATION>` (Windows).

For example, the actions below compile the model from the miniMIPS specification:

```
$ cd $MICROTESK_HOME
$ sh bin/compile.sh arch/minimips/model/minimips.nml
arch/minimips/model/mmu/minimips.mmu
```

NOTE	Models for all demo specifications are included into the MicroTESK distribution package. There is no need to compile them.

Test Program Generation

To generate a test program for a given architecture (model) and a given test template, run the following command:

- `sh generate.sh <ARCH> <TEMPLATE>` (Linux and macOS);
- `generate.bat <ARCH> <TEMPLATE>` (Windows).

For example, the actions below generate a test program for the miniMIPS model compiled in the previous example and the `euclid.rb` test template:

```
$ cd $MICROTESK_HOME
$ sh bin/generate.sh minimips arch/minimips/templates/euclid.rb
```

The output file name depends on the `--code-file-prefix` and `--code-file-extension` options (see Options).

To specify which SMT solver should be used to solve constraints (if required), specify the `--solver <SOLVER>` (or `-s <SOLVER>`) option, where `<SOLVER>` is `z3` or `cvc4` (by default, Z3 is used):

```
$ cd $MICROTESK_HOME
$ sh bin/generate.sh -s cvc4 minimips arch/minimips/templates/constraint.rb
```

More information on command-line options can be found in Command-Line Options.

Options

Command-Line Options

MicroTESK works in two main modes:

- model compilation (`--translate`);
- test program generation (`--generate`).

NOTE The `--translate` and `--generate` keys are inserted into the command line by `compile.sh` (or `compile.bat`) and `generate.sh` (or `generate.bat`) correspondingly.

Other options should be specified explicitly:

Name	Shortcut	Description	Requires
`--help`	`-h`	Shows help message	—
`--verbose`	`-v`	Enables printing diagnostic messages	—
`--translate`	`-t`	Translates formal specifications	—
`--generate`	`-g`	Generates test programs	—
`--output-dir <arg>`	`-od`	Sets where to place generated files	—
`--include <arg>`	`-i`	Sets include files directories	`--translate`
`--extension-dir <arg>`	`-ed`	Sets directory that stores user-defined Java code	`--translate`
`--random-seed <arg>`	`-rs`	Sets seed for randomizer	`--generate`
`--solver <arg>`	`-s`	Sets constraint solver engine to be used	`--generate`
`--branch-exec-limit <arg>`	`-bel`	Sets the limit on control transfers to detect endless loops	`--generate`
`--solver-debug`	`-sd`	Enables debug mode for SMT solvers	`--generate`
`--trace-log`	`-tl`	Saves simulator log in Tarmac format	`--generate`
`--self-checks`	`-sc`	Inserts self-checking code into test programs	`--generate`
`--default-test-data`	`-dtd`	Enables generation of default test data	`--generate`
`--arch-dirs <arg>`	`-ad`	Home directories for tested architectures	`--generate`
`--rate-limit <arg>`	`-rl`	Generation rate limit, causes error when broken	`--generate`
`--code-file -extension <arg>`	`-cfe`	The output file extension	`--generate`
`--code-file-prefix <arg>`	`-cfp`	The output file prefix (file names are as follows `prefix{\_}xxxx.ext`, where `xxxx` is a 4-digit decimal number)	`--generate`
`--data-file -extension <arg>`	`-dfe`	The data file extension	`--generate`

Name	Shortcut	Description	Requires
`--data-file-prefix <arg>`	`-dfp`	The data file prefix	`--generate`
`--exception-file-prefix <arg>`	`-efp`	The exception handler file prefix	`--generate`
`--program-length-limit <arg>`	`-pll`	The maximum number of instructions in output programs	`--generate`
`--trace-length-limit <arg>`	`-tll`	The maximum length of execution traces of output programs	`--generate`
`--comments-enabled`	`-ce`	Enables printing comments; if not specified no comments are printed	`--generate`
`--comments-debug`	`-cd`	Enables printing detailed comments; must be used together with `--comments-enabled`	`--generate`
`--no-simulation`	`-ns`	Disables simulation of generated test programs on the model	`--generate`
`--time-statistics`	`-ts`	Enables printing time statistics	`--generate`

Settings File

Default values of options are stored in the `<INSTALL_DIR>/etc/settings.xml` configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<settings>
  <setting name="random-seed" value="0"/>
  <setting name="branch-exec-limit" value="1000"/>
  <setting name="code-file-extension" value="asm"/>
  <setting name="code-file-prefix" value="test"/>
  <setting name="data-file-extension" value="dat"/>
  <setting name="data-file-prefix" value="test"/>
  <setting name="exception-file-prefix" value="test_except"/>
  <setting name="program-length-limit" value="1000"/>
  <setting name="trace-length-limit" value="1000"/>
  <setting name="comments-enabled" value=""/>
  <setting name="comments-debug" value=""/>
  <setting name="default-test-data" value=""/>
  <setting
    name="arch-dirs"
    value="cpu=arch/demo/cpu/settings.xml:minimips=arch/minimips/settings.xml"
  />
</settings>
```