temci Documentation

Release 0.8.2

Johannes Bechberger

Dec 01, 2020

CONTENTS

An advanced benchmarking tool written in Python 3 that supports setting up an environment for benchmarking and the generation of visually appealing reports.

It runs on Linux systems and (rudimentarily) on macOS.

CHAPTER

ONE

WHY SHOULD YOU USE TEMCI?

temci allows you to easily measure the execution time (and other things) of programs and compare them against each other resulting in a pretty HTML5 based report. Furthermore it can set up the environment to ensure benchmarking results with a low variance. The latter feature can be used without using temci for benchmarking by using temci short shell.

CHAPTER

TWO

USAGE

The main commands of temci are temci exec and temci report.

Suppose you want to see whether grepping for the strings that consist of `a` and `b` in the current folder is slower than for strings that consist only of `a`.

First we have to install temci (using Nix, see below for more instructions):

```
nix-env -f https://github.com/parttimenerd/temci/archive/master.tar.gz -i
```

After this, we can benchmark both commands with temci:

```
# benchmark both commands 20 times
temci short exec "grep '[ab]*' -R ." "grep 'a*' -R ." --runs 10

# append --watch to get report (in which you can move with the arrow keys and scroll)
# after every benchmark completed (use --watch_every to decrease interval)
temci short exec "grep '[ab]*' -R ." "grep 'a*' -R ." --runs 10 --watch

# if you want to improve the stability your benchmarks, run them with root privileges
# the benchmarked programs are run with your current privileges
temci short exec "grep '[ab]*' -R ." "grep 'a*' -R ." --runs 10 --sudo --preset usable
```

This results in a `run_output.yaml` file that should look like:

```
- attributes: {description: 'grep ''[ab]*'' -R .'}
  data:
    etime: [0.03, 0.02, 0.02, 0.03, 0.03, 0.03, 0.02, 0.03, 0.03, 0.02]
    ... # other properties
- attributes: {description: grep 'a*' -R .}
  data:
    etime: [0.02, 0.03, 0.02, 0.03, 0.03, 0.02, 0.03, 0.03, 0.02, 0.02]
    ... # other properties
- property_descriptions: {etime: elapsed real (wall clock) time, ... }
```

For more information on the support measurement tools (like perf stat and rusage), the supported plugins for setting up the environment and more, see temci exec.

We can now create a report from these benchmarking results using temci report. We use the option `--properties` to include only the elapsed time in the report to keep the report simple:

```
> temci report run_output.yaml --properties etime
Report for single runs
grep '[ab]*' -R .     (   10 single benchmarks)
     etime mean =     2(6).(000)m, deviation = 18.84223%
```

(continues on next page)

(continued from previous page)

```
grep 'a*' -R .        (   10 single benchmarks)
     etime mean =     2(5).(000)m, deviation = 20.00000%

Equal program blocks
     grep '[ab]*' -R .    grep 'a*' -R .
         etime confidence =        67%, speed up =      3.85%
```

We see that there is no significant difference between the two commands.

There are multiple reporters besides the default console reporter. Another reporter is the html2 reporter that produces an HTML report, use it by adding the `--reporter html2` option:

CHAPTER

THREE

INSTALLATION

The simplest way is to use the Nix package manager, after installing Nix, run:

```
nix-env -f https://github.com/parttimenerd/temci/archive/master.tar.gz -i
```

Using pip requiring at least Python 3.6:

```
pip3 install git+https://github.com/parttimenerd/temci.git
```

For more information see the Installation page.

3.1 Auto completion

Temci can generate auto completion files for bash and zsh. Add the following line to your *.bashrc* or *.zshrc*:

```
. `temci_completion $0`
```

CHAPTER

FOUR

USING TEMCI TO SET UP A BENCHMARKING ENVIRONMENT

Use the `temci short shell COMMAND` to run a command (`sh` by default) in a shell that is inside the benchmarking environment. Most options of `temci short exec` are supported. For more information, see temci shell.

CHAPTER

FIVE

WHY IS TEMCI CALLED TEMCI?

The problem in naming programs is that most good program names are already taken. A good program or project name has (in my opinion) the following properties:

- it shouldn't be used on the relevant platforms (in this case: github and pypi)
- it should be short (no one wants to type long program names)
- it should be pronounceable
- it should have at least something to do with the program

temci is such a name. It's lojban for time (i.e. the time duration between two moments or events).

CHAPTER

SIX

CONTRIBUTING

Bug reports and code contributions are highly appreciated.

For more information, see the Contributing page.

CHAPTER

SEVEN

CONTENTS OF THIS DOCUMENTATION

7.1 Installation

This page covers installating and updating temci.

7.1.1 System Requirements

- Linux or macOS (see Supported Operating Systems)
- Processor with an x86 or AMD64 architecture (although most features should work on ARM too)

7.1.2 Using Nix

The simplest way is to use the Nix package manager. After installing Nix, run:

```
nix-env -f https://github.com/parttimenerd/temci/archive/master.tar.gz -i
```

This method has the advantage that Nix downloads a suitable python3 interpreter and all packages like matplotlib that could otherwise cause problems. The Nix installation also runs all the test cases, to ensure that temci works properly on your system.

To install temci from source, run:

```
git clone https://github.com/parttimenerd/temci
cd temci
nix-env -i -f .
```

`nix-env -i -f .` can also be used to update your installation after updating the git repository. For a more convenient development environment, see also Temporary Python environment with nix-shell.

7.1.3 Using pip3

There is also the traditional way of using pip, requiring at least Python 3.6.

temci depends on the existence of some packages that cannot be installed properly using pip and have to be installed manually:

```
# on debian/ubuntu/...
time python3-pandas python3-cffi python3-cairo python3-cairocffi python3-matplotlib␣
↪python3-numpy python3-scipy linux-tools-`uname -r`
```

(continues on next page)

(continued from previous page)

```
# on fedora
time python3-pandas python3-cffi python3-cairo python3-cairocffi python3-matplotlib␣
↪python3-numpy python3-scipy perf
# on OS X (using homebrew)
gnu-time
```

The Linux packages can be installed by calling the `install_packages.sh` script.

After installing these packages, temci can be installed by calling:

```
pip3 install git+https://github.com/parttimenerd/temci.git
```

A package called temci exists on pypi, but temci depends on an unpublished version of the `click` library that is only available on github. This should change in the near future when the version 8.0 of `click` is published.

If there a problems with `click` (if you get an exception like `ImportError: cannot import name 'ParameterSource'`), try installing it directly from github:

```
pip3 install https://github.com/pallets/click/archive/
↪f537a208591088499b388b06b2aca4efd5445119.zip
```

To install temci from source, run:

```
git clone https://github.com/parttimenerd/temci
cd temci
pip3 install -e
```

Post Installation

Run the following command after the installation to compile some binaries needed for the `rusage` runner or the disabling of caches:

```
temci setup
```

This requires `gcc` and `make` to be installed.

7.1.4 Optional Requirements

Requirements that aren't normally needed are the following:

- `kernel-devel` packages (for compiling the kernel module to disable caches)
- `pdflatex` (for `pdf` report generation)

Temci runs perfectly fine without them if you are not using the mentioned features.

Auto Completion

Temci can generate auto completion files for bash and zsh. Add the following line to your *.bashrc* or *.zshrc*:

```
. `temci_completion $0`
```

7.2 temci build

Build programs before the actual benchmarks, can checkout specific git commits. This has the advantage of being able to configure the build for all benchmarked programs and to build these programs at once. This build config also contains the run config for each program. `temci build` compiles a run config and stores it into a file that can be directly used with temci exec (or other configured run drivers).

For most cases using the builder capabilities of `temci exec <temci_exec.html#building>`_ should be enough. This also has the advantage of using a single command for all benchmarked programs, whether they need to built or not.

7.2.1 Usage

```
Usage: temci build [OPTIONS] BUILD_FILE

Options:
  --tmp_dir TEXT                  Used temporary directory  [default:
                                  /tmp/temci]
  --threads INTEGER               Number of threads that build simultaneously
                                  [default: 1]
  --sudo                          Acquire sudo privileges and run benchmark
                                  programs with non-sudo user. Only supported
                                  on the command line.  [default: False]
  --sudo / --no-sudo              Acquire sudo privileges and run benchmark
                                  programs with non-sudo user. Only supported
                                  on the command line.  [default: False]
  --settings TEXT                 Additional settings file  [default: ]
  --out TEXT                      Resulting run config file  [default:
                                  run.exec.yaml]
  --log_level [debug|info|warn|error|quiet]
                                  Logging level  [default: info]
  --in TEXT                       Input file with the program blocks to build
                                  [default: build.yaml]
  --help                          Show this message and exit.
```

`in`, `out` and `threads` can also be set in the settings in the `build` block.

Be aware the parallel building or building multiple version of a program is still fragile.

Example

A build config (`build_config.yaml`) file for tool called `test` might look like this:

```
- attributes:
    description: 'test'
  run_config:
    run_cmd: 'sh test'
  build_config:
    build_cmd: 'echo "sleep 1" > test'
```

To build it, run `temci build build_config.yaml`, resulting in the following `run_config.yaml`:

```
- attributes:
    description: test
    tags: []
  run_config:
    cwd: [.]
    run_cmd: sh test
```

With temci exec

`temci exec` supports calling the builder directly, omitting the call to `temci build`. Just call `temci build` if you want to separate building and benchmarking.

7.2.2 File Format

`temci build` accepts a file that consists of a YAML list of the entries in the following format:

```
# Optional attributes that describe the block
attributes:
    description:         Optional(Str())

    # Tags of this block
    tags:         ListOrTuple(Str())

# Build configuration for this program block
build_config:
    # Base directory that contains everything to build an run the program
    base_dir:         Either(DirName()|non existent)
                default: .

    # Used version control system branch (default is the current branch)
    branch:         Either(Str()|non existent)

    # Command to build this program block, might randomize it
    cmd:          Str()

    # Number of times to build this program
    number:         Either(Int()|non existent)
                default: 1

    # Used version control system revision of the program (-1 is the current revision)
    revision:         Either(Either(Str()|Int())|non existent)
                default: -1
```

(continues on next page)

(continued from previous page)

```
    # Working directory in which the build command is run
    working_dir:          Either(DirName()|non existent)
                default: .

# Run configuration for this program block
run_config:          Dict(, keys=Any, values=Any, default = {})
```

7.2.3 VCS Support

Currently only Git is supported, but adding support for other version control systems is simple. The code for the VCS drivers is in the temci.utils.vcs module.

7.3 temci exec

This page explains `temci exec` and `temci short exec` that allow you to run the actual benchmarks.

The basic concept is that there are

run drivers that support a specific benchmarking concept (like benchmarking whole programs that can be executed in the shell), these run drivers use

runners for the actual benchmarking and

plugins to setup up the benchmarking environment

Currently only one run driver is implemented, the *exec* run driver that supports benchmarking programs executed in a shell.

The benchmarking process produces a YAML file with the benchmarking results.

There are multiple features that require root privileges. To use these features, call temci with the `--sudo` option. It will run only temci in super user mode, but not the benchmarked programs themself. Notable features that require these rights are cpu sets (for separating the benchmarked programs from the rest of the system), disabling hyperthreading and setting the CPU governor.

7.3.1 temci short exec

Supports basic benchmarks, without creating a configuration file. It supports the same command line options as `temci exec`:

```
Usage: temci short exec [OPTIONS] COMMANDS

-wd, --without_description COMMAND:
      Benchmark the command and use
      itself as its description. Appends
      '$ARGUMENT' to the command if the string
      isn't present. Use the '--argument' option
      to set the value that this string is
      replaced with.
-d, --with_description DESCRIPTION COMMAND:
      Benchmark the command
      and set its description attribute.Appends
```

(continues on next page)

(continued from previous page)

```
      '$ARGUMENT' to the command if the string
      isn't present. Use the '--argument' option
      to set the value that this string is
      replaced with.

...
(options of temci exec)
```

7.3.2 Usage

Basic benchmarking of two programs using the *time*:

```
# compare the run times of two programs, running them each 20 times
> temci short exec "sleep 0.1" "sleep 0.2" --runs 20
Benchmark 20 times                 [####################################]  100%
Report for single runs
sleep 0.1             (   20 single benchmarks)
     avg_mem_usage mean =            0.000, deviation =   0.0
     avg_res_set   mean =            0.000, deviation =   0.0
     etime         mean =      100.00000m, deviation = 0.00000%
     max_res_set   mean =         2.1800k, deviation = 3.86455%
     stime         mean =            0.000, deviation =   0.0
     utime         mean =            0.000, deviation =   0.0

sleep 0.2             (   20 single benchmarks)
     avg_mem_usage mean =            0.000, deviation =   0.0
     avg_res_set   mean =            0.000, deviation =   0.0
     etime         mean =      200.00000m, deviation = 0.00000%
     max_res_set   mean =         2.1968k, deviation = 3.82530%
     stime         mean =            0.000, deviation =   0.0
     utime         mean =            0.000, deviation =   0.0
```

The produced `run_output.yaml` file is:

```
- attributes: {__description: sleep 0.1, description: sleep 0.1}
  data:
    max_res_set: [2148.0, 2288.0, 2152.0, 2120.0, 2340.0, 2076.0, 2152.0, 2280.0,
      2080.0, 2276.0, 2124.0, 2120.0, 2136.0, 2156.0, 2272.0, 2280.0, 2284.0, 2060.0,
      2120.0, 2136.0]
    ...
- attributes: {__description: sleep 0.2, description: sleep 0.2}
  data:
    max_res_set: [2080.0, 2284.0, 2140.0, 2124.0, 2156.0, 2096.0, 2096.0, 2284.0,
      2288.0, 2120.0, 2284.0, 2280.0, 2284.0, 2272.0, 2272.0, 2152.0, 2152.0, 2328.0,
      2152.0, 2092.0]
    ...
- property_descriptions: {avg_mem_usage: average total mem usage (in K), ...}
```

More information on the format of the result file can be found in the documentation for temci report.

This documentation focuses on `temci exec` and its input file and options.

Presets

temci has the `--preset` option (and the setting `run/exec_misc/preset`) that enables a specific combination of plugins:

none no plugins are enabled, the default for non super user benchmarking

all Use all available plugins and render the system partially unusable by stopping all unnecessary processes etc., enables: *cpu_governor*, *disable_swap*, *sync*, *stop_start*, *other_nice*, *nice*, *disable_aslr*, *disable_ht*, *disable_intel_turbo*, *cpuset*

usable Use all plugins that do not affect other processes (besides restricting them to a single CPU), covers essentially the benchmarking tips of the LLVM project and enables: *cpu_governor*, *disable_swap*, *sync*, *nice*, *disable_aslr*, *disable_ht*, *cpuset*, *disable_intel_turbo*. This preset is used by default in super user mode (with ``–sudo``option).

Important: These presets don't include the *sleep* plugin. Enable it via `--sleep` if needed.

An overview over all available plugins is given at *Overview*.

Runners

The runners are selected on the command line using the `--runner` option and the configuration file via `run/exec_misc/runner`. They obtain the actual measurements and are configured in the run configuration. Configuring them in `temci short exec` is currently not possible.

time Uses the GNU time utility to measure basic properties. This is the default runner. It is relatively imprecise but gives good ball park numbers for the performance.

rusage Uses the `getrusage` method and a small wrapper written in C (be sure to call `temci setup` if you install temci via pip, to build the wrapper).

perf_stat Uses `perf stat` for measurements, might require root privileges. Allows measuring a wide range of properties

output This runner obtains the measurements by parsing the output of the benchmarked program and interpreting it as a YAML mapping of properties to measurements (`property: NUMBER` lines). It can be used in combination with the *time* and *perf_stat* runners (using the `--parse_output` option or setting `parse_output` to true in the run block config).

Building

`temci exec` supports to build the programs that are then benchmarked. It supports the same format and the same options as temci build.

In the most basic case (and the case that is thoroughly tested), just supply a build command:

```
- attributes: ...
  run_config: ...
  build_config:
    cmd: make # a sample build command
```

Executing the file with `temci exec` runs all available build commands.

This can be configured using the following options (set in the `run` settings block):

no_build Do not build, default is false

only_build: Only build the build configs for all blocks, default is false

abort_after_build_error default true

If building a block fails and `abort_after_build_error` is not true (e.g. `--no-abort_after_build_error` is passed), then temci produces an `EXEC_INPUT_FILE.erroneous.yaml` that contains the configurations of all failing blocks. This file can be used to execute the missing blocks again after the error is fixed. Use the `--append` option to append the benchmarks to the preexisting benchmark result file.

Error Codes

0	no error
1	at least one benchmarked program failed
255	temci itself failed

7.3.3 File format

The input file for `temci exec` consists of a list of entries per run program block:

```
-
  # Optional build config to integrate the build step into the run step
  build_config:          Either(Dict(, keys=Any, values=Any, default = {})|non␣
↪existent)

  # Optional attributes that describe the block
  attributes:
      description:          Optional(Str())

      # Tags of this block
      tags:          ListOrTuple(Str())

  run_config:
      # Command to benchmark, adds to run_cmd
      cmd:          Str()

      # Configuration per plugin
      time:
          ...
      ...

      # Command to append before the commands to benchmark
      cmd_prefix:          List(Str())

      # Execution directories for each command
      cwd:          Either(List(Str())|Str())
                  default: .

      # Disable the address space layout randomization
      disable_aslr:          Bool()

      # Override all other max runspecifications if > -1
      max_runs:          Int()
                  default: -1

      # Override all other min runspecifications if > -1
      min_runs:          Int()
                  default: -1
```

(continues on next page)

(continued from previous page)

```
    # Parse the program output as a YAML dictionary of that gives for a specific␣
↪property a
    # measurement. Not all runners support it.
    parse_output:          Bool()
                default: False

    # Used revision (or revision number).-1 is the current revision, checks out the␣
↪revision
    revision:          Either(Int()|Str())
                default: -1

    # Commands to benchmark
    run_cmd:          Either(List(Str())|Str())

    # Used runner
    runner:          ExactEither()
                default: time

    # Override min run and max runspecifications if > -1
    runs:          Int()
                default: -1

    # Environment variables
    env:          Dict(, keys=Str(), values=Any, default = {})

    # Configuration for the output and return code validator
    validator:
        # Program error output without ignoring line breaks and spaces at the␣
↪beginning
        # and the end
        expected_err_output:          Optional(Str())

        # Strings that should be present in the program error output
        expected_err_output_contains:          Either(List(Str())|Str())

        # Program output without ignoring line breaks and spaces at the beginning
        # and the end
        expected_output:          Optional(Str())

        # Strings that should be present in the program output
        expected_output_contains:          Either(List(Str())|Str())

        # Allowed return code(s)
        expected_return_code:          Either(List(Int())|Int())

        # Strings that shouldn't be present in the program output
        unexpected_err_output_contains:          Either(List(Str())|Str())

        # Strings that shouldn't be present in the program output
        unexpected_output_contains:          Either(List(Str())|Str())
```

A basic config file looks like:

```
- run_config:
    run_cmd: sleep 0.1
- run_config:
    run_cmd: sleep 0.2
```

7.3.4 Common options

These options are passed in the `run` settings block (see Settings API or directly on the command line, flags are of the schema `--SETTING/--no-SETTING`):

```
# Append to the output file instead of overwriting by adding new run data blocks
append:          Bool()

# Disable the hyper threaded cores. Good for cpu bound programs.
disable_hyper_threading:          Bool()

# Discard all run data for the failing program on error
discard_all_data_for_block_on_error:          Bool()

# First n runs that are discarded
discarded_runs:          Int()
            default: 1

# Possible run drivers are 'exec' and 'shell'
driver:          ExactEither('exec'|'shell')
            default: exec

# Input file with the program blocks to benchmark
in:          Str()
            default: input.exec.yaml

# List of included run blocks (all: include all)
# or their tag attribute or their number in the
# file (starting with 0), can be regular expressions
included_blocks:          ListOrTuple(Str())
            default: [all]

# Maximum time one run block should take, -1 == no timeout,
# supports normal time span expressions
max_block_time:          ValidTimespan()
            default: '-1'

# Maximum number of benchmarking runs
max_runs:          Int()
            default: 100

# Maximum time the whole benchmarking should take
#    -1 == no timeout
# supports normal time spans
# expressions
max_time:          ValidTimespan()
            default: '-1'

# Minimum number of benchmarking runs
min_runs:          Int()
            default: 20

# Output file for the benchmarking results
out:          Str()
            default: run_output.yaml

# Record the caught errors in the run_output file
record_errors_in_file:          Bool()
```

(continues on next page)

(continued from previous page)

```
            default: true

# Number of benchmarking runs that are done together
run_block_size:         Int()
            default: 1

# if != -1 sets max and min runs to its value
runs:           Int()
            default: -1

# Order in which the plugins are used, plugins that do not appear in this list are␣
↪used before all others
plugin_order: ListOrTuple(Str())
            default: ["drop_fs_caches", "sync", "sleep", "preheat", "flush_cpu_caches
↪"]

# If not empty, recipient of a mail after the benchmarking finished.
send_mail:          Str()

# Print console report if log_level=info
show_report:          Bool()
            default: true

# Randomize the order in which the program blocks are benchmarked.
shuffle:          Bool()
            default: true

# Store the result file after each set of blocks is benchmarked
store_often:          Bool()

cpuset:
    # Use cpuset functionality?
    active:         Bool()

    # Number of cpu cores for the base (remaining part of the) system
    base_core_number:         Int(range=range(0, NUMBER OF CPUS))
                default: 1

    #   0: benchmark sequential
    # > 0: benchmark parallel with n instances
    #  -1: determine n automatically (based on the number of cpu cores)
    parallel:          Int()

    # Number of cpu cores per parallel running program.
    sub_core_number:          Int(range=range(0, NUMBER OF CPUS))
                default: 1

    # Place temci in the same cpu set as the rest of the system?
    temci_in_base_set:  Bool()
                default: True

 # Maximum runs per tag (block attribute 'tag'), min('max_runs', 'per_tag') is used
max_runs_per_tag:          Dict(, keys=Str(), values=Int(), default = {})

# Minimum runs per tag (block attribute 'tag'), max('min_runs', 'per_tag') is used
min_runs_per_tag:          Dict(, keys=Str(), values=Int(), default = {})
```

(continues on next page)

(continued from previous page)

```
# Runs per tag (block attribute 'tag'), max('runs', 'per_tag') is used
runs_per_tag:         Dict(, keys=Str(), values=Int(), default = {})

# Do not build, the building process should not set the working directory
no_build: Bool()
                default: False

# Only build the build configs for all blocks
only_build: Bool()
                default: False

# Abort after the build error
abort_after_build_error: Bool()
                default: True
```

There also some exec run driver specific options:

```
# Parse the program output as a YAML dictionary of that gives for a specific property␣
↪a
# measurement. Not all runners support it.
parse_output:         Bool()

# Enable other plugins by default
preset:         ExactEither('none'|'all'|'usable')
            default: none

# Pick a random command if more than one run command is passed.
random_cmd:          Bool()
            default: true

# If not '' overrides the runner setting for each program block
runner:         ExactEither(''|'perf_stat'|'rusage'|'spec'|'spec.py'|'time'|'output')
```

Number of runs

The number of runs per block is either fixed by the `runs` settings that apply or is between the applying `min_runs` and `max_runs` setting. In the latter case, the benchmarking of a program block is stopped early as soon as there is some significance in the benchmarking results compared to all other benchmarked programs.

7.3.5 Runners

The runners are selected on the command line using the `--runner` option and the configuration file via `run/exec_misc/runner`. They are configured in the run configuration file using the settings block named like the runner in each run block.

time runner

Uses the GNU `time` tool and is mostly equivalent to the rusage runner but more user friendly.

The runner is configured by modifying the `time` property of a run configuration. This configuration has the following structure:

```
# Measured properties that are included in the benchmarking results
properties:          ValidTimePropertyList()
             default: [utime, stime, etime, avg_mem_usage, max_res_set, avg_res_set]
```

The measurable properties are:

utime user CPU time used (in seconds)

stime system (kernel) CPU time used (in seconds)

avg_unshared_data average unshared data size in K

etime elapsed real (wall clock) time (in seconds)

major_page_faults major page faults (required physical I/O)

file_system_inputs blocks wrote in the file system

avg_mem_usage average total mem usage (in K)

max_res_set maximum resident set (not swapped out) size in K

avg_res_set average resident set (not swapped out) size in K

file_system_output blocks read from the file system

cpu_perc percent of CPU this job got (total cpu time / elapsed time)

minor_page_faults minor page faults (reclaims; no physical I/O involved)

times_swapped_out times swapped out

avg_shared_text average amount of shared text in K

page_size page size

invol_context_switches involuntary context switches

vol_context_switches voluntary context switches

signals_delivered signals delivered

avg_unshared_stack average unshared stack size in K

socket_msg_rec socket messages received

socket_msg_sent socket messages sent

This runner is implemented in the TimeExecRunner class.

Supports the `parse_output` option.

rusage runner

Uses the `getrusage` method and a small wrapper written in C (be sure to call `temci setup` if you install temci via pip, to build the wrapper).

The runner is configured by modifying the `rusage` property of a run configuration. This configuration has the following structure:

```
# Measured properties that are stored in the benchmarking result
properties:         ValidRusagePropertyList()
            default: [idrss, inblock, isrss, ixrss,
                      majflt, maxrss, minflt,
                      msgrcv, msgsnd, nivcsw, nsignals,
                      nswap, nvcsw, oublock, stime, utime]
```

The measurable properties are:

utime user CPU time used

stime system CPU time used

maxrss maximum resident set size

ixrss integral shared memory size

idrss integral unshared data size

isrss integral unshared stack size

nswap swaps

minflt page reclaims (soft page faults)

majflt page faults (hard page faults)

inblock block input operations

oublock block output operations

msgsnd IPC messages sent

msgrcv IPC messages received

nsignals signals received

nvcsw voluntary context switches

nivcsw involuntary context switches

This runner is implemented in the RusageExecRunner class.

perf_stat runner

This runner uses the `perf stat` tool to obtain measurements. It might have to be installed separately (see *Installation <installation.html>*). `perf stat` allows measuring a myriad of properties but might require root privileges.

The runner is configured by modifying the `perf_stat` property of a run configuration. This configuration has the following structure:

```
# Limit measurements to CPU set, if cpusets are enabled
limit_to_cpuset:          Bool()
            default: true
```

(continues on next page)

(continued from previous page)

```
# Measured properties. The number of properties that can be measured at once is␣
↪limited.
properties:          List(Str())
             default: [wall-clock, cycles, cpu-clock, task-clock,
                       instructions, branch-misses, cache-references]

# If runner=perf_stat make measurements of the program repeated n times. Therefore␣
↪scale the number of
# times a program is benchmarked.
repeat:          Int()
             default: 1
```

The measureable properties can be obtained by calling `perf list`. Common properties are given above, other notable properties are `cache-misses` and `branch-misses`. The `wall-clock` property is obtained by parsing the non-csv style output of `perf stat` which is fragile.

This runner is implemented in the PerfStatExecRunner class.

Supports the `parse_output` option.

output runner

This runner obtains the measurements by parsing the output of the benchmarked program and interpreting it as a YAML mapping of property to measurement (`property: NUMBER` lines).

It can be used in combination with the *time* and the *perf_stat* runner, (using the `--parse_output` option), allowing benchmarking a command and parsing its result for additional measurements.

An example output is:

```
time: 10
load_time: 5
```

It also supports lists of values if the lists of all properties have the same number of elements. This can be used return the result of multiple measurements in one call of the benchmarked program:

```
time:      [11.0, 10.01, 8.5]
load_time: [5.0,   6.7,  4.8]
```

This runner is implemented in the OutputExecRunner class.

spec runner

This runner might not really work and is not really used.

Runner for SPEC like single benchmarking suites. It works with resulting property files, in which the properties are colon separated from their values.

The runner is configured by modifying the `spec` property of a run configuration. This configuration has the following structure:

```
# Base property path that all other paths are relative to.
base_path:         Str()

# Code that is executed for each matched path.
# The code should evaluate to the actual measured value
```

(continues on next page)

(continued from previous page)

```
# for the path. It can use the function get(sub_path: str = '')
# and the modules pytimeparse, numpy, math, random, datetime and time.
code:          Str()
             default: get()

# SPEC result file
file:          Str()

# Regexp matching the base property path for each measured property
path_regexp:          Str()
             default: .*
```

An example configuration is given in the following:

```
- attributes:
    description: spec
  run_config:
    runner: spec
    spec:
      file: "spec_like_result.yaml"
      base_path: "abc.cde.efg"
      path_regexp: 'bench\d'
      code: 'get(".min") * 60 + get(".sec") + random.random()'
- attributes:
    description: "spec2"
  run_config:
    runner: spec
    spec:
      file: "spec_like_result.yaml"
      base_path: "abc.cde.efg"
      path_regexp: 'bench\d'
      code: 'get(".min") * 60 + get(".sec") + 0.5 * random.random()'
```

This runner is implemented in the SpecExecRunner class.

7.3.6 Plugins

Plugins setup the benchmarking environment (e.g. set the CPU governor, ...). All their actions are reversible and are reversed if temci aborts or finishes.

The plugins are enabled via the command line option `--NAME`, in the configuration file via `run/exec_plugins/NAME_active` or by adding the name to set of active plugins in `run/exec_plugins/exec_active`. A collection of them can be activated using *Presets*.

All plugins are located in the temci.run.run_driver_plugin module.

Overview

New plugins can be added easily (see Extending temci) but there are multiple plugins already available:

cpu_governor Set the cpu governor

cpuset Uses *CPUSets* to separate the CPUs used for benchmarking from the CPUs that the rest of the system runs on

disable_aslr Disable address space randomisation

disable_cpu_caches Disables the L1 and L2 caches

disable_ht Disables hyper-threading

disable_intel_turbo Disables the turbo mode on Intel CPUs

disable_swap Disables swapping data from the RAM into a backing hard drive

discarded_runs Discard the first runs (sets the `run/discarded_runs` setting)

drop_fs_caches Drops file system caches

env_randomize Adds random environment variables to mitigate some cache alignment effects

flush_cpu_caches Flush the CPU caches on x86 CPUs

nice Increases the CPU and IO scheduling priorities of the benchmarked program

other_nice Decreases the CPU scheduling priority of all other programs

preheat Preheats the system with a CPU bound task

sleep Keeps the system idle for some time before the actual benchmarking

stop_start Stops almost all other processes (as far as possible)

sync Synchronizes cached writes of the file system to a persistent storage

The order in which the plugins are used (and called) is defined by the `run/plugin_order`, see common-options.

cpu_governor

Sets the CPU governor of all CPU cores.

The governor can be configured by either using the `--cpu_governor_governor GOVERNOR` option or by setting `run/exec_plugins/cpu_governor_misc/governor`.

The default governor is `performance` which is recommended for benchmarks.

The available governors can be obtained by calling

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

Requires root privileges.

cpuset

Uses cpusets to separate the CPUs used for benchmarking from the CPUs that the rest of the system runs on. For more information see *CPUSets*.

Requires root privileges.

disable_aslr

Disables the address space randomisation which might lead to less variance in the benchmarks.

Requires root privileges.

disable_cpu_caches

Disables the L1 and L2 caches on x86 and x86-64 architectures. It uses a small custom kernel module (be sure to compile it via `temci setup --build_kernel_modules` after install the appropriate `kernel-devel` package, see Installation).

Attention: It will slow down your system by orders of magnitude, giving you essentially a Pentium I like processor. Only use it for demonstration purposes.

Requires root privileges.

disable_ht

Disables hyper-threading, enabling it is equivalent to using the `disable_hyper_threading` option (see Common options).

It disable a number of CPU cores so that only one core per physical CPU core is active, thereby effectively disabling hyper-threading.

Requires root privileges.

disable_intel_turbo

Disables the turbo mode on Intel CPUs. Might reduce the variance of benchmarks, as the CPUs cannot overclock partially.

Requires root privileges.

disable_swap

Disables swapping data from the RAM into a backing hard drive. Swapping during benchmarking sessions increases the variance as accessing data on a hard drive is significantly slower than accessing data in RAM.

Requires root privileges.

discarded_runs

Discard the first runs (sets the `run/discarded_runs` setting). As a result, the benchmark files should already be in the file system caches.

drop_fs_caches

Drops the page cache, directoy entries and inodes before every benchmarking run. This might improve the usability of the produced benchmarks for IO bound programs.

It can be either configured by using the `run/exec_plugins/drop_fs_caches_misc` block in the settings or by using the command line options of the same names prefixed by `--drop_fs_caches_`:

```
# Free dentries and inodes
free_dentries_inodes: true

# Free the page cache
free_pagecache: true
```

Requires root privileges.

env_randomize

Adds random environment variables before each benchmarking run. This causes the stack frames of the called program to be aligned differently. Can mitigate effects caused by a specific cache alignment.

It can be either configured by using the `run/exec_plugins/env_randomize_misc` block in the settings or by using the command line options of the same names prefixed by `--env_randomize_`:

```
# Maximum length of each random key
key_max: 4096

# Maximum number of added random environment variables
max: 4

# Minimum number of added random environment variables
min: 4

# Maximum length of each random value
var_max: 4096
```

flush_cpu_caches

Write back and flush Internal caches; initiate writing-back and flushing of external caches (see WBINVD).

It uses a small custom kernel module (be sure to compile it via `temci setup --build_kernel_modules` after install the appropriate `kernel-devel` package, see Installation).

nice

Sets the `nice` and `ionice` values (and therefore the CPU and IO scheduler priorities) of the benchmarked program to a specific value.

It can be either configured by using the `run/exec_plugins/nice_misc` block in the settings or by using the command line options of the same names prefixed by `--nice_`:

```
# Specify the name or number of the scheduling class to use
#   0 for none
#   1 for realtime
#   2 for best-effort
#   3 for idle
io_nice: 1

# Niceness values range from -20 (most favorable to the process)
# to 19 (least favorable to the process).
nice: -15
```

`nice` values lower than -15 seem to cripple Linux systems.

Requires root privileges.

other_nice

Sets the `nice` value of processes other than the benchmarked one. Prioritises the benchmarked program over all other processes.

It can be either configured by using the `run/exec_plugins/other_nice_misc` block in the settings or by using the command line options of the same names prefixed by `--other_nice_`:

```
# Processes with lower nice values are ignored.
min_nice: -10

# Niceness values for other processes.
nice: 19
```

Requires root privileges.

preheat

Preheats the system with a CPU bound task (calculating the inverse of a big random matrix with numpy on all CPU cores).

The length of the preheating can be configured by either using the `--preheat_time SECONDS` option or by setting `run/exec_plugins/preheat_misc/time`.

When the preheating takes place (before each run or at the beginning of the benchmarking) can be configured via `--preheat_when [before_each_run|at_setup]` or by setting `run/exec_plugins/preheat_misc/when` (accepts a list).

sleep

Keep the system idle for some time before the actual benchmarking.

See Gernot Heisers Systems Benchmarking Crimes:

> Make sure that the system is really quiescent when starting an experiment, leave enough time to ensure all previous data is flushed out.

stop_start

Stops almost all other processes (as far as possible).

This plugin tries to stop most other processes on the system that aren't really needed. By default most processes that are children (or children's children, ...) of a process whose name ends with "dm" are stopped. This is a simple heuristic to stop all processes that are not vital (i.e. created by some sort of display manager). SSH and X11 are stopped too.

Advantages of this plugin (which is used via the command line flag `--stop_start`):

- No one can start other programs on the system (via ssh or the user interface)
- → fewer processes can interfere with the benchmarking
- Noisy processes like Firefox don't interfere with the benchmarking as they are stopped, this reduces the variance of benchmarks significantly

Disadvantages:

- You can't interact with the system (therefore use the send_mail option to get mails after the benchmarking finished)
- Not all processes that could be safely stopped are stopped as this decision is hard to make
- You can't stop the benchmarking as all keyboard interaction is disabled (by stopping X11)
- You might have to wait several minutes to be able to use your system after the benchmarking ended

Stopping a process here means to send a process a SIGSTOP signal and resume it by sending a SIGCONT signal later.

It can be either configured by using the `run/exec_plugins/stop_start_misc` block in the settings or by using the command line options of the same names prefixed by `--stop_start_`:

```
# Each process which name (lower cased) starts with one of the prefixes is not␣
↪ignored.
# Overrides the decision based on the min_id.
comm_prefixes: [ssh, xorg, bluetoothd]

# Each process which name (lower cased) starts with one of the prefixes is ignored.
# It overrides the decisions based on comm_prefixes and min_id.
comm_prefixes_ignored: [dbus, kworker]

# Just output the to be stopped processes but don't actually stop them?
dry_run: false

# Processes with lower id are ignored.
min_id: 1500

# Processes with lower nice values are ignored.
min_nice: -10
```

(continues on next page)

(continued from previous page)

```
# Suffixes of processes names which are stopped.
subtree_suffixes: [dm, apache]
```

Requires root privileges.

sync

Synchronizes cached writes of the file system to a persistent storage by calling `sync`.

7.3.7 CPUSets

The idea is to separate the benchmarked program from all other programs running on the system.

The usage of cpusets can be configured by using the following settings that are part of `run/cpuset` and can also be set using the options with the same names prefixed with `--cpuset_`:

```
# Use cpuset functionality?
active:          Bool()

# Number of cpu cores for the base (remaining part of the) system
base_core_number:          Int(range=range(0, 8))
             default: 1

#  0: benchmark sequential
# > 0: benchmark parallel with n instances
#  -1: determine n automatically, based on the number of CPU cores
parallel:          Int()

# Number of cpu cores per parallel running program.
sub_core_number:          Int(range=range(0, 8))
             default: 1

# Place temci in the same cpu set as the rest of the system?
temci_in_base_set:  Bool()
             default: True
```

This functionality can also be enabling by using the `--cpuset` flag or by enabling the *cpuset* plugin.

7.4 temci shell

`temci short shell` opens a shell in a benchmarking environment. It allows to execute your own benchmarking suite in its own cpuset with disabled hyper threading, ... This command has the same options as temci exec (regarding presets and plugins).

For example running your own benchmarking suite `bench.sh` in a reasonably setup environment can be done via:

```
temci short shell ./bench.sh
```

The launched shell is interactive:

```
> temci short shell
>> echo 1
1
```

`temci shell` accepts an input file as its argument which has the following structure (see ShellRunDriver:

```
# Optional build config to integrate the build step into the run step
build_config:         Either(Dict(, keys=Any, values=Any, default = {})|non existent)

# Optional attributes that describe the block
attributes:
    description:         Optional(Str())

    # Tags of this block
    tags:          ListOrTuple(Str())

run_config:
    # Execution directory
    cwd:          Either(List(Str())|Str())
                 default: .

    # Command to run
    run_cmd:           Str()
                 default: sh

    # Environment variables
    env:          Dict(, keys=Str(), values=Any, default = {})
```

7.5 temci report

`temci report` supports the statistical evaluation of benchmarking runs. It processes the output file of `temci exec`. This page gives an overview over the different reporters and the expected format of the input file. The creation of a new reporter is explained in Extending temci.

There a currently four different reporters:

`console` Outputs a summary of the benchmarks on the console, the default reporter

`html2` Creates a HTML based report with many graphics

`csv` Outputs a configurable csv table

`codespeed` Outputs JSON as expected by the codespeed tool

7.5.1 Usage

Using the html2 reporter:

7.5.2 File format

The input file for `temci report` consists of list of entries per run program block:

```
-
  # Optional attributes that describe the block
  attributes:
      description:         Optional(Str())

      # Tags of this block
```

(continues on next page)

(continued from previous page)

```
      tags:          ListOrTuple(Str())

  data:
      property_1: List(Either(Int()|Float()))
      ...

  # the run program aborted with an error
  error:
      message: Str()
      return_code': Int()
      output: Str()
      error_output: Str()

  # there was an internal error
  internal_error:
      message: Str()

  # only the error or the internal_error block can be present
  # the recorded data is the data recorded till the error occurred

# optional property descriptions
- property_descriptions:
      property_1: long name of property_1
```

7.5.3 Common Options

These options are passed in the `reporter` settings block (see Settings API or directly on the command line (flags are of the schema `--SETTING/--no-SETTING`):

```
# Exclude all data sets that contain only NaNs.
exclude_invalid:          BoolOrNone()
            default: true

# Properties that aren't shown in the report.
excluded_properties:          ListOrTuple(Str())
            default: [__ov-time]

# Files that contain the benchmarking results
in:          Either(Str()|ListOrTuple(Str()))
            default: run_output.yaml

# List of included run blocks (all: include all), identified by their description
# or tag attribute, can be regular expressions
included_blocks:          ListOrTuple(Str())
            default: [all]

# Replace the property names in reports with longer more descriptive versions?
long_properties:          BoolOrNone()

# Possible reporter are 'console', 'html2', 'csv' and 'codespeed'
reporter:          ExactEither('console'|'html2'|'csv'|'codespeed')
            default: console

# Produce xkcd like plots (requires the humor sans font to be installed)
xkcd_like_plots:          BoolOrNone()
```

Furthermore the formatting of numbers can be partially configured using the settings file block described in temci format.

The statistical evaluation and the used properties can be configured via the `stats` settings block or with the unprefixed options of the same names:

```
# Properties to use for reporting and null hypothesis tests,
# can be regular expressions
properties:         ListOrTuple(Str())
            default: [all]

# Possible testers are 't', 'ks' and 'anderson'
tester:          ExactEither('t'|'ks'|'anderson')
            default: t

# Range of p values that allow no conclusion.
uncertainty_range:          Tuple(float, float)
            default: [0.05, 0.15]
```

7.5.4 Console

A simple reporter that just outputs a basic analysis of the benchmarks on the command line. It works for large result files and can compute pair-wise statistical tests.

This reporter is either configured via the `report/console_misc` settings block or via the command line options of the same name (prefixed with `console_`):

```
# Matches the baseline block
baseline: ''

# Position of the baseline comparison:
# 'each': after each block
# 'after': after each cluster
# 'both': after each and after cluster
# 'instead': instead of the non baselined
baseline_position: each

# 'auto': report clusters (runs with the same description)
#         and singles (clusters with a single entry, combined) separately
# 'single': report all clusters together as one
# 'cluster': report all clusters separately
# 'both': append the output of 'cluster' to the output of 'single'
mode: auto

# Output file name or `-` (stdout)
out: '-'

# Report on the failing blocks
report_errors: true

# Print statistical tests for every property for every two programs
with_tester_results: true
```

Output for a simple benchmark (with `--properties utime`):

```
Report for single runs
sleep 0.5            (    2 single benchmarks)
```

(continues on next page)

(continued from previous page)

```
     utime mean =         1.(211)m, deviation =    33.27828%

sleep 1              (     2 single benchmarks)
     utime mean =         1.(172)m, deviation =    29.91891%

Equal program blocks
     sleep 0.5    sleep 1
         utime confidence =         95%, speed up =       3.26%
```

Or using *sleep 0.5* as a baseline (`--console_baseline "sleep 0.5"`):

```
Report for single runs
sleep 0.5            (     5 single benchmarks)
     utime mean =       (1).(661)m, deviation =    18.91399%

sleep 1              (     5 single benchmarks)
     utime mean =       (1).(138)m, deviation =    37.83985%

sleep 1              (     5) with baseline sleep 1              (     5)
     utime mean =      (68).(554)%, confidence =     9%, dev =    37.83985%,    18.91399%
geometric mean of relative mean =          68.554%

Uncertain program blocks
     sleep 0.5     sleep 1
         utime confidence =     9%, speed up =      31.45%
```

The sample `run_output.yaml` was created via `temci short exec 'sleep 0.5' 'sleep 1' --runs 5 --runner rusage`:

```
- attributes:
    description: sleep 0.5
  data:
    utime: [0.00145, 0.001275, 0.001518, 0.002089, 0.001971]
    # ...
- attributes:
    description: sleep 1
  data:
    utime: [0.00174, 0.000736, 0.001581, 0.00085, 0.000785]
```

7.5.5 HTML2

Creates a report with many graphics (box-plots and bar-graphs) and tables that can be exported to TeX. The produced HTML page also contains many explanations. Viewing it requires an internet connection.

Output for the simple benchmark from above (with `--properties utime --properties maxrss`):

All images and tables are statically generated, this results in a large HTML file with many ressources. It is therefore not recommended to use this reporter with a large number of benchmarking results (benchmarked programs and properties). Rule of thumb: Only use it to analyse results comparing less than eight programs.

This reporter is either configured via the `report/html2_misc` settings block or via the command line options of the same name (prefixed with `html2_`)

```
# Alpha value for confidence intervals
alpha: 0.05
```

(continues on next page)

(continued from previous page)

```
# Height per run block for the big comparison box plots
boxplot_height: 2.0

# Width of all big plotted figures
fig_width_big: 25.0

# Width of all small plotted figures
fig_width_small: 15.0

# Format string used to format floats
float_format: '{:5.2e}'

# Override the contents of the output directory if it already exists?
force_override: false

# Generate pdf versions of the plotted figures?
gen_pdf: false

# Generate simple latex versions of the plotted figures?
gen_tex: true

# Generate excel files for all tables
gen_xls: false

# Name of the HTML file
html_filename: report.html

# Show the mean related values in the big comparison table
mean_in_comparison_tables: true

# Show the mininmum related values in the big comparison table
min_in_comparison_tables: false

# Output directory
out: report

# Format string used to format floats as percentages
percent_format: '{:5.2%}'

# Show zoomed out (x min = 0) figures in the extended summaries?
show_zoomed_out: false
```

7.5.6 CSV

A reporter that outputs the configurable csv table with rows for each run block. It can be used to access the benchmarking result for further processing in other tools without using temci as a library or creating a new reporter (see Extending temci).

This reporter is either configured via the `report/csv_misc` settings block or via the command line options of the same name (prefixed with `csv_`):

```
# List of valid column specs
# format is a comma separated list of 'PROPERTY[mod]' or 'ATTRIBUTE'
# mod is one of: mean, stddev, property, min, max and stddev per mean
```

(continues on next page)

(continued from previous page)

```
# optionally a formatting option can be given via PROPERTY[mod|OPT1OPT2...]
# where the OPTs are one of the following:
#        % (format as percentage)
#        p (wrap insignificant digits in parentheses (+- 2 std dev))
#        s (use scientific notation, configured in report/number) and
#        o (wrap digits in the order of magnitude of 2 std devs in parentheses).
# PROPERTY can be either the description or the short version of the property.
# Configure the number formatting further via the number settings in the settings file
columns: [description]

# Output file name or standard out (-)
out: '-'
```

Output for a simple benchmark (with `--csv_columns "utime[mean|p],utime[stddev],utime[max]"`, see *Console <temci_report.html#Console>*):

```
utime[mean|p],utime[stddev],utime[max]
0.00(2),0.000,0.002
0.00(1),0.000,0.002
```

7.5.7 Codespeed

Reporter that outputs JSON as expected by codespeed. Branch name and commit ID are taken from the current directory. Use it like this:

```
temci report --reporter codespeed ... \
   | curl --data-urlencode json@- http://localhost:8000/result/add/json/
```

This reporter is either configured via the `report/codespeed_misc` settings block or via the command line options of the same name (prefixed with `codespeed_`):

```
# Branch name reported to codespeed. Defaults to current branch or else 'master'.
branch: ''

# Commit ID reported to codespeed. Defaults to current commit.
commit_id: ''

# Environment name reported to codespeed. Defaults to current host name.
environment: ''

# Executable name reported to codespeed. Defaults to the project name.
executable: ''

# Project name reported to codespeed.
project: ''
```

Output for a simple benchmark (with `--properties utime`, see *Console <#Console>*):

```
[
   {
      "project":"",
      "executable":"",
      "environment":"i44pc17",
      "branch":"master",
```

(continues on next page)

(continued from previous page)


```
        "commitid":null,
        "benchmark":"sleep 0.5: utime",
        "result_value":0.0016606000000000004,
        "std_dev":0.0003140857207833556,
        "min":0.001275,
        "max":0.002089
    },
    {
        "project":"",
        "executable":"",
        "environment":"i44pc17",
        "branch":"master",
        "commitid":null,
        "benchmark":"sleep 1: utime",
        "result_value":0.0011384,
        "std_dev":0.00043076889395591227,
        "min":0.000736,
        "max":0.00174
    }
]
```


7.6 temci init

Commands to create documented sample config files. Accepts the option `--settings FILE` to configure a backing settings file.

temci init settings Creates a sample settings file with all the default (and currently applied) settings. Might be used to update a settings file for a new version of temci.

temci init build_config Creates a sample build configuration file, for more information on the format see temci build.

temci init run_config Creates a sample exec configuration, for more information on the format see temci exec

7.7 temci format

```
temci format [OPTIONS] NUMBER [ABS_DEVIATION]
```

A small formatting utility, to format numbers with their standard deviation and si prefixes.

7.7.1 Usage Example

```
> temci format 1.0 0.5
1.(000)

> temci format 1.56 0.005
1.56(0)

> temci format 1560 --scientific_notation
1.560k
```

(continues on next page)

(continued from previous page)

```
> temci format 1560 --no-scientific_notation_si_prefixes
1.560e3
```

This tool uses the number formatting module temci.utils.number. The therein defined method format_number can be used to format numbers and has the same options as the tool itself. Read Usage as a Library on how to use the module in a project other than temci.

7.7.2 Options

```
Usage: temci format [OPTIONS] NUMBER [ABS_DEVIATION]

Options:
  --settings TEXT                 Additional settings file  [default: ]
  --log_level [debug|info|warn|error|quiet]
                                  Logging level  [default: info]
  --sigmas INTEGER                Number of standard deviation used for the
                                  digit significance evaluation  [default: 2]
  --scientific_notation_si_prefixes
                                  Use si prefixes instead of 'e...'  [default:
                                  True]
  --scientific_notation_si_prefixes / --no-scientific_notation_si_prefixes
                                  Use si prefixes instead of 'e...'  [default:
                                  True]
  --scientific_notation           Use the exponential notation, i.e. '10e3'
                                  for 1000  [default: True]
  --scientific_notation / --no-scientific_notation
                                  Use the exponential notation, i.e. '10e3'
                                  for 1000  [default: True]
  --percentages                   Show as percentages  [default: False]
  --percentages / --no-percentages
                                  Show as percentages  [default: False]
  --parentheses_mode [d|o]        Mode for showing the parentheses: either d
                                  (Digits are considered significant if they
                                  don't change if the number itself changes +=
                                  $sigmas * std dev) or o (digits are
                                  consideredsignificant if they are bigger
                                  than $sigmas * std dev)  [default: o]
  --parentheses                   Show parentheses around non significant
                                  digits? (If a std dev is given)  [default:
                                  True]
  --parentheses / --no-parentheses
                                  Show parentheses around non significant
                                  digits? (If a std dev is given)  [default:
                                  True]
  --omit_insignificant_decimal_places
                                  Omit insignificant decimal places  [default:
                                  False]
  --omit_insignificant_decimal_places / --no-omit_insignificant_decimal_places
                                  Omit insignificant decimal places  [default:
                                  False]
  --min_decimal_places INTEGER    The minimum number of shown decimal places
                                  if decimal places are shown  [default: 3]
  --max_decimal_places INTEGER    The maximum number of decimal places
                                  [default: 5]
  --force_min_decimal_places      Don't omit the minimum number of decimal
```

(continues on next page)

(continued from previous page)

```
                                places if insignificant?  [default: True]
--force_min_decimal_places / --no-force_min_decimal_places
                                Don't omit the minimum number of decimal
                                places if insignificant?  [default: True]
--help                          Show this message and exit.
```

These options can also be set in the settings file, under `report/number`.

7.8 OS Support

Linux is the main target for this tool. The support for other Unix like operating systems is limited. Most of the advanced environment setup functionality, like cpu sets or disabling hyper threading, is Linux specific.

7.8.1 What works and what does not

- **`temci exec` and `temci short`**
  - the `perf_stat` runner is Linux specific
  - all other runners should work, but it is uncertain whether the `rusage` runner works
  - the `time` runner requires the `gtime` program to be installed
  - most the environment setup code (i.e. the plugins) don't work, with the exception of `preheat` and `sleep` that are implemented in python
  - `--sudo` is only supported on Linux
- **`temci shell`**
  - see `temci exec` for the supported plugins
- **`temci setup`**
  - might not work
- **`temci report`, `temci build`, `temci clean`, `temci completion`, ...**
  - without any constraints

7.8.2 Other Unixes

Other Unix like operating systems aren't currently tested. But there is a chance that they might work as well.

7.8.3 Windows

Windows is currently not supported, but `temci report` might still work. The Linux subsystem in Windows might enable the usage of the features that work on Apples OS X.

7.9 Extending temci

Temci can be extended by either editing the code of temci directly or by placing the code in a file in your local `~/.temci` folder or in a folder that is passed to temci via the `TEMCI_PLUGIN_PATH` variable.

This page documents how to implement new reporters, runners and run plugins and how to use temci directly as a library.

7.9.1 Usage as a Library

temci can be used in library mode by importing via

```
import temci.utils.library_init
```

7.9.2 New Reporter

New reporters can be added be creating a subclass of AbstractReporter. Adding a new reporter can be useful to integrate temci into other tools. It has the advantage over using temci as a library that it is directly integrated into the cli and the settings framework.

The following is an implementation of a sample reporter that outputs some benchmarking information as JSON. This reporter is based on the `codespeed` reporter:

```
@register(ReporterRegistry, "json", Dict({
    # define the settings for this reporter
    # currently every setting has to have a valid default value
    "project": Str() // Default("") // Description("Project name reported to␣
↪codespeed."),
})) # the register call registers the reporter
class JSONReporter(AbstractReporter):
    """
    Outputs the benchmarking information with some meta data on the command line.
    """

    def report(self):
        """
        Create a report and output it as configured.
        """
        import json
        self.meta = {
            "project": self.misc["project"]  # access the settings specific to this␣
↪reporter
        }
        data = [self._report_prop(run, prop)
                # iterate overall recorded properties of all run programs
                for run in self.stats_helper.runs
                for prop in sorted(run.get_single_properties()]
        json.dump(data, sys.stdout)

    def _report_prop(self, run: RunData, prop: SingleProperty) -> dict:
        return {
            **self.meta,
            "benchmark": "{}: {}".format(run.description(), prop.property),
            "result_value": prop.mean(),
```

(continues on next page)

(continued from previous page)

```
            "std_dev": prop.stddev(),
            "min": prop.min(),
            "max": prop.max(),
        }
```

For more information, consider looking into the documentation of the report module.

7.9.3 New Runner

Before implementing a new runner, you should consider whether using the `output` runner is enough. The output runner parses the output of the benchmarked programs as a list of `property: value` mappings, e.g. the output of a program could be *time: 10000.0*.

Implementing a new runner offers more flexibility, but is also slightly more work. A runner can be implemented by extending the ExecRunner class.

A good example is the OutputRunner itself, with some added documentation:

```
@ExecRunDriver.register_runner()  # register the runner
class OutputExecRunner(ExecRunner):
    """
    Parses the output of the called command as YAML dictionary (or list of␣
↪dictionaries)
    populate the benchmark results (string key and int or float value).
    For the simplest case, a program just outputs something like `time: 1000.0`.
    """

    name = "output"   # name of the runner
    misc_options = Dict({})
    # settings of the runner, these can be set under `run/exec/NAME_misc` in the␣
↪settings file

    def __init__(self, block: RunProgramBlock):
        """
        Creates an instance.

        :param block: run program block to measure
        """
        super().__init__(block)

    def setup_block(self, block: RunProgramBlock, cpuset: CPUSet = None, set_id: int␣
↪= 0):
        """
        Configure the passed copy of a run program block (e.g. the run command).

        The parts of the command between two `$SUDO$` occurrences is run with
        super user privileges if in `--sudo` mode.

        :param block:  modified copy of a block
        :param cpuset: used CPUSet instance
        :param set_id: id of the cpu set the benchmarking takes place in
        """
        pass

    def parse_result_impl(self, exec_res: ExecRunDriver.ExecResult,
                          res: BenchmarkingResultBlock = None) -> BenchmarkingResultBlock:
```

(continues on next page)

(continued from previous page)

```
        """
        Parse the output of a program and turn it into benchmarking results.
        :param exec_res: program output
        :param res:      benchmarking result to which the extracted results should be␣
↪added
                         or None if they should be added to an empty one
        :return: the modified benchmarking result block
        """
        res = res or BenchmarkingResultBlock()
        # schema for the output of a program
        dict_type = Dict(key_type=Str(),
                         value_type=Either(Int(), Float(), List(Either(Int(),␣
↪Float()))),
                         unknown_keys=True)
        output = yaml.safe_load(exec_res.stdout.strip())
        if isinstance(output, dict_type):
            res.add_run_data(dict(output))
        elif isinstance(output, List(dict_type)):
            for entry in list(output):
                res.add_run_data(entry)
        else:
            raise BenchmarkingError("Not a valid benchmarking program output: {}"
                                    .format(exec_res.stdout))
        return res

    def get_property_descriptions(self) -> t.Dict[str, str]:
        """
        Returns a dictionary that maps some properties to their short descriptions.
        """
        return {}
```

7.9.4 New exec Plugin

New plugins for setting up the benchmarking environment can be developed by extending the AbstractRunDriverPlugin class.

A simple example is the *DisableSwap* plugin:

```
# register the plugin and state the configuration
@register(ExecRunDriver, "disable_swap", Dict({}))
class DisableSwap(AbstractRunDriverPlugin):
    """
    Disables swapping on the system before the benchmarking and enables it after.
    """

    needs_root_privileges = True

    def setup(self):  # called before the whole benchmarking starts
        self._exec_command("swapoff -a")

    def teardown(self):  # called after the benchmarking (and on abort)
        self._exec_command("swapon -a")
```

7.10 Contributing

Pull requests and issues are always welcomed.

7.10.1 Issues

Issues can be submitted at GitHub and should specify the used settings (and if possible the local `temci.yaml` configuration file).

7.10.2 New Features

New features, runners, reporters, ... are welcome. To learn how to extend temci, see Extending temci. The code can be added to the appropriate places and should be tested with a few tests.

7.10.3 Coding Style

The code should use type annotations everywhere and use functionality of the typecheck module whenever there is uncertainty over the type of a variable (e.g. when reading from a YAML file). The currently used python version 3.6, all code should run in python 3.6 and above.

7.10.4 Documentation

Be sure to keep the documentation up to date and document your code. The code comments are written in reStructuredText.

7.10.5 Testing

The tests are located in the `tests` folder and roughly grouped by the temci subcommand they belong to. New features should by covered by tests.

There is also support for doctests that can be added into the documentation.

The tests are using the pytest framework and can be executed by simply calling

```
./test.sh
```

It recommended to install the package `pytest-clarity` to improve the error output.

7.11 Changelog

7.11.1 0.8.2

- improve HTML2 reporter - fix typos - change "error" into "severe warning" - support disabling warnings alltogether - clean up duplicates - further improve the summary section - support zoomed out graphs (make this the default) - use local copy of all JS and CSS (no works offline)
- record some information on the execution environment
- don't build kernel modules by default

- remove meta analysis code

7.11.2 0.8.1

- fixed minor issues
- add new runner capabilities like output parsing or rusage

7.11.3 0.8.0

- removed the randomization features from the builder
- removed the html reporter (use the html2 reporter instead)

7.12 License

Some devices are designed to deny users access to install or run

modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents.

States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and

modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of

works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this

License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work

in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based

on the Program.

To "propagate" a work means to do anything with it that, without

permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other

parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices"

to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work

for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official

standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other

than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all

the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users

can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that

same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of

copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not

convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under

the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological

measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the

work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you

receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey,

and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to

produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

> a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
>
> b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
>
> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
>
> d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent

works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms

of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

> a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
>
> b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software

in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded

from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any

tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods,

procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or

specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a

requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided,

in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this

License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option

remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you

add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
>
> e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
>
> f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further

restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you

must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the

form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly

provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your

license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is

reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the

licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or

run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically

receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an

organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the

rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this

License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims

owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this

definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free

patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express

agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license,

and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or

arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within

the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting

any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have

permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue

to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of

the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the

Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future

versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different

permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY

APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING

WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided

above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest

possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest

to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short

notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school,

if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program

into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

7.13 API Documentation

7.13.1 Subpackages

temci.build package

Submodules

temci.build.build_processor module

temci.build.builder module

Module contents

This module contains the build part of temci (usable from the command line with *temci build*).

It's separated into four parts with the following purposes:

- build_processor.py: facade for the the builders
- builder.py: Build programs

temci.misc package

Submodules

temci.misc.game module

Module contents

The stuff in this folder doesn't really belong to the temci tool but builds on top of it some cool applications, like a benchmarksgame inspired comparison of different implementations of several languages. The tools may depend on other code or packages than the temci tool itself.

temci.run package

Submodules

temci.run.cpuset module

temci.run.run_driver module

temci.run.run_driver_plugin module

temci.run.run_processor module

temci.run.run_worker_pool module

Module contents

This module contains code to make the actual benchmarks.

temci.scripts package

Submodules

temci.scripts.cli module

temci.scripts.temci_completion module

Just a more performant version of *temci completion* that rebuilds the completion files only if the temci version changed. The advantage over using *temci completion* directly is, that it's normally significantly faster.

Usage:

``` temci_completion [zsh|bash]

``` This returns the location of the completion file.

`temci.scripts.temci_completion.`**`cli`**`()`
Process the command line arguments and call `temci completion` if needed.

`temci.scripts.temci_completion.`**`completion_dir`**`()` → str
Get the name of the completion directory

`temci.scripts.temci_completion.`**`completion_file_name`**`(`*shell: str*`)` → str
Get the completion file name for the passed shell and the current temci version

`temci.scripts.temci_completion.`**`create_completion_dir`**`()` → str
Create the directory for the completion files if it doesn't already exist.

`temci.scripts.temci_completion.`**`print_help`**`()`

#### temci.scripts.version module

Contains the current version of temci.

`temci.scripts.version.`**`version = '0.8.2'`**
The current version of temci

#### Module contents

This directory contains the command line interface and tab completion code and also the several wrapper scripts and the projects C++ code in sub directories.

### temci.setup package

#### Submodules

#### temci.setup.setup module

This module helps to build the C and C++ code in the scripts directory.

**`exception`** `temci.setup.setup.`**`ExecError`**`(`*cmd: str*, *out: str*, *err: str*`)`
Bases: `Exception`

Error raised if a command failed.

**cmd**
Failed command

**err**
Error output of the command

**out**
Output of the command

temci.setup.setup.**exec**(*dir: str*, *cmd: str*)
Run the passed command in the passed directory

**Parameters**

- **dir** – passed directory
- **cmd** – passed command

**Raises** ***ExecError*** – if the executed program has a > 0 error code

temci.setup.setup.**make_scripts**(*build_kernel_modules: bool = False*)
Builds the C and C++ code inside the scripts directory.

**Parameters** **build_kernel_modules** – build the kernel modules for disabling the CPU caches too

temci.setup.setup.**script_relative**(*file: str*) → str
Returns the absolute version of the passed file name. :param file: passed file name relative to the scripts directory

### Module contents

### temci.report package

### Submodules

### temci.report.report module

### temci.report.report_processor module

### temci.report.rundata module

### temci.report.stats module

### temci.report.testers module

### Module contents

This module is about generating meaningful reports an working with the resulting measurements of serveral benchmarks.

### temci.utils package

### Submodules

### temci.utils.click_helper module

### temci.utils.config_utils module

Types shared between different file config definitions

### temci.utils.library_init module

### temci.utils.mail module

Utilities to send mails.

`temci.utils.mail.`**`hostname`**`()` → str
Returns the hostname of the current machine

`temci.utils.mail.`**`send_mail`**(*recipient: str*, *subject: str*, *content: str*, *attached_files: List[str] = None*)
Sends a mail to the recipient with the passed subject, content and attached files.

**Parameters**

- **`recipient`** – recipient of the mail, i.e. a mail address
- **`subject`** – subject of the mail
- **`content`** – content of the mail
- **`attached_files`** – optional list of names of files that are attached to the mail

### temci.utils.number module

**`class`** `temci.utils.number.`**`FNumber`**(*number: Union[int, float]*, *rel_deviation: Union[int, float] = None*, *abs_deviation: Union[int, float] = None*, *is_percent: bool = None*, *scientific_notation: bool = None*, *parentheses_mode: Union[str,* temci.utils.number.ParenthesesMode*] = None*, *parentheses: bool = None*)
Bases: `object`

A formattable number wrapper.

Configuration format, is in the settings under report/number

```
Don't omit the minimum number of decimal places if insignificant?
force_min_decimal_places: Bool()
            default: true

The maximum number of decimal places
max_decimal_places: Int(constraint=<function>)
            default: 5

The minimum number of shown decimal places if decimal places are shown
min_decimal_places: Int(constraint=<function>)
```

(continues on next page)

(continued from previous page)

```
            default: 3

# Omit insignificant decimal places
omit_insignificant_decimal_places:         Bool()

# Show parentheses around non significant digits? (If a std dev is given)
parentheses:         Bool()
            default: true

# Mode for showing the parentheses: either d (Digits are considered significant␣
↪if they don't change
# if the number itself changes += $sigmas * std dev) or o (digits are␣
↪consideredsignificant if they
# are bigger than $sigmas * std dev)
parentheses_mode:         ExactEither('d'|'o')
            default: o

# Show as percentages
percentages:         Bool()

# Use the exponential notation, i.e. '10e3' for 1000
scientific_notation:         Bool()
            default: true

# Use si prefixes instead of 'e...'
scientific_notation_si_prefixes:         Bool()
            default: true

# Number of standard deviation used for the digit significance evaluation
sigmas:         Int(constraint=<function>)
            default: 2
```

deviation
Relative deviation

format() → str

classmethod init_settings(*new_settings: Dict[str, Union[int, bool]]*)

settings = {'force_min_decimal_places': True, 'max_decimal_places': 5, 'min_decimal_

settings_format = # Don't omit the minimum number of decimal places if insignificant? :

temci.utils.number.**Number**
Numeric type

alias of Union[int, float]

class temci.utils.number.**ParenthesesMode**(*value*)
Bases: enum.Enum

An enumeration.

DIGIT_CHANGE = 'd'

ORDER_OF_MAGNITUDE = 'o'

classmethod map(*key: Union[str,* ParenthesesMode*]*) → *temci.utils.number.ParenthesesMode*

temci.utils.number.**fnumber**(*number: Union[int, float]*, *rel_deviation: Union[int, float] = None*, *abs_deviation: Union[int, float] = None*, *is_percent: bool = False*) → str

temci.utils.number.**format_number**(*number: Union[int, float], deviation: float = 0.0, parentheses: bool = True, explicit_deviation: bool = False, is_deviation_absolute: bool = True, min_decimal_places: int = 3, max_decimal_places: Optional[int] = None, omit_insignificant_decimal_places: bool = True, scientific_notation: bool = False, scientific_notation_steps: int = 3, scientific_notation_decimal_places: int = None, scientific_notation_si_prefixes: bool = True, force_min_decimal_places: bool = True, relative_to_deviation: bool = False, sigmas: int = 2, parentheses_mode: temci.utils.number.ParenthesesMode = <ParenthesesMode.ORDER_OF_MAGNITUDE: 'o'>*) → str

Format the passed number

```
>>> format_number(1.0, 0.5)
'1.(000)'
```

```
>>> format_number(1.56, 0.005)
'1.56(0)'
```

```
>>> format_number(1560, scientific_notation=True)
'1.560k'
```

```
>>> format_number(1560, scientific_notation_si_prefixes=False, scientific_
↪notation=True)
'1.560e3'
```

```
>>> format_number(float("inf"))
'inf'
```

Parameters

- **number** – formatted number
- **deviation** – standard deviation associated with the number
- **parentheses** – show parentheses around non significant digits?
- **explicit_deviation** – show the absolute deviation, e.g. "100±456.4"
- **is_deviation_absolute** – is the given deviation absolute?
- **min_decimal_places** – the minimum number of shown decimal places if decimal places are shown
- **max_decimal_places** – the maximum number of decimal places
- **omit_insignificant_decimal_places** – omit insignificant decimal places
- **scientific_notation** – use the exponential notation, i.e. "10e3" for 1000
- **scientific_notation_steps** – steps in which the exponential part is incremented
- **scientific_notation_decimal_places** – number of decimal places that are shown in the scientic notation
- **scientific_notation_si_prefixes** – use si prefixes instead of "e…"
- **force_min_decimal_places** – don't omit the minimum number of decimal places if insignificant?

- **relative_to_deviation** – format the number relative to its deviation, i.e. "10 sigma"
- **sigmas** – number of standard deviations for significance
- **parentheses_mode** – mode for selecting the significant digits

Returns the number formatted as a string

temci.utils.number.**format_number_sn**(*number: Union[int, float]*, *scientific_notation_steps: int = 3*, *deviation: Optional[float] = None*, *decimal_places: int = None*, *si_prefixes: bool = True*, ***kwargs*)

temci.utils.plugin module

Utilities for loading plugins. Plugins are python files (ending *.py*) that are loaded prior to building the cli. These files may e.g. add runners, plugins, ...

Plugins are loaded from the application directory (*~/.temci*) and from the paths given in the environment variable *TEMCI_PLUGIN_PATH* which contains a colon separated list of paths.

temci.utils.plugin.**load_plugins**()
Load the plugins from the plugin folders :return:

temci.utils.plugin.**plugin_paths**() → List[str]
Returns the paths that plugins are located in (might return folders and files)

temci.utils.registry module

temci.utils.settings module

temci.utils.sudo_utils module

temci.utils.typecheck module

Implements basic type checking for complex types.

Why? Because it's nice to be able to type check complex structures that come directly from the user (e.g. from YAML config files).

The Type instance are usable with the standard isinstance function:

```
isinstance(4, Either(Float(), Int()))
```

Type instances also support the "&" (producres All(one, two)) and "|" (produces Either(one, two)) operators. The above sample code can therefore be written as:

```
isinstance(4, Float() | Int())
```

The native type wrappers also support custom constraints. With help of the fn module one can write:

```
t = Float(_ > 0) | Int(_ > 10)
isinstance(var, t)
```

"t" is a Type that matches only floats greater than 0 and ints greater than 10.

For more examples look into the test_typecheck.py file.

class temci.utils.typecheck.**All**(**types:* Tuple[temci.utils.typecheck.Type])
Bases: *temci.utils.typecheck.Type*

Checks for the value to be of all of several types.

Creates an All instance.

Parameters **types** – list of types (or SpecialType subclasses)

Raises ConstraintError if some of the contraints aren't (typechecker) Types

types
Expected types

class temci.utils.typecheck.**Any**(*completion_hints:* Dict[*str*, Any] = *None*)
Bases: *temci.utils.typecheck.Type*

Checks for the value to be of any type.

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

class temci.utils.typecheck.**Bool**
Bases: *temci.utils.typecheck.Type*, click.types.ParamType

Like Bool but with a third value none that declares that the value no boolean value. It has None as its default value (by default).

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

completion_hints
Completion hints for supported shells for this type instance

name = 'bool'
click.ParamType name, that makes this class usable as a click type

class temci.utils.typecheck.**BoolOrNone**
Bases: *temci.utils.typecheck.Type*, click.types.ParamType

Like Bool but with a third value none that declares that the value no boolean value. It has None as its default value (by default).

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

completion_hints
Completion hints for supported shells for this type instance

convert(*value*, *param*, *ctx: click.core.Context*) → *Optional*[bool]
Convert method that makes this class usable as a click type.

default
The default value of this instance

name = 'bool_or_none'
click.ParamType name, that makes this class usable as a click type

class temci.utils.typecheck.**CompletionHint**(***hints*)
Bases: object

A completion hint annotation for a type. Usage example:

```
Int() // Completion(zsh="_files")
```

hints
Completion hints for every supported shell

class temci.utils.typecheck.**Constraint**(*constraint: Callable[[Any], bool], constrained_type: temci.utils.typecheck.Type = Any, description: str = None*)
Bases: *temci.utils.typecheck.Type*

Checks the passed value by an user defined constraint.

Creates an Constraint instance.

Parameters

- **constraint** – function that returns True if the user defined constraint is satisfied
- **constrained_type** – Type that the constraint is applied on
- **description** – short description of the constraint (e.g. ">0")

Raises ConstraintError if constrained_type isn't a (typechecker) Types

constrained_type
Type that the constraint is applied on

constraint
Function that returns True if the user defined constraint is satisfied

description
Short description of the constraint (e.g. ">0")

string_representation(*indents: int = 0, indentation: int = 4, str_list: bool = False, defaults=None*) → Union[str, *List*[str]]
Produce a YAML string that contains the default value (if possible), the description of this type and more and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

class temci.utils.typecheck.**Default**(*default*)
Bases: object

A default value annotation for a Type. Usage example:

```
Int() // Default(3)
```

Especially useful to declare the default value for a key of an dictionary. Allows to use Dict(...).get_default() -> dict.

default
Default value of the annotated type

class temci.utils.typecheck.**Description**(*description: str*)
Bases: object

A description of a Type, that annotates it. Usage example:

```
Int() // Description("Description of Int()")
```

description
Description string

class temci.utils.typecheck.**Dict**(*data: Dict[Any, temci.utils.typecheck.Type] = None*, *unknown_keys: bool = False*, *key_type: temci.utils.typecheck.Type = Any*, *value_type: temci.utils.typecheck.Type = Any*)
Bases: *temci.utils.typecheck.Type*

Checks for the value to be a dictionary with expected keys and values satisfy given type constraints.

Creates a new instance.

Parameters

- **data** – dictionary with the expected keys and the expected types of the associated values
- **unknown_keys** – accept unknown keys in value
- **key_type** – expected Type of all dictionary keys
- **value_type** – expected Type of all dictionary values

Raises ConstraintError if one of the given types isn't a (typechecker) Types

get_default() → dict
Returns the default value of this type :raises: ValueError if the default value isn't set

get_default_yaml(*indent: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → str
Produce a YAML like string that contains the default value and the description of this type and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

get_description(*key: str*) → str
Returns the description for the passed key or None if there isn't one.

Parameters **key** – passed key

has_default() → bool
Does this type instance have an default value?

is_obsolete(*key: str*) → bool
Is the type that belongs to the key Obsolete?

Parameters **key** – dict key

Returns obsolete?

key_type
Expected Type of all dictionary keys

obsoleteness_reason(*key: str*) → *Optional*[temci.utils.typecheck.Obsolete]
Return the obsoleteness reason (the Obsolete type object) if the key is obsolete

Parameters **key** – dict key

Returns Obsolete object or none

string_representation(*indents: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → Union[str, *List*[str]]

Produce a YAML string that contains the default value (if possible), the description of this type and more and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

unknown_keys

Fail if value contains unknown keys

value_type

Expected Type of all dictionary values

class temci.utils.typecheck.**DirName**(*constraint: Callable[[*Any*], bool] = None*)

Bases: *temci.utils.typecheck.Str*

A valid directory name. If the directory doesn't exist, at least the parent directory must exist.

Creates an instance.

Parameters **constraint** – function that returns True if the user defined constraint is satisfied

completion_hints

Completion hints for supported shells for this type instance

constraint

Function that returns True if the user defined constraint is satisfied

temci.utils.typecheck.**E**(*exp_value*) → *temci.utils.typecheck.Exact*

Alias for Exact.

class temci.utils.typecheck.**Either**(**types: tuple*)

Bases: *temci.utils.typecheck.Type*

Checks for the value to be of one of several types.

Creates an Either instance.

Parameters **types** – list of types (or SpecialType subclasses)

Raises ConstraintError if some of the contraints aren't (typechecker) Types

types

Possible types

class temci.utils.typecheck.**Exact**(*exp_value*)

Bases: *temci.utils.typecheck.Type*

Checks for value equivalence.

Creates an Exact object.

Parameters **exp_value** – value to check for

exp_value

Expected value

class temci.utils.typecheck.**ExactEither**(**exp_values: tuple*)
Bases: *temci.utils.typecheck.Type*

Checks for the value to be of one of several exact values.

Creates an ExactEither instance.

Parameters **exp_values** – list of types (or SpecialType subclasses)

Raises ConstraintError if some of the contraints aren't (typechecker) Types

exp_values
Expected values

class temci.utils.typecheck.**FileName**(*constraint: Callable[[Any], bool] = None*, *allow_std: bool = False*, *allow_non_existent: bool = True*)
Bases: *temci.utils.typecheck.Str*

A valid file name. If the file doesn't exist, at least the parent directory must exist and the file must be creatable.

Creates an instance.

Parameters

- **constraint** – function that returns True if the user defined constraint is satisfied
- **allow_std** – allow '-' as standard out or in
- **allow_non_existent** – allow files that don't exist

allow_non_existent
Allow files that don't exist

allow_std
Allow '-' as standard out or in

completion_hints
Completion hints for supported shells for this type instance

constraint
Function that returns True if the user defined constraint is satisfied

temci.utils.typecheck.**FileNameOrStdOut**() → *temci.utils.typecheck.FileName*
A valid file name or "-" for standard out.

temci.utils.typecheck.**Float**(*constraint: Callable[[Any], bool] = None*) → Union[*temci.utils.typecheck.T*, *temci.utils.typecheck.Constraint*]
Alias for Constraint(constraint, T(float)) or T(float)

Parameters **constraint** – function that returns True if the user defined constraint is satisfied

class temci.utils.typecheck.**Info**(*value_name: str = None*, *value=None*, *_app_str: str = None*)
Bases: object

Information object that is used to produce meaningful type check error messages.

Creates a new info object.

Parameters

- **value_name** – name of the value that is type checked
- **value** – value that is type checked

add_to_name(*app_str: str*) → *temci.utils.typecheck.Info*
Creates a new info object based on this one with the given appendix to its value representation. It's used to give information about what part of the main value is currently examined.

Parameters **app_str** – app string appended to the own app string to create the app string for the new info object

Returns new info object

errormsg(*constraint:* temci.utils.typecheck.Type, *value*, *msg: str = None*) → temci.utils.typecheck.InfoMsg

Creates an info message object with the passed expected type and the optional message.

Parameters

- **constraint** – passed expected type
- **value** – value that violates th constraint
- **msg** – additional message, it should give more information about why the constraint isn't met

errormsg_cond(*cond: bool*, *constraint:* temci.utils.typecheck.Type, *value*, *msg: str = None*) → temci.utils.typecheck.InfoMsg

Creates an info message object with the passed expected type and the optional message.

Parameters

- **cond** – if this is false *InfoMsg(True)* is returned.
- **constraint** – passed expected type
- **value** – value that violates th constraint
- **msg** – additional message, it should give more information about why the constraint isn't met

errormsg_key_non_existent(*constraint:* temci.utils.typecheck.Type, *key: str*) → temci.utils.typecheck.InfoMsg

Creates an info message object with the passed expected type that contains the message that currently examined part of the value is unexpected.

Parameters **constraint** – passed expected type

errormsg_unexpected(*key: str*) → temci.utils.typecheck.InfoMsg

Creates an info message object with the passed expected type that contains the message that currently examined part of the value has an unexpected key.

Parameters **key** – the unexpected key

get_value() → *Any*

Get the main value of this object. :raises: ValueError if the main value isn't set

has_value

Is the value property of this info object set to a meaningful value?

set_value(*value*)

Set the main value of this object

value

Main value that is type checked

value_name

Name of the value that is typechecked

wrap(*result: bool*) → temci.utils.typecheck.InfoMsg

Wrap the passed bool into a InfoMsg object.

class temci.utils.typecheck.**Int**(*constraint: Callable[[Any], bool] = None*, *range: range = None*, *description: str = None*)

Bases: *temci.utils.typecheck.Type*

Checks for the value to be of type int and to adhere to some constraints.

Creates an instance.

Parameters

- **constraint** – function that returns True if the user defined constraint is satisfied
- **range** – range (or list) that the value has to be part of
- **description** – description of the constraints

completion_hints
Completion hints for supported shells for this type instance

constraint
Function that returns True if the user defined constraint is satisfied

description
Description of the constraints

range
Range (or list) that the value has to be part of

class temci.utils.typecheck.**List**(*elem_type:* temci.utils.typecheck.Type = *Any*)

Bases: *temci.utils.typecheck.Type*

Checks for the value to be a list with elements of a given type.

Creates a new instance.

Parameters **elem_type** – type of the list elements

Raises ConstraintError if elem_type isn't a (typechecker) Types

elem_type
Expected type of the list elements

get_default() → *Any*
Returns the default value of this type :raises: ValueError if the default value isn't set

get_default_yaml(*indents: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → Union[str, *List*[str]]
Produce a YAML like string that contains the default value and the description of this type and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

class temci.utils.typecheck.**ListOrTuple**(*elem_type:* temci.utils.typecheck.Type = *Any*)

Bases: *temci.utils.typecheck.Type*

Checks for the value to be a list or tuple with elements of a given type.

Creates an instance.

Parameters **elem_type** – type of the list or tuple elements

Raises ConstraintError if elem_type isn't a (typechecker) Types

elem_type
Expected type of the list or tuple elements

temci.utils.typecheck.**NaturalNumber**(*constraint: Callable[[Any], bool] = None*) → *temci.utils.typecheck.Int*
Matches all natural numbers (ints >= 0) that satisfy the optional user defined constrained.

Parameters **constraint** – function that returns True if the user defined constraint is satisfied

class temci.utils.typecheck.**NonErrorConstraint**(*constraint: Callable[[Any], Any], error_cls: type, constrained_type: temci.utils.typecheck.Type = Any, description: str = None*)
Bases: *temci.utils.typecheck.Type*

Checks the passed value by an user defined constraint that fails if it raises an error.

Creates a new instance

Parameters

- **constraint** – function that doesn't raise an error if the user defined constraint is satisfied
- **error_cls** – class of the errors the constraint method raises
- **constrained_type** – Type that the constraint is applied on
- **description** – short description of the constraint (e.g. ">0")

Raises ConstraintError if constrained_type isn't a (typechecker) Types

constrained_type
Type that the constraint is applied on

constraint
Function that returns True if the user defined constraint is satisfied

description
Short description of the constraint (e.g. ">0")

error_cls
Class of the errors the constraint method raises

class temci.utils.typecheck.**NonExistent**(*completion_hints: Dict[str, Any] = None*)
Bases: *temci.utils.typecheck.Type*

Checks a key of a dictionary for existence if its associated value has this type.

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

class temci.utils.typecheck.**Optional**(*other_type: temci.utils.typecheck.Type*)
Bases: *temci.utils.typecheck.Either*

Checks the value and checks that its either of native type None or of another Type constraint. Alias for Either(Exact(None), other_type)

Creates an Optional instance.

Parameters **other_type** – type to make optional

Raises ConstraintError if other_type isn't a (typechecker) Types

temci.utils.typecheck.**PositiveInt**(*constraint: Callable[[Any], bool] = None*) → *temci.utils.typecheck.Int*

Matches all positive integers that satisfy the optional user defined constrained.

Parameters **constraint** – function that returns True if the user defined constraint is satisfied

class temci.utils.typecheck.**Str**(*constraint: Callable[[Any], bool] = None*)

Bases: *temci.utils.typecheck.Type*

Checks for the value to be a string an optionally meet some constraints.

Creates an instance.

Parameters **constraint** – function that returns True if the user defined constraint is satisfied

constraint

Function that returns True if the user defined constraint is satisfied

class temci.utils.typecheck.**StrList**

Bases: *temci.utils.typecheck.Type*, click.types.ParamType

A comma separated string list which contains elements from a fixed set of allowed values.

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

allowed_values

Possible values that can appear in the string list, if None all values are allowed.

convert(*value*, *param*, *ctx: click.core.Context*) → *List*[str]

Convert method that makes this class usable as a click type.

get_default_yaml(*indents: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → str

Produce a YAML like string that contains the default value and the description of this type and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

name = 'comma_sep_str_list'

click.ParamType name, that makes this class usable as a click type

class temci.utils.typecheck.**T**(*native_type: type*)

Bases: *temci.utils.typecheck.Type*

Wrapper around a native type.

Creates an isntance.

Parameters **native_type** – wrapped native type

native_type

Native type that is wrapped

class temci.utils.typecheck.**Tuple**(**elem_types:* Tuple[temci.utils.typecheck.Type])

Bases: *temci.utils.typecheck.Type*

Checks for the value to be a tuple (or a list) with elements of the given types.

Creates a new instance.

Parameters **elem_types** – types of each tuple element

Raises ConstraintError if elem_type isn't a (typechecker) Types

elem_types
Expected type of each tuple element

class temci.utils.typecheck.**Type**(*completion_hints: Dict[str, Any] = None*)
Bases: object

A simple type checker type class.

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

completion_hints
Completion hints for supported shells for this type instance

default
Default value of this type instance

description
Description of this type instance

dont_typecheck_default() → *temci.utils.typecheck.Type*
Disable type checking the default value. :return: self

get_default() → *Any*
Returns the default value of this type :raises: ValueError if the default value isn't set

get_default_yaml(*indents: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → Union[str, *List*[str]]
Produce a YAML like string that contains the default value and the description of this type and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

has_default() → bool
Does this type instance have an default value?

string_representation(*indents: int = 0*, *indentation: int = 4*, *str_list: bool = False*, *defaults=None*) → Union[str, *List*[str]]
Produce a YAML string that contains the default value (if possible), the description of this type and more and its possible sub types.

Parameters

- **indents** – number of indents in front of each produced line
- **indentation** – indentation width in number of white spaces
- **str_list** – return a list of lines instead of a combined string?
- **defaults** – default value that should be used instead of the default value of this instance

typecheck_default
Type check the default value

class temci.utils.typecheck.**ValidTimeSpan**
Bases: *temci.utils.typecheck.Type*, click.types.ParamType

A string that is parseable as timespan by pytimeparse. E.g. "32m" or "2h 32m".

Creates an instance.

Parameters **completion_hints** – completion hints for supported shells for this type instance

convert(*value*, *param*, *ctx: click.core.Context*) → int
Convert method that makes this class usable as a click type.

name = 'valid_timespan'
click.ParamType name, that makes this class usable as a click type

class temci.utils.typecheck.**ValidYamlFileName**(*allow_non_existent: bool = False*)
Bases: *temci.utils.typecheck.Str*

A valid file name that refers to a valid YAML file.

Create an instance.

Parameters **allow_non_existent** – allow files that don't exist

allow_non_existent
Allow files that don't exist

completion_hints
Completion hints for supported shells for this type instance

temci.utils.typecheck.**YAML_FILE_COMPLETION_HINT = "_files -g '*\\.yaml'"**
YAML file name completion hint for ZSH

temci.utils.typecheck.**typecheck**(*value*, *type: Union[temci.utils.typecheck.Type, type]*, *value_name: str = None*)
Like verbose_isinstance but raises an error if the value hasn't the expected type.

Parameters

- **value** – passed value
- **type** – expected type of the value
- **value_name** – optional description of the value

Raises TypeError

temci.utils.typecheck.**typecheck_locals**(*locals: Dict[str, Any] = None*, ***variables: Dict[str, Union[temci.utils.typecheck.Type, type]]*)
Like typecheck but checks several variables for their associated expected type. The advantage against typecheck is that it sets the value descriptions properly. Example usage:

```
def func(a: str, b: int):
    typecheck_locals(locals(), a=Str(), b=Int())
```

Parameters

- **locals** – directory to get the variable values from
- **variables** – variable names with their associated expected types

Raises TypeError

temci.utils.typecheck.**verbose_isinstance**(*value*, *type: Union[temci.utils.typecheck.Type, type]*, *value_name: str = None*) → temci.utils.typecheck.InfoMsg

Verbose version of isinstance that returns a InfoMsg object.

Parameters

- **value** – value to check
- **type** – type or Type to check for
- **value_name** – name of the passed value (improves the error message)

temci.utils.util module

Utility functions and classes that don't depend on the rest of the temci code base.

class temci.utils.util.**InsertionTimeOrderedDict**

Bases: object

A dictionary which's elements are sorted by their insertion time.

classmethod from_list(*items: Optional[list]*, *key_func: Callable[[Any], Any]*) → *temci.utils.util.InsertionTimeOrderedDict*

Creates an ordered dict out of a list of elements.

Parameters

- **items** – list of elements
- **key_func** – function that returns a key for each passed list element

Returns created ordered dict with the elements in the same order as in the passed list

items() → List[Tuple[Any, Any]]

keys() → List

Returns all keys of this dictionary. They are sorted by their insertion time.

values() → List

Rerturns all values of this dictionary. They are sorted by their insertion time.

class temci.utils.util.**Singleton**

Bases: type

Singleton meta class. @see http://stackoverflow.com/a/6798042

temci.utils.util.**allow_all_imports = True**

Allow all imports (should the can_import method return true for every module)?

temci.utils.util.**can_import**(*module: str*) → bool

Can a module (like scipy or numpy) be imported without a severe and avoidable performance penalty? The rational behind this is that some parts of temci don't need scipy or numpy.

Parameters **module** – name of the module

temci.utils.util.**document**(***kwargs: Dict[str, str]*)

Document

Parameters **kwargs** – class attribute, documentation prefix

temci.utils.util.**does_command_succeed**(*cmd: str*) → bool

Does the passed command succeed (when executed by /bin/sh)?

temci.utils.util.**does_program_exist**(*program: str*) → bool
Does the passed program exist?

temci.utils.util.**geom_std**(*values: List[float]*) → float
Calculates the geometric standard deviation for the passed values. Source: https://en.wikipedia.org/wiki/Geometric_standard_deviation

temci.utils.util.**get_cache_line_size**(*cache_level: int = None*) → Optional[int]
Returns the cache line size of the cache on the given level. Level 0 and 1 are actually on the same level.

Parameters **cache_level** – if None the highest level cache is used

Returns cache line size or none if the cache on the given level doesn't exist

temci.utils.util.**get_distribution_name**() → str
Returns the name of the current linux distribution (requires *lsb_release* to be installed)

temci.utils.util.**get_distribution_release**() → str
Returns the used release of the current linux distribution (requires *lsb_release* to be installed)

temci.utils.util.**get_doc_for_type_scheme**(*type_scheme: Type*) → str
Return a class documentation string for the given type scheme. Use the default_yaml method.

temci.utils.util.**get_memory_page_size**() → int
Returns the size of a main memory page

temci.utils.util.**handler = <RainbowLoggingHandler <stderr> (NOTSET)>**
Colored logging handler that is used for the root logger

temci.utils.util.**has_pdflatex**() → bool
Is pdflatex installed?

temci.utils.util.**has_root_privileges**() → bool
Has the current user root privileges?

temci.utils.util.**in_standalone_mode = False**
In rudimentary standalone mode (executed via run.py)

temci.utils.util.**join_strs**(*strs: List[str]*, *last_word: str = 'and'*) → str
Joins the passed strings together with ", " except for the last to strings that separated by the passed word.

Parameters

- **strs** – strings to join
- **last_word** – passed word that is used between the two last strings

temci.utils.util.**on_apple_os**() → bool
Is the current operating system an apple OS X?

temci.utils.util.**parse_timespan**(*time: str*) → float
Parse a time span expression, see https://pypi.org/project/pytimeparse/

Supports -1 to express an infinite time span

Parameters **time** – time span expression, mixture of different time units is possible

Returns time span in seconds

class temci.utils.util.**proc_wait_with_rusage**
Bases: object

Each Popen object gets a field rusage

temci.utils.util.**recursive_exec_for_leafs**(*data: dict*, *func*, *_path_prep=[]*)
Executes the function for every leaf key (a key without any sub keys) of the data dict tree.

Parameters

- **data** – dict tree
- **func** – function that gets passed the leaf key, the key path and the actual value

temci.utils.util.**rusage_header**() → str

temci.utils.util.**sphinx_doc**() → bool
Is the code only loaded to document it with sphinx?

temci.utils.util.**warn_for_pdflatex_non_existence_once**(*_warned=[False]*)
Log a warning if the pdflatex isn't available, but only if this function is called the first time

temci.utils.vcs module

Module contents

Package with utility modules.

7.13.2 Module contents

7.14 Resources

This is a collection of additional resources that are related to temci.

Bachelor thesis The development of temci started as part of this bachelor thesis at the Karlsruhe Institute of Technology. It's written in German.

A talk in german on the topic of benchmarking in german:

- modindex
- search

PYTHON MODULE INDEX

t

INDEX

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y