QPar Quickstart Guide

This Quickstart Guide helps the reader to quickly get going with QPar. It will cover topics like solving a specific formula, run an evaluation or implementing a new back end in QPar. This guide is valid for version 1.0 of QPar, as tagged in the current official repository. All topics are subject to change due to active development. In doubt consult the README file provided with the package.

Compiling the Package

QPar v1.0 relies on Apache Ant for building. To generate the executables, use the `ant build` command.

Hint: When recompiling do not forget to issue `ant clean`, or a possibly modified log4j.conf is not copied to the `bin` directory.

Configure QPar

The main configuration file of QPar is located `qpar.conf` in the project root directory. The configurable properties are listed here.

availableProcessors

This option defines the number of processors/cores the slave can use for parallel computations. Entering the value `0` tells QPar to ask the runtime for the number of cores on the system using the `Runtime.getRuntime().availableProcessors()` call.

It is advisable to use less than the maximum available cores, to leave computing capacity for the OS and other running services (like *SSH*).

mailServer, mailUser, mailPass

These properties are used to supply QPar with a SMTP server and credentials to enable its mail capabilities. Per default QPar uses *StartTLS* and the *submission* port. This configuration was tested with the *gmail* service. Other providers may need different configurations, which can be changed in the `qpar.master.Mailer` class.

mailEvaluationResults, evaluationAddress

The Boolean property `mailEvaluationResults` decides if QPar sends the results of an evaluation via e-mail to the address defined in the property `evaluationAddress`.

enableExceptionNotifications, exceptionNotifierAddress

The Boolean property `enableExceptionNotifications` decides if QPar sends occurred exceptions via e-mail to the address defined in the property `exceptionNotifierAddress`.

benchmarkMode

This Boolean setting enables/disables the *virtual cluster* result merging. If enabled, results from the slaves are merged in the order they would have arrived if started synchronously. If false, the results are merged on a first-come first-serve basis.

resultCaching

This Boolean setting enables/disables a global cache for results. Hashes of sub-formulas are used as keys to store the satisfiability of the sub-formula. This setting can save time in production environments if a lot of similar formulas are tested.

Hint: Do not enable this property with evaluations.

An example of a qpar.conf is shown in Figure 0.1.

```
# Qpar configuration file

# set max available processors (0 = max of system)
availableProcessors = 4

# Mailing
mailServer       =smtp.gmail.com
mailUser         =
mailPass         =

# Send results per mail
mailEvaluationResults = true
evaluationAddress =

# Exception Notifications
enableExceptionNotifications = false
exceptionNotifierAddress =

# benchmarking (avg case is near worst case) or
    production (tqbfs are merged as they arrive)
benchmarkMode = true
#benchmarkMode = false

# result cache. chances are, different interpretations
    lead to the same
# reduced subformula. results of subformulas are stored
    centrally on the master
# and checked prior to execution of a solver for a
    matching formula
# SET TO FALSE IF BENCHMARKING OR EXPECT FUNNY RESULTS
#resultCaching = true
resultCaching = false

# Identifier and class name of output plugins
plugin.qpro = qpar.slave.solver.QProPlugin
plugin.cirqit = qpar.slave.solver.CirqitPlugin
```

Figure 0.1: Example of qpar.conf

Solving a Single Formula

To compute a single formula the `newjob` and `startjob` commands of the built-in shell are needed. There are several ways to get to the shell:

- Start the *master* by the Ant command `ant master` (the file `batch.txt` in the project root is automatically as input)
- Start the master by the provided BASH script `qpar_master.sh` (or directly by a java command)
- Pass a input file to the shell interpreter (eg. `qpar_master.sh -i batch.txt`)

Example:
```
newjob path_to_formula/formula.qbf report.txt qpro simple 16 60
startjob 1
```

This job starts the computation of the formula in the `formula.qbf` file. The formula is divided into 16 sub-problems having a timeout of 60 seconds. *Qpro* is used as back end and for splitting the *simple* heuristic.

Hint: Check if the back end executable is in the OS path. Otherwise error message like `''Qpro returned no result. Treating as error.''` will occur.

QPar 1.0 contains the following heuristic identifiers:

- simple,
- rand,
- litcount,
- probnet,
- shallow, and
- edgecount.

For the description of the heuristics please refer to Chapter **??** of this thesis.

Running an Evaluation

An evaluation runs computations for all combinations of

- files in the specified directory,
- heuristics listed in the `getAvailableHeuristics()` method of the `HeuristicFactory` class and
- specified range of cores to be used (in steps of 2^n).

The result of an evaluation will be written in the specified directory into the file
`evaluation.txt` and printed to the console.

To start an evaluation issue use the `evaluate` command of the shell.
Example:

```
evaluate path_to_directory 1 16 qpro 60
```

This command starts an evaluation of the directory `path_to_directory` with a timeout of 60 seconds per formula. The command will run computations with 1, 2, 4, 8 and 16 cores with all available heuristics.

Implementing a New Back End

To use a new back end with QPar, a wrapper class for this solver must be implemented. This wrapper must implement the interface `SolverPlugin`. Since `SolverPlugin` extends
`Runnable` the wrapper also must implement a `void run()` method.

The calling order of the methods of the wrapper is as follows:

1. Call to `initialize(ReducedInterpretation ri)`. This method should prepare the computation, like transforming the formula tree to the back end input format.
2. Call to `run()` in its own thread. Calling the back end and interpreting the input takes place here.
3. Call to `Boolean waitForResult()`. This method has to block until the computation is complete, and returns the result or throws an exception in the error case.
4. Call to `kill()` if the computation has to be aborted.

The implemented wrapper is to be placed in the classpath of QPar. Keys in the configuration property file starting with “plugin” are interpreted as plugin information. To add a back end to QPar with the identifier “testsolver” and the full class name `package.TestSolverPlugin` we would add the following line to the qpar.conf file:

```
plugin.testsolver = package.TestSolverPlugin
```

Implementing a new heuristic

Heuristics have to extend the `qpar.master.heuristic.Heuristic` abstract class. The method

```
abstract LinkedHashSet<Integer> sortGroup(Set<Integer>, Qbf)
```

receives a set of re-orderable variables and the original formula. This method shall return an ordered list of variables in which the first variables are assigned first (if they are assigned at all, depending on the number of sub-problems).

To make the heuristic available, it must be added to the methods

`getAvailableHeuristics()` and `getHeuristic()`

in the class `qpar.master.heuristic.HeuristicFactory`. To add heuristics only for debugging or testing purposes, omit the addition to the

`getAvailableHeuristics()`

method. Evaluations will then not use this heuristic.