1 Installing

Just compile the Murphi compiler:

`cd src; make clean; make`

The final executable name is `mu`.

2 Using the examples

Predefined examples are in the directory ex. In every subdirectory of ex/ (except for scripts) there is at least one protocol example (extension .m).

2.1 Using Makefiles

In every subdirectory of ex/ (except for scripts) there are also 6 makefiles:

Makefile.FULL creates model.C from model.m, then compiles model.C in order to obtain the executable model, i.e. the standard Murphi verifier. Without parameters, compiles all the (predefined) model.m in the directory. Possible parameters: a single model, or "clean" (deletes .C and executables files). There are two Makefile constants which are left for user specification:

`MURPHIOPT` to specify extra options to the Murphi compiler (when generating model.C)

`CFLAGSOPT` to specify extra compiler options when compiling model.C

Makefile.FULL.bc the same as Makefile.FULL, but enables hash compaction (only if model.C does not exist or is out-of-date w.r.t. model.m)

Makefile.MPI the same as Makefile.FULL, but creates an executable of Eddy_Murphi (only if model.C does not exist or is out-of-date w.r.t. model.m).

Makefile.MPI.bc the same as Makefile.MPI, but enables hash compaction (see Makefile.FULL.bc)

Makefile.MPI.bc.old the same as Makefile.MPI.bc, but creates an executable of an MPI porting of Stern's PMurphi (hash compaction is required).

Makefile link to Makefile.MPI.bc

Thus, for example, to make the model adash in original Murphi with hash compaction (from ex/dash directory), just type:

`make -f Makefile.FULL.bc adash; ./adash -m`*m* `-ndl`

On the other hand, to compile the same model with Eddy_Murphi (hash compaction enabled), just type:

`make adash`

Note that `make -f Makefile.FULL adash MURPHIOPT=''-b -c''` would have produced the same result.

2.2 Running verifications

To run a verification with the original Murphi, the following options are useful:

-mn where n is the amount of RAM available for the verification (in MB)

-ndl disables deadlock detection

-pn prints verification progress reports every 10^n states.

To run a verification with Eddy_Murphi, the same options as above are still available (the -m option of course is meant to give the amount of memory on each process), plus these two:

-bsv$n$ is the value for BUFSIZE, i.e. there will be n states in each queue line (default is 1024)

-bcv$n$ is the value for BUFCOUNT, i.e. there will be n queue lines (default is 8)

This is a typical run[1] for Eddy_Murphi (from ex/dash directory):
`lamboot; mpirun -np` n `adash -m`m `-ndl; lamhalt`
where m is the amount of RAM available for the verification for each process (in MB) and n is the number of processes on which to run the verification. In this case, BUFSIZE is 1024 and BUFCOUNT is 8.

Note however that error tracing is still not completely tested, and that, in order to enable it, it is necessary to compile defining the `HASHC_TRACE` constant (e.g. `make -f Makefile.MPI.bc adash CFLAGSOPT=''-DHASHC_TRACE''`).

Note moreover that Eddy_Murphi typically needs to generate huge files during the verification (especially if error tracing is required). To let user to put these files in different directories thatn the current one, the following options are available:

-dq *path* sets where to put the files for the BF queue

-d *path* sets where to put the files for the error tracing

Note that on some Linux systems, it is not allowed to create files larger than 2GB. If you're using one of these systems, define the `SPLITFILE` constant when compiling (e.g. `make -f Makefile.MPI.bc adash CFLAGSOPT=''-DHASHC_TRACE -DSPLITFILE''`).

[1] Note however that some MPI implementations do not require `lamboot` and `lamhalt`, and require more options to be specified to `mpirun`.

2.3 Using PBS

In order to run a verification with Eddy_Murphi, also a PBS script is available - for hosts on which PBS is installed[2]. The script is in the scripts/ directory, and its name is create_PBS.script. It can be executed in this way (from ex directory):

`./scripts/create_PBS.script -f <protocol> -N` n `[-ppn` p`] [-vo \(<opts>\)] [-mtf <max times file>] [-keep] [-m <mem per node>] [-nompi] [-noexec]`

where:

- <protocol> is the name of the executable for the Eddy_Murphi verification (with absolute or relative path; if the file doesn't exists, the script tries to make it up using Makefile.MPI.bc in the same directory);
- <n> is the number of nodes on which to run the verification;
- <p> is the number of processes for each node (default is 1);
- <opts> are the verification options for <protocol> (note that \(and \) are mandatory, default is no option);
- <queue name> is the name of the queue in the PBS (default is the first enabled queue)
- <max time> is the maximum verification time allowed (defualt is the maximum walltime defined for the current queue)
- <max times file> is a text file where each line is of type <from_nodes> <to_nodes> <max time>, meaning that using a number of nodes between <from_nodes> and <to_nodes> the maximum allowed time is <max time> (useful when usage policies are defined)
- the option -keep, if given, prevents the deleting of the qsub script.
- <mem per node> is the amount of memory (in MB) that PBS has to reserve on each node
- the option -nompi, if given, allows to run the verification without using mpirun (thus standalone programs not using MPI)
- if the option -noexec is given then the PBS script is generated but not executed.

Thus, for example (from ex directory),

`./scripts/create_PBS.script -f dash/ldash -N 7 -ppn 1 -vo \(-p4 -ndl -m100\)`

will post to PBS the request of the verification for the protocol ldash with options -p4 -ndl -m100, using 7 nodes (1 process per node). If dash/ldash does not exist, the script will create it by using dash/Makefile.MPI.bc.

[2]This script only works for those systems on which the note 1 does not hold.