Assisted ontology translation and verification with theorem provers
— Manual —

Torsten Hahmann

March 7, 2013

1 The COLORE tools

The COLORE tools have been developed to assist the ontology design and verification by automatically translating theories and their import closures. It mainly consists of a single python script, `colore-prover.py`, that comes with several options for different tasks.

Each Common Logic module is assumed to have the filename `modulename.clif` with the namespace defining the directory. For example, the module `inc/axioms/lineparts_segs` should be located in the directory `inc/axioms` (relative to the current path) and named `lineparts_segs.clif`.

2 Setup

These are step-by-step instructions for setting up the COLORE tools.

Step 1: Install SWI Prolog Download and follow the instructions from `http://www.swi-prolog.org/`.

Step 2: Install Python 2.7 Download and follow the instructions from `http://www.python.org/download/releases/2.7/`. Do not use use Python version 3.x as it is not backward compatible.

Step 3: Install Prover9-Mace4 Download and follow the instructions from `http://www.cs.unm.edu/~mccune/mace4/download/`.

Step 4: Install cltools Download and copy everything from `https://github.com/cmungall/cltools` to a local directory, we use `/stl/tmp/cltools/` as example directory.

Step 5: Copy colore-prover Copy all python files (ending in `.py`) for colore-prover to a local directory, for example to `/stl/tmp/cltools/`.

Step 6: Add directories to local PATH environment variable Add the directories where the binaries of Python, SWI Prolog, cltools, Mace4-Prover9 are located to the `PATH` variable.

For example, `/usr/bin/` (location of `python`), `/usr/local/bin/` (location of `swipl`), `/stl/tmp/cltools/bin/` (location of `clif-to-prover9`), `usr/local/prover9-mace4/bin/` (location of Prover9-Mace4 binaries).

You also want to add `/stl/tmp/cltools` (the location of `ColoreProver.py`) to your `PYTHONPATH` for ease of access.

In `tcsh` (TC shell) you can do this by adding the following to your `.tcshrc`. Make sure that you keep all other directories that were previously in the `PATH` variable.

```
setenv PATH /usr/bin:/usr/local/bin/:/stl/tmp/cltools/bin/:/usr/local/prover9-mace4/bin/
setenv PYTHONPATH /stl/tmp/cltools
```

Having the `PYTHONPATH` variable point to the folder of the colore-prover python scripts allows us to call them by their name alone using the `-m` option, omitting the full path. For example, we can use

```
python -m ColoreProver input
```

instead of

```
python /stl/tmp/cltools/ColoreProver.py input
```

Step 7: Change Prolog path in cltools In `cltools/bin/clif-to-prover9` change the first line so that the first part points to the location of the installed version of SWI Prolog (from Step 1).

```
#!/usr/local/bin/swipl -L0 -G0 -T0 -q -g main -t halt -s
```

If installing on Windows, remove this first line altogether.

Step 8: Create Prolog library index Create a Prolog library index within the `cltools` folder. To do so, go into that directory and call (assuming `swipl` is in the PATH):

```
swipl -g "make_library_index('.')" -t halt
```

More documentation: `http://www.swi-prolog.org/pldoc/doc_for?object=section(2,'2.13',swi('/doc/Manual/autoload.html'))`.

Step 9: Point to correct Prolog libraries Edit the top portion of `cltools/bin/clif-to-prover9` so that the `library_directory` points to the location of the `cltools` directory. The four `library` statements should then point to the relative directory of the Prolog libraries `cl`, `cl_io`, `clif_parser`, and `p9_writer` – all ending with `.pl`.

```
:- assertz(library_directory('/stl/tmp/')).

:- use_module(library('cltools/cl')).
:- use_module(library('cltools/cl_io')).
:- use_module(library('cltools/clif_parser')).
:- use_module(library('cltools/p9_writer')).
```

Step 10: Adapt the command to invoke clif-to-prover9 in `ClifModule.py` (only necessary for installation in Windows) In `ClifModule.py` change the attribute `CLIF_TO_PROVER9` to

```
CLIF_TO_PROVER9 = "swipl -L0 -G0 -T0 -q -g main -t halt -s clif-to-prover9 --"
```

Step 11: Test Type the following command, e.g., to test:

```
python -m ColoreProver /stl/tmp/cltools/t/overlaps
```

3 Simple tasks

3.1 Model construction

3.1.1 Mace4

To search for a model of a theory in prover9 format:

```
mace4  [-n size] -f input_files.p9 goal_file.p9 > goal_file.model
```

To check for all models of a particular size:

```
mace4 -n 24 -m -1 -f SPOLS input_files.p9 > goal_file.model

interpformat standard -f goal_file.model | isofilter check '^ v' output '^ v' > goal_file.out
```

3.1.2 Paradox3

Currently only in Windows, still awaiting a Linux version of Paradox.

```
paradox3 --verbose 2 --model [--tstp] input.tptp > input-paradox.model
```

3.2 Theorem Proving

3.2.1 Prover9

To prove a theorem in the goal_file about a theory (axiom_files) in prover9 format:

```
prover9 -f axiom_files.p9 goal_file.p9 >  goal_file.out
```

3.2.2 IProver

If the file is saved from Windows, it needs to be saved in linux, e.g. with pico.

```
iproveropt --clausifier /usr/local/bin/eprover
           --clausifier_options "--tstp-format --silent --cnf"
           input.tptp > input-iprover.out
```

4 Theory translation

4.1 From Common Logic to Prover9 syntax (LADR)

Common Logic uses double quotes (") for comments, these need to be replaced by single quotes (').

```
clif-to-prover9 input > output
```

Before running Prover9 on the generated files, the correctness of all parenthesis, especially those for ORs need to be verified. Moreover, all imports must be commented by %.

4.1.1 On Windows (from Maryam)

Just in case you would want to run clif-to-prover9 on a Windows machine. I removed the first line in clif-to-prover9, and then from the cltools folder I called the following command:

```
swipl -L0 -G0 -T0 -q -g main -t halt -s clif-to-prover9 -- input > output
```

4.2 Prover9 syntax (LADR) to TPTP syntax

```
ladr_to_tptp -q < input > output
```

Subsequently, all predicates listed at the beginning in single quotation marks must be replaced by names starting with a lowercase letter and only containing letters, numbers and the underscore. Moreover, some theorem provers on TPTP require distinct names for all axioms and other sentences.

5 Complex tasks

5.1 Translating a Common Logic theory to Prover9 syntax

Still work in progress, there is no separate option yet available. We can use the standard option that tries to verify consistency:

```
python -m ColoreProver input_theory
```

All imported modules are also translated, each module is located in the same directory as the original `.clif` file with the same name and the ending `.p9`.

5.2 Translating a Common Logic theory to TPTP syntax

We use the option `--tptp` to generate a single file TPTP-syntax output. One or several instances of the optional option `-s=ReplacedSymbol:ReplacementSymbol` can be used to automate the renaming of symbols that are unacceptable as relations or functions in TPTP syntax, e.g. `-s='<':less` renames the predicate `<` to `less`.

```
python -m ColoreProver input_theory -tptp -s='ReplacedSymbol':ReplacementSymbol
```

We use e.g.:

```
python -m ColoreProver codi/consistency/codi\_down\_nontrivial -tptp -s='<':less
```

5.3 Proving a lemma or a set of lemmas

We use the `-l` option, e.g.:

```
python -m ColoreProver dim/lemmas/inc_p_lemmas -l
```

It is assumed that all ontologies necessary are imported by the file `dim/lemmas/inc_p_lemmas`, i.e., the axioms directly included in `dim/lemmas/inc_p_lemmas` are the sentences to be proved, while all imports (including the import closure) are the axioms we are allowed to use to produce a proof. If `dim/lemmas/inc_p_lemmas` contains more than a single sentence, they will be broken into individual files and each will be proved separately.

5.4 Testing a set of heuristics for proving a lemma

We use, e.g.:

```
python -m ColoreProver dim/lemmas/inc_p_lemmas -l -t
```