TCLP Manual

Emmanuel COQUERY*

June 2, 2003

Contents

*Emmanuel.Coquery@inria.fr

1 What is TCLP ?

TCLP is a type checker for Prolog/CLP(X). TCLP thus aims to introduce a typing discipline to contraint logic programming. That is, given a description of the types' universe, and the set of the types of used function symbols, TCLP will check the coherence of the use of these function symbols. TCLP take the program plus any type definitions as input and will output the type of infered predicates plus any errors w.r.t. the type checking.

Currently TCLP comes with definitions for three Prolog/CLP(X) dialects: ISO-Prolog, GNU-Prolog (with CLP(FD) extension) and SICStus Prolog with its libraries (including the constraint programming libraries CLP(B), CLP(FD), CLP(QR), CHR).

TCLP has been written in SICStus Prolog and comes in three forms: an online demo[1] (Section 2), a SICStus Prolog library (Section 4) and a command line tool[2] (Section 3).

1.1 Features

- Uses three kinds of polymorphism:
  - parametric polymorphism (e.g. list(α))
  - subtyping (e.g. list(α) ¡ term)
  - overloading (i.e. ad hoc polymorphism, like `-` : $int \times int \rightarrow int$ and `-` : $\alpha \times \beta \rightarrow pair(\alpha, \beta)$
- Is able to infer a heuristic type for predicates
- Default type *term* or *atom* for undeclared function symbols
- Possibility to define your own clause syntax (ex `:+` instead of `:-` in CLP(FD) constraint definitions)

2 Online demo

2.1 Quick start

You can find the online demo at:
`http://contraintes.inria.fr/~coquery/tclp/exemples/demo.en.html`.

There you can try to type check some programs by writing them into the "Program to type check" area, ex:

[1] `http://contraintes.inria.fr/~coquery/tclp/exemples/demo.en.html`
[2] `http://contraintes.inria.fr/~coquery/tclp/download.en.html`

```
append([],L,L).
append([X|L],L2,[X|R]) :- append(L,L2,R).
```

By clicking on the *type inference* button, you will get the following result:

```
%% tclp 0.2.99g

%% starting type checking ...

:- typeof append(list(A),list(A),list(A)) is pred.

%% finished
```

You can also tell TCLP what Prolog/CLP dialect you are using with *CLP dialect* box.

2.2 Advanced usage

Here is a description of the different options you can use in the online demo:

- **Clear** clears the form.
- **CLP dialect** allows one to choose which Prolog/CLP dialect the program to type check is made for. You can choose either *ISO Prolog*, *GNU Prolog*, *SISCtus Prolog* or load your own type library with *Other (file)* and the *Browse ...* button on the right.
- **View built-in declarations** allows you to view the type definitions for the selected dialect.
- **Program to type check**: The source of the program to type check must be placed on this area. You can also use the *Browse ...* button to use one of your own file directly instead of copying it into the area.
- **Optionnal type declaration**: you can put any type declaration in this area. You can also load them using the *Browse ...* button.
- **Type inference** starts the type check the program. The type of unknown predicates defined in the program source will be infered
- **Type check only** starts the type checking of the program. Does not infer the type of unknown predicates.
- **Declared type of** prints the type of the given predicate or functor on the right. The syntax can be of the following forms:
  - `append`
  - `append/3`
  - `append/X`
  - `append(X,Y,Z)`
- **Declarations for module** prints the content of the *.typ* file for the given module. This is useful if you want to know available predicates and data structures of a peticular module.

3 Command line tool

3.1 Install

You can download the binaries at the follwing location:
`http://contraintes.inria.fr/~coquery/tclp/download.en.html`

3.1.1 Linux / MacOS X

Extract the archive in some folder, enter it and execute the install script:

```
tar xzvf tclp-i386-xxx.tgz

cd tclp-xxx

./install.sh
```

Install options:

–**prefix** `<dir>` the directory where TCLP files and directories will be installed (defaults to `/usr/local`)

–**bindir** `<dir>` the directory where the `tclp` executable will be installed, which should be in your `PATH` environment variable (defaults to *prefix*`/bin`)

–**tclpdir** `<dir>` the directory where TCLP files will be installed (defaults to *prefix*`/tclp`)

3.2 Get started

Basically, you just have to run on your file:
`tclp file.pl`
Example:

```
$ ../Devel/tclpchr/bin/tclp append.pl
%% tclp 0.3
%% tclp directory: /home/gewurz/coquery/Devel/tclpchr/bin
:- typeof append(list(A),list(A),list(A)) is pred.

$
```

3.3 Command line options

Usage:
`tclp [ --prolog <dialect> ] [ -I <dir> ] [ --stdlib <file> ] [ -v <sort> | -nv <sort> ] [ -i | --type_inference | --type_check ] [ -l | -nl ] [ --help ] <file> [ <files> ]`

`--prolog <dialect> <dialect>` is the name of a prolog dialect. This option causes TCLP to use type definitions for the corresponding dialect. Currently, available dialects are:

- `ISO`: ISO Prolog (default)
- `gprolog`: GNU Prolog
- `sicstus`: SICStus Prolog

`-I <dir>` This options adds `<dir>` to the list of directories where TCLP looks for files.

`--stdlib <file>` Makes TCLP use `<file>` as the standard type library (for built-in predicates). By default TCLP will use the first file named `stdlib.typ` in the lookup directories (which contains the dialect directory *`tclpdir`*`/lib/`*`dialect`*).

`-v <sort>` Enable verbose mode for `<sort>`, where `<sort>` can be either:

- `actions`: prints the different actions of TCLP , like, e.g., loading a type library.
- `timing`: prints CPU time used by TCLP for different actions.
- `defaults`: output a message each time TCLP encounters a term that has no defined type. This can be useful to understand some error messages.
- `types`: prints the type of each infered predicate (this is activated by default)

`-nv <sort>` Disables verbose mode for `<sort>`.

`-i, --type_inference` Enables predicate type inference (this is activated by default).

`--type_check` Disables predicate type inference.

`--help` Prints the different options.

4 SICStus Prolog library

4.1 Install and get started

You can download the library at the follwing location:
`http://contraintes.inria.fr/~coquery/tclp/download.en.html`

Just extract the archive where who want the library to be installed. To install the library as a SICStus Prolog library, extract it to a temporary folder and copy the files in the directory `tclp-xxx` to the SICStus Prolog library directory (e.g. `/usr/local/lib/sicstus-3.9.1/library`).

The library can be loaded with the `use_module` directive:

```
:- use_module('tclp-installation-directory/tclp')
```

or `:- use_module(library(tclp))` if the TCLP library is installed into the SICStus Prolog library directory.

Then you must choose the prolog dialect of the files to type check, using one of the following predicates:

- `iso/0`: the initializes TCLP for ISO dialect
- `gnu/0`: the initializes TCLP for GNU Prolog dialect
- `sicstus/0`: the initializes TCLP for SICStus Prolog dialect

You can type check files and load types using the `tclp/1` predicate:

```
:- tclp('file1.typ').

:- tclp(['file2.typ', 'file3.pl']).

...
```

4.2 Available predicates

These predicate are available in the TCLP library

4.2.1 Type initialization

`tclp__reinit` Remove all types from the current type data base and reload the types corresponding to the current dialect, using `tclp__reinit/1` and `user:init_dialect/1`.

`tclp__reinit(+Goal)` Remove all types from the current type data base, executes *Goal* and real the standard library according to the current settings (usually, *Goal* is used to determine these settings).

`tclp__set_dialect(+Dialect)` Sets the current CLP dialect to *Dialect*.

`user:init_dialect(+Dialect)` This is a dynamic predicate used by `tclp__reinit/0`.

`sicstus` Sets the dialect to SICStus Prolog and reinits the types.

`iso` Sets the dialect to ISO Prolog and reinits the types.

`gnu` Sets the dialect to GNU Prolog and reinits the types.

4.2.2 Processing

`tclp(+FileOrFiles)` *FileOrFiles* is either on file name or a list of file names. TCLP first loads '`.typ`' files of *FileOrFiles*, then reads and type checks '`.pl`' files of *FileOrFiles*.

`tclp__process_typ(+FileOrFiles)` *FileOrFiles* is a file or a list files. TCLP loads TCLP declarations in these files.

`tclp__process_pl(+FileOrFiles)` *FileOrFiles* is a file or a list files. TCLP reads Prolog programs in these files and type check them.

`tclp__process_phrases(+PhraseList)` *PhraseList* is a list of pairs *Phrase-Location*, where *Phrase* is a phrase to type check and *Location* is some information about the phrase, given by `tclp__reader:read_one_term(`*Stream*, *Phrase*, *Location*, *FileName*`)`.

4.2.3 Options

`tclp__set_inference(+TrueFalse)` *TrueFalse* is either `true` or `false`. This predicate en(dis)ables type predicate inference in TCLP.

`tclp__add_search_directory(+Directory)` *Directory* is a directory name. This predicate adds *Directory* to the directories used by TCLP for file lookup.

`tclp__set_stdlib_name(+File)` Sets the name of the file to load as the standard type library to *File*.

`tclp__enable_verbose(+Sort)` Enables verbose mode for *Sort*, where *Sort* can be one of:

- `actions`: prints the different actions of TCLP , like, e.g., loading a type library.
- `timing`: prints CPU time used by TCLP for different actions.
- `defaults`: output a message each time TCLP encounters a term that has no defined type. This can be useful to understand some error messages.
- `types`: prints the type of each infered predicate (this is activated by default)

`tclp__disable_verbose(+Sort)` Disables verbose mode for *Sort*, where *Sort* can take the values above.

4.2.4 The tclp() path alias

The loading of the `tclp` module will cause a path alias '`tclp`' to be created. This alias can used, either to load a module of the TCLP implementation (like `tclp(tclp__reader)`) or to access the type files for ISO (resp. GNU and SICStus) Prolog using `tclp('lib/ISO/file.typ')` (resp. `tclp('lib/gprolog/file.typ')` and `tclp('lib/sicstus/file.typ')`.

5 Syntax of TCLP declarations

5.1 Syntax of types

type	**::=**	**symbol(type , ... , type)**
	|	**symbol**
	|	**variable**
symbol	:	a prolog atom, which corresponds to a type constructor
variable	:	a prolog variable, which corresponds to a type parameter

5.2 Type declaration

`:- order` *`Type1(A1, ... , AM) < Type2(B1, ... , BN)`* Declares that the type constructor *`Type1`* is smaller than the type constructor *`Type2`*. The mapping are given by the arguments, i.e. if *`AI == BJ`* then the I-th argument of *`Type1`* corresponds to the J-th argument of *`Type2`*.

`:- typeof` *`+TypedTerm`* `is` *`+Type`* Declares a type for the given term. *`TypedTerm`* is of the form *`Name`*`(`*`Type1`*`,...,`*`TypeN`*`)`, where *`Type`* and *`TypeI`* are types. *Type1*,...,*TypeN* are the types of the arguments and *`Type`* is the type of the result.

`:- type` *`+TypeConstructor`*`.` Declares a type. *`TypeConstructor`* is either of the form *`Name/Arity`* or of the form *`Name`*`(A,B,...)`. It declares the existence of a type named *`Name`* which have *`Arity`* arguments.

`:- type` *`+Type`* `is` *`+TermConstructorList`*`.` Combinaison of type and typeof. *`Type`* is of the form *`Name`*`(A,B,...)` and *+TermConstructorList* is alist of terms of the form *`Name`*`(`*`Type1`*`,...,`*`TypeN`*`)`. The declared type is *`Type`*. The result type is *`Type`*.

`:- untyped` *`+SpecOrSpecList`*`.` Avoid to type check the given terms. *`SpecOrSpecList`* is a functor specification of the form *`Name/Arity`*, or a list of such specifications. It causes TCLP to avoid to type check these terms.

5.3 TCLP meta declarations

`:- tclp__include(`*`+File`*`)` Include TCLP declarations found in *`File`*.

`:- user:args_location(`*`+Location,?LocationList`*`)` This predicate decompose the location of a term into the list of the locations of its subterms.

`:- tclp__define_clause(`*`+Phrase,+Location,+Heads,+Bodies,+Condition`*`)` This will tell TCLP what the clauses look like. *`Phrase`* is the phrase to cut into bodies and heads. *`Location`* is the location of the phrase in the program source. *`Heads`* is a list of triplets *`Head-HLocation-Type`*, where *`Head`* is a head of the clause, *`HLocation`* is the location of *`Head`* in the program source (one can use `user:args_location/2` to find it) and *`Type`* is the type expected for the head of the clause. *`Bodies`* is a list of pairs

Body-BLocation, where *Body* is a body of the clause and *BLocation* is its location. *Condition* is a goal executed when recognizing clauses. E.g. (prolog directive "`:- Body`"):

```
:- tclp__define_clause((:- Body), Location, [], [Body-BodyLocation],
user:args_location(Location,[BodyLocation]))
```

`:- tclp__define_clause(+Phrase,+Location,+Heads,+Bodies)` Same as `tclp__define_clause(Phrase,Location,Heads,Bodies,true)`

`:- tclp__define_clause_op(+BinOp,+Type)` This predicate is a shortcut of `tclp__define_clause/5` for defining binary clause operators such as `':-'/2`. *BinOp* is the name of the operator and *Type* is the type expected for the head of the clause. E.g.

```
:- tclp__define_clause_op(':-',pred).
```

`:- tclp__define_clause_op(+BinOp)` Same as `tclp__define_clause_op(BinOp,pred)`

`:- tclp__executable(+Goal,+Condition)` Tells TCLP that the goal *Goal* is a TCLP declaration if the goal *Condition* succeeds. E.g. the `op/3` directive:

```
:- tclp__executable(op(_,_,_),true).
```

`:- tclp__executable(+Goal)` Same as `tclp__executable(+Goal,true)`

`:- tclp__add_hook(+Goal,+Location,+Hook,+Condition)` Whenever TCLP encounters a TCLP declaration, it will handle it using a hook defined via this predicate. *Goal* is the TCLP declaration, *Location* is its location in the program source, *Hook* is the goal that will be executed by TCLP to handle the declaration and *Condition* is a goal that must succeed. E.g. the `op/3` directive:

```
:- tclp__add_hook(op(Priority,Mode,Operators), _, op(Priority,Mode,Operators),
true).
```

`:- tclp__add_hook(+Goal,+Location,+Hook)` Same as `tclp__add_hook(Goal,Location,Hook,true)`

`:- tclp__add_hook(+Goal,+Hook)` Same as `tclp__add_hook(Goal,_,Hook)`

5.4 Dialect specific declarations

`:- tclp__load_prolog(+File)` Causes TCLP to consult *File* is in `consult/1`. This can be useful if you want to define complex treatments of some TCLP declarations.

6 Limitations

modules Currently, TCLP only loads `.typ` files automatly when it encounters the directives `use_module/[123]` and `module/2`. The functor `':'/2` is not handled. Moreover if two predicates with the same name and arity exists in two different modules, they will be considered as the same predicate by TCLP.

A TCLP source files

`tclp.pl` This the main file for the library. It mainly consists in wrappers for calling predicates in other TCLP modules.

`tclp__main.pl` The main file for the command line tool.

`tclp/tclp__arity.pl` The file contains predicates for handling depandancies between the arguments of the different type constructors.

`tclp/tclp__arrays.pl` Defines dynamic arrays with predicates to manipulate them.

`tclp/tclp__clp_types.pl` Defines the solver for constraints over subtyping inequalities.

`tclp/tclp__connexity.pl` Handles the computation of mutually recursive predicates in the program source, as well as the order in which the clauses must be type checked.

`tclp/tclp__declarations.pl` Defines the core meta declarations of TCLP.

`tclp/tclp__def.pl` Some constants definitions.

`tclp/tclp__errors.pl` Handles printing of errors and warning. Also handles the verbose predicates (should be in an IO module I think)

`tclp/tclp__files.pl` Handles file loading, file names, etc ...

`tclp/tclp__functor_types.pl` Handles the typing of the application of a functor to its arguments. In particular, it handles the treatment of overloading.

`tclp/tclp__handle_goals.pl` Core handling TCLP definitions.

`tclp/tclp__reader.pl` Reads source phrases and handle their location in the program source.

`tclp/tclp__type_checker.pl` Type checking predicates.

`tclp/tclp__type_inference.pl` Defines how the inference of the type of predicates.

`tclp/tclp__type_order.pl` Handles the order between type constructors.

`tclp/tclp__utils.pl` Miscellaneous predicates.

`tclp/version.pl` The version TCLP.

`tclp/sicstus_modules.pl` Predicates to handle some SICStus Prolog directives (e.g. `use_module/1`).

B TCLP `.typ` files

ISO Prolog

`lib/ISO/corelib.typ` Meta definitions for ISO Prolog. Included from `lib/ISO/stdlib.typ`.

`lib/ISO/stdlib.typ` Standard type library for ISO Prolog.

GNU Prolog

`lib/gprolog/corelib.typ` Meta definitions for GNU Prolog. Included from `lib/gprolog/stdlib.typ`

`lib/gprolog/stdlib.typ` Standard type library for GNU Prolog.

SICStus Prolog

`lib/sicstus/corelib.typ` Meta definitions for SICStus Prolog. Included from `lib/sicstus/stdlib.typ`

`lib/sicstus/stdlib.typ` Standard type library for SICStus Prolog.

`lib/sicstus/prolog.typ` Internal type library for SICStus Prolog. You need this one for predicates used with the `prolog:predicate(...)` prefix.

`lib/sicstus/arrays.typ` types for SICStus module arrays

`lib/sicstus/assoc.typ` types for SICStus module assoc

`lib/sicstus/attributes.typ` types for SICStus module atts

`lib/sicstus/bdb.typ` types for SICStus module bdb

`lib/sicstus/charsio.typ` types for SICStus module charsio

`lib/sicstus/chr.typ` types for SICStus module chr implementing the CHR extension. It contains meta definitions to handle the solver definitions (e.g. `'<=>'/2`).

`lib/sicstus/clpb.typ` types for SICStus module clpb implementing the CLP(B) extension

`lib/sicstus/clpfd.typ` types for SICStus module clpfd implementing the CLP(FD) extension. It contains meta dclarations to handle contraints definitions (e.g. `':+'/2`).

`lib/sicstus/clpq.typ`**,** `lib/sicstus/clpr.typ` **and** `lib/sicstus/clpqr.typ`
types for SICStus modules clpr, clpq and clpqr implementing extension CLP(Q) and CLP(R).

`lib/sicstus/fastrw.typ` types for SICStus module fastrw

`lib/sicstus/heaps.typ` types for SICStus module heaps

`lib/sicstus/jasper.typ` types for SICStus module jasper

`lib/sicstus/linda.typ` types for SICStus module linda

`lib/sicstus/lists.typ` types for SICStus module lists

`lib/sicstus/ordsets.typ` types for SICStus module ordsets

`lib/sicstus/queues.typ` types for SICStus module queues

`lib/sicstus/random.typ` types for SICStus module random

`lib/sicstus/sockets.typ` types for SICStus module sockets

`lib/sicstus/system.typ` types for SICStus module system

`lib/sicstus/terms.typ` types for SICStus module terms

`lib/sicstus/timeout.typ` types for SICStus module timeout

`lib/sicstus/trees.typ` types for SICStus module trees

`lib/sicstus/ugraphs.typ` types for SICStus module ugraphs

`lib/sicstus/user.typ` types for SICStus module user

`lib/sicstus/wgraphs.typ` types for SICStus module wgraphs