UTF-8

An example of documentation and tests

Isabel Garcia, C347963
Manuel Hermenegildo, D165125

Table of Contents

ejemplo

This module defines a calculator for Peano numbers.

The numbers accepted by this calculator have to be of the form:

```
nat(0).
nat(s(X)) :-
    nat(X).
```

Examples of use:

1. Adding two numbers:

```
?- calculate('+', 0, s(0), X).

X = s(0) ?
yes
?-
```

2. Subtracting two numbers:

```
?- calculate('-', s(s(0)), s(0), X).

X = s(0) ?
yes
?-
```

The available operations are:

```
operation(+).
operation(-).
```

Generating the documentation

This documentation has been generated automatically by the **lpdoc** (`http://ciao-lang.org/ciao/build/doc/lpdoc.html/`) tool.

- To generate it, after opening the file inside Emacs, select the following menu options:
  `LPdoc -> Generate documentation for buffer`
  (or type ⟨C-c⟩ ⟨D⟩ ⟨B⟩). You can also type
  `lpdoc -t html calculator.pl`
  or
  `lpdoc -t pdf calculator.pl`
  at the command line.
- For visualizing the output, select:
  `LPdoc -> View documentation in selected format`
  (or type ⟨C-c⟩ ⟨D⟩ ⟨V⟩), or
  `lpdoc --view calculator.pl`
  at the command line.

You can select different formats such as `pdf` or `html` at generation time. For generating `pdf` directly from `lpdoc` you need to have a TeX/LaTeX distribution such as TeX Live, etc. installed (depending on the distribution you may need to install `texlive`, `texinfo`, and `imagemagick`).

Alternatively, you can also generate the `html`, open it in a browser, and then save it from the browser to `pdf` (e.g., by *printing* it to a `pdf` file).

This module includes some formatting commands but there are many more in: `http://ciao-lang.org/ciao/build/doc/lpdoc.html/comments.html#stringcommand/1`.

To document each predicate, specific assertions are used. For example:

```
:- pred calculate(Op,A,B,C)
   # "@var{C} is the result from applying operation
      @var{Op} to @var{A} and @var{B}.".
```

With assertions you can also specify and check types and many other properties.

For more information consult:

`http://ciao-lang.org/ciao/build/doc/lpdoc.html/assertions_doc.html`.

A question often asked is how to do accents: in text that is within strings use `@'{...}` and `@~{...}` for eñe. E.g.:

```
:- doc(title, "Programaci@'{o}n L@'{o}gica").
```

In `alumno_prode/4` facts you can use accents, but please remeber to always put the arguments in single quotes, `'...'`, to make sure they are constants:

```
alumno_prode('lvaro', 'Fernndez', 'Gmez', 'Y16M025').
```

Automatic tests

This module also includes some assertions that start with `:- test`. For example:

```
:- test calculate(Op,A,B,C)
      :  (Op = '+', A = 0, B = 0)
      => (C = 0) + not_fails # "Base case.".
```

These assertions define test cases. Given an assertion:

`:- test` *Head* `:` *Call* `=>` *Exit* `+` *Comp*.

`Head` denotes the predicate to which the assertion applies, `Call` describes the values to call the predicate with for the test, `Exit` defines the expected values upon exit **if the predicate succeeds** and `Comp` will be used to define global properties, for example if the predicate should fail or succeed for that call:

- `not_fails`: means that the call to the predicate with the values in `Call` will generate at least one solution.
- `fails`: means that the call to the predicate with the values in `Call` will fail.

Including tests in the documentation

You can have `lpdoc` include the tests in the documentation. For this, include option `--doc_mainopts=tests` in the `lpdoc` command. E.g., at the command line:

```
lpdoc -t html --doc_mainopts=tests calculator.pl
```

Or, if you are generating the manual from Emacs, switch to the `*LPdoc*` buffer (you can use the 'Buffers' menu) and issue the command:

```
lpdoc ?- doc_cmd('SETTINGS.pl',[name_value(doc_mainopts,tests)],gen(html)).
```

Launching the tests automatically

To run the tests, open the `.pl` file inside Emacs and select the following menu options: `CiaoDbg -> Run tests in current module` (or type ⟨C-c⟩ ⟨u⟩).

Note that when these tests are run the system by default tries to also find a second solution for each test (i.e., like typing ⟨;⟩ in the top level).

Usage and interface

- **Library usage:**
  `:- use_module(/home/nicocossio/UPM/Prolog/ejemplo.pl).`
- **Exports:**
  - *Predicates:*
    `calculate/4`, `sum/3`.
  - *Properties:*
    `operation/1`, `nat/1`.

Documentation on exports

operation/1: PROPERTY

Usage: `operation(Op)`

`Op` is an operation accepted by the calculator.

```
operation(+).
operation(-).
```

calculate/4: PREDICATE

Usage: `calculate(Op,A,B,C)`

`C` is the result of applying operation `Op` to `A` and `B`.

```
calculate(+,A,B,C) :-
    sum(A,B,C).
calculate(-,A,B,C) :-
    sum(B,C,A).
```

sum/3: PREDICATE

Usage: `sum(A,B,C)`

`C` is the sum of `A` and `B` in Peano format.

```
sum(0,X,X) :-
    nat(X).
sum(s(X),Y,s(Z)) :-
    sum(X,Y,Z).
```

nat/1: PROPERTY

Usage:

Natural number.

```
nat(0).
nat(s(X)) :-
    nat(X).
```

Documentation on imports

This module has the following direct dependencies:

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

References

(this section is empty)