Lab 0: A first taste of Why3

15-414: Automated program verification

Lab goals

This first Why3 lab is not designed to be challenging and it obeys two main goals:

- Making sure that you've successfully installed the Why3 platform on your machine.
- Introducing the Why3 syntax and providing you examples that will help you getting started for the next labs.

Therefore, we encourage you to be curious and try to understand the examples we provide, even if you won't be evaluated on this. These examples will be further discussed during an interactive live coding session.

Installing Why3

We provide here some instructions for installing Why3 along with Alt-Ergo and Z3 (the two automated provers we will use for this class) on Ubuntu Linux and MacOs. It should not be too hard to adapt these instructions for other Linux distributions. If you happen to use Windows however, we recommend that you use the Ubuntu virtual machine that we provide on the class website (`http://www.cs.cmu.edu/~15414/labs/15-414-VM.zip`).

Instructions for MacOs

You will need the Homebrew package manager. Then:

1. Install `opam`, the OCaml package manager and libraries for why3:
   ```
   brew install hg darcs opam gtk+ gmp gtksourceview libgnomecanvas z3 autoconf
   ```

2. Initialize opam: `opam init ; opam config setup -a ; eval $(opam config env)`. Answer *yes* if queried.

3. `opam install lablgtk zarith why3 alt-ergo`

4. Finally, setup why3 by running `why3 config --detect-provers`. The two provers Alt-Ergo and Z3 should be detected on your system (at least).

Instructions for Ubuntu Linux

1. Install `opam`, the OCaml package manager: `sudo apt-get install opam`

2. Initialize opam: `opam init ; opam config setup -a ; eval $(opam config env)`. Answer *yes* if queried.

3. `sudo apt-get install libgmp-dev libgtksourceview2.0-dev m4`

4. `opam install why3 alt-ergo`

5. `sudo apt-get install z3`

6. Finally, setup why3 by running `why3 config --detect-provers`. The two provers Alt-Ergo and Z3 should be detected on your system (at least).

Setting up the virtual machine

1. Download the VM at `http://www.cs.cmu.edu/~15414/labs/15-414-VM.ova`.

2. Install the latest version of Oracle VirtualBox.

3. Launch VirtualBox, go to "*Files/Import Appliance*" and select the file above.

4. Start the virtual machine, and then you're done: everything you need is already pre-installed.

Installing a text editor for WhyML

There are WhyML plugins for Emacs and Atom. If you have no experience with Emacs, we recommend that you use the atom plugin.

- To install the Atom plugin, start Atom (`https://atom.io/`) and go to `Preferences/Install`. Search for `language-whyml`.
- To install the Emacs plugin, download `why3.el` at

  `https://gist.github.com/sagotch/46a92bcc41abfecf4d79`

  and put in in your `~/.emacs.d/lisp/` folder. Then, add the following lines to your `.emacs` file:

  ```
  (add-to-list 'load-path "~/.emacs.d/lisp/")
  (load "why3.el")
  ```

Lab instructions

After installing Why3 using the instructions above, download the file `lab0.mlw` on the course's website and start the Why3 IDE using the following command:

```
why3 ide lab0.mlw
```

Then, select the top node in the hierarchy (`lab0.mlw`) and click on the `Alt-Ergo` button to run proof attempts for every example in the file. Some goals should become green and a folder `lab0` should be created, that contains the current *proof session*. Make sure to always save the current proof session when you exit the Why3 IDE.

Unfortunately, the current version of the Why3 IDE does not allow WhyML files to be edited within it. Therefore, in order to modify `lab0.mlw`, you will have to open it using an external editor. After you've changed it, you can reload it in the Why3 IDE by pressing Ctrl+R.

Example 1 : lists and recursive functions

If you followed the instructions above, you should see a green bullet next to the line "`VC for remove`" in the IDE, indicating that the function `remove` has been proved correct successfully. Look at it carefully and answer the following question.

Question: What aspects of the `remove` function are not constrained by its specification ? More specifically, can you think of an alternative implementation for it that would still match the specification but might fail to match user's expectations of how it should behave ? Give such a faulty implementation and informally describe an extension of the specification that would rule it out.

Example 2 : arrays and imperative programming

A bug is hidden in the `find_first_pos_idx` function, which certainly explains why our provers did not manage to prove it correct.

Task: Fix this bug and verify the patched implementation (by clicking on the *Alt-Ergo* or *Z3* button again).

Example 3 : specifying code that modifies mutable structures

We did not finish to write the specification of the `replace` function.

Task: Fill the holes in the specification of `replace` by replacing the two instances of

```
false (* TODO *)
```

by something smarter.

What to hand back

First make sure that every goal is handled successfully by the provers in your completed version of `lab0.mlw`. Then, send us a zip archive with the following content:

1. The completed `lab0.mlw` file
2. The session folder generated by Why3 IDE
3. An ASCII text file with extension `.txt` containing some comments you may want to share with us along with the answer of the question we asked in Example 1.

In order to make sure your archive is valid, you can uncompress it and run the following two commands (we will do the same):

```
why3 replay lab-XXX  # should print that everything replayed OK
why3 session info --stats lab-XXX  # prints the list of unproved goals
```