Getting Started with Isabelle

Jørgen Villadsen

4 September 2014

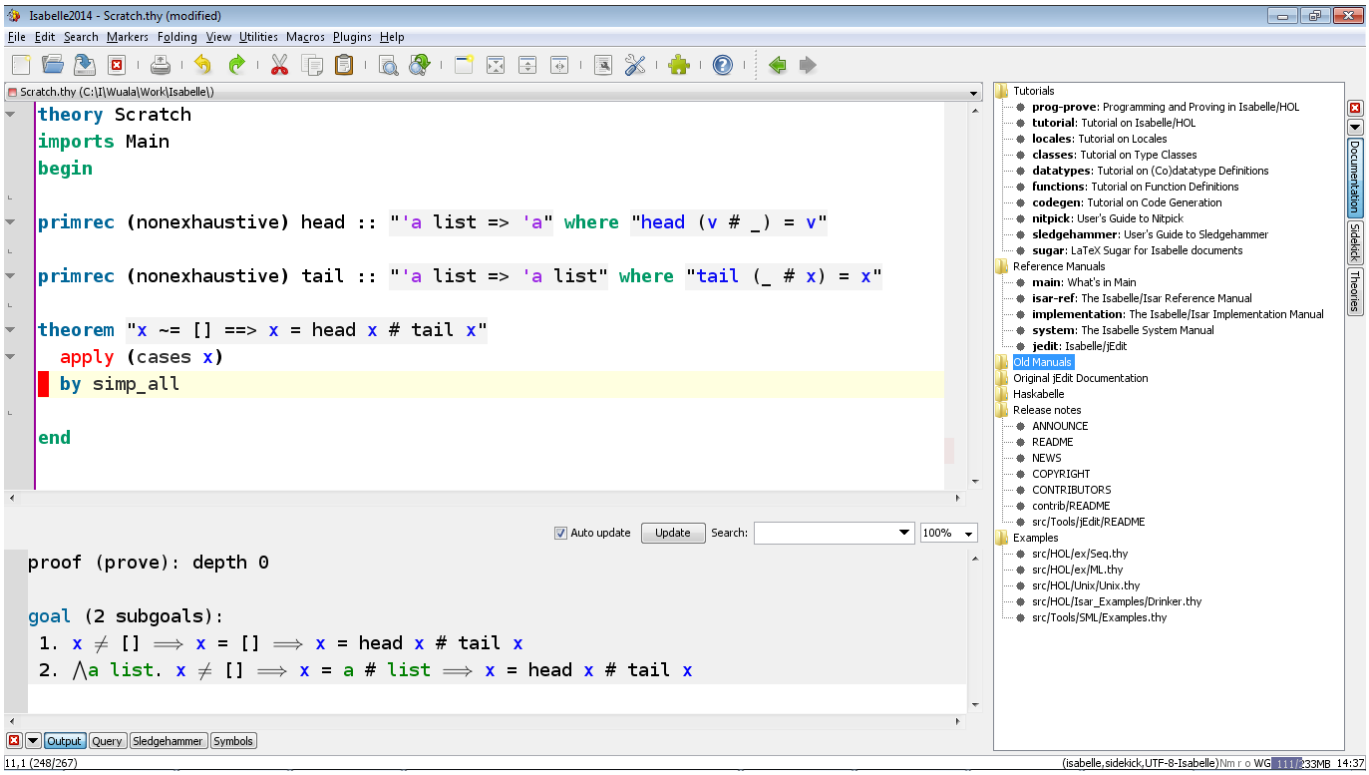
The Isabelle Prover IDE (Integrated Development Environment) allows the user to edit a source file and lets the proof assistant Isabelle run asynchronously in the background processing the file.

Follow the installation instructions here:

http://isabelle.in.tum.de/

Wait for Isabelle to start — it can take a while...

After selecting *Output* at the bottom you should have a window like this one:



The bottom part of the window shows the output buffer with messages from the current command. The current command is determined by the cursor position in the main buffer in the top part of the window. In the following we discuss the text shown in the main buffer.

We want to formalize a simple fact about lists. Before we can do so, we need to create a *theory* which is Isabelle parlance for a source file that bundles definitions and facts under a common name.

Start by typing "`theory Test imports Main begin`" into the main buffer. Isabelle will create a theory with name *Test* on top of the theory named *Main* which provides a kind of "standard library" of definitions and facts.

At this point you may notice an error message in the output buffer:

```
Bad file name "Scratch.thy" for theory "Test"
```

This just happened because the default file name for a theory is `Scratch.thy` and the problem can easily be solved by saving using menu item File→Save the current file under the name `Test.thy`. A theory with name *Test* must be in a file with name `Test.thy`. The Isabelle convention is to capitalize theory names and separate words by underscores, hence `Complete_Partial_Order` rather than `completePartialOrder`, `complete_partial_order`, etc.

Now, let us come back to the announced fact about lists: when we combine the head and the tail of a non-empty list, we obtain the same list again. To make it a bit more interesting, we define our own functions for computing the head and the tail of a list as follows (when typing `=>` it is replaced by a graphical symbol and similarly in some other cases):

```
fun head :: "'a list => 'a" where "head (v # x) = v"
fun tail :: "'a list => 'a list" where "tail (v # x) = x"
```

When defining `head`, a warning occurs:

```
Missing patterns in function definition:
head [] = undefined
```

This tells us that our function is not defined on input `[]` and it is not an error but just a warning, since we could intend our function to be undefined in this case (as we indeed do). A similar warning is issued for `tail`.

Now we prove our little theorem:

```
theorem "x ~= [] ==> head x # tail x = x" by (cases x) auto
```

We finish our small theory by typing "`end`" at the end of the file.

Here is an alternative version of the same example which avoids the warnings and can be inserted directly into the `Scratch.thy` file:

```
theory Scratch imports Main
begin
primrec (nonexhaustive) head :: "'a list => 'a" where "head (v # _) = v"
primrec (nonexhaustive) tail :: "'a list => 'a list" where "tail (_ # x) = x"
theorem "x ~= [] ==> x = head x # tail x" apply (cases x) by simp_all
end
```