Isabelle/HOL basics

This is only a short memo for Isabelle/HOL. For a more detailed documentation, please refer to
`http://isabelle.in.tum.de/website-Isabelle2014/documentation.html`

1 Survival kit

1.1 ASCII Symbols used in Logic Formulas

Symbol	ASCII
True	True
False	False
$\wedge$	/\

Symbol	ASCII
$\vee$	\/
$\neg$	~
$\neq$	~=

Symbol	ASCII
$\longrightarrow$	-->
$\leftrightarrow$	=
$\forall$	ALL

Symbol	ASCII
$\exists$	?
λ	%
$\Rightarrow$	=>

1.2 Lemma declaration and visualization

- declare a lemma (resp. theorem) ..... `lemma` (resp. `theorem`)

```
lemma "A --> (B \/ A)"
lemma deMorgan: "~(A /\ B)=(~A \/ ~B)"
```

- to visualize the lemma/theorem/simplification rule associated to a given name..... `thm`

```
thm "deMorgan"
thm "append.simps"
```

- to find and visualize all the lemmas/theorems/simplification rules defined using given symbols `find_theorems`

```
find_theorems "append" "_ + _"
```

1.3 Basic Proof Commands

- search for a counterexample for the first subgoal using SAT-solving ..... `nitpick`
- search for a counterexample for the first subgoal using automatic testing ..... `quickcheck`
- automatically solve or simplify all subgoals ..... `apply auto`
- close the proof of a proven lemma or theorem ..... `done`

```
lemma "A --> (B \/ A)"
apply auto
done
```

- abandon the proof of an unprovable lemma or theorem ..... `oops`

```
lemma "A /\ B"
nitpick
oops
```

- abandon the proof of a (potentially) provable lemma or theorem ..... `sorry`

1.4 Evaluation

- evaluate a term ..... `value`

```
value "(1::nat) + 2"        value "[x,y] @ [z,u]"        value "(%x y. y) 1 2"
```

1.5 Basic Definition Commands

- associate a name to a value (or a function) ..... `definition`

```
definition "l1=[1,2]"        definition "l2= l1@l1"        definition "f= (%x y. y)"
```

- define a function using equations ..... `fun`

```
fun count:: "'a => 'a list => nat"
where
"count _    []  = 0" |
"count e (x#xs) = (if e=x then (1+(count e xs)) else (count e xs))"
```

- define an Abstract Data Type ..... `datatype`

```
datatype 'a list = Nil | Cons 'a  "'a list"
```

1.6 Code exportation

- export code (in `Scala, Haskell, OCaml, SML`) for a list of functions ..... `export_code`

```
export_code function1 function2 function3 in Scala
  file "myfile.scala"
```

2 To go further... and faster

- apply structural induction on a variable `x` of an inductive type ..... `apply (induct x)`
- apply an induction principle adapted to the function call `(f x y z)` . `apply (induct x y z rule:f.induct)`
- automatically solve or simplify the first subgoal ..... `apply simp`
- insert an already defined lemma `lem` in the current subgoal ..... `apply (insert lem)`
- do a proof by cases on a variable `x` or on a formula `F` ..... `apply (case_tac "x")` or `apply (case_tac "F")`
- try to prove the first subgoal with Sledgehammer ..... Plugins>Isabelle>Sledgehammer
- set the goal number i as the first goal ..... `prefer i`
- options of `nitpick`
  - `timeout=t`, nitpick searches for a counterexample during at most `t` seconds. (`timeout=none` is also possible)
  - `show_all`, nitpick displays the chosen domains and interpretations for the counterexample to hold.
  - `expect=s`, specifies the expected outcome of the nitpick call, where `s` can be `none` (no found counterexample) or `genuine` (a counterexample has been found).
  - `card=i-j`, specifies the cardinalities to use for building the SAT problem.
  - `eval=l`, gives a list `l` of terms to eval with the values found for the counterexample.

    ```
    nitpick [timeout=120, card=3-5, eval= "member e l" "length l"]
    ```

- options for `quickcheck`
  - `timeout=t`, quickcheck searches for a counterexample during at most `t` seconds.
  - `tester=tool`, specifies the type of testing to perform, where `tool` can be `random`, `exhaustive` or `narrowing`.
  - `size=i`, specifies the maximal size of the search space of testing values.
  - `expect=s`, specifies the expected outcome of quickcheck, where `s` can be `no_counterexample` (no found counterexample), `counterexample` (a counterexample has been found) or `no_expectation` (we don't know).
  - `eval=l`, gives a list `l` of terms to eval with the values found for the counterexample. Not supported for `narrowing` and `random` testers.

    ```
    quickcheck [tester=narrowing, eval=["member e l","length l"]]
    ```

- setting option values for all calls to `nitpick` ..... `nitpick_params`

```
nitpick_params [timeout=120, expect=none]
```

- setting option values for all calls to `quickcheck` ..... `quickcheck_params`

```
quickcheck_params [tester=narrowing, timeout=500]
```