

PVS & Emacs interaction

- to typecheck a file: CTRL-c CTRL-t
or PVS → Parsing and Typechecking → Typecheck
- to prove a lemma: CTRL-c CTRL-p x
or PVS → Prover Invocation → x prove
- CTRL-c CTRL-c interrupts PVS; continue with the command `restore`)
- ALT-P cycles through previous commands
- CTRL-] kills request for input on bottom line
- CTRL-x 1 unsplit window, ie. kill all windows except the one where cursor is
- CTRL-x 2 splits window in two
- `*pvs*` in the "Buffers" menu is the PVS prover

Most common tactics and their abbreviation

fnum: a number of antecedent of consequent

expr: a arbitrary expression

name: a name of a definition or lemma

[] denotes an optional parameter

control

(undo)	TAB u
(undo undo) ie. redo	
(postpone)	TAB P
(help <i>command</i>) eg (help replace)	
(quit)	

propositional logic

(flatten [<i>fnum</i>])	TAB f
(split [<i>fnum</i>])	TAB s
(case " <i>expr</i> ") eg (case "n!1 > 0")	TAB c
(lift-if [<i>fnum</i>])	TAB l
(prop)	TAB p

General tip: always try to (flatten).

predicate logic

(skolem <i>fnum</i> " <i>name</i> " ..." <i>name</i> ")	
Eg (skolem -1 "x")	
(skolem! [<i>fnum</i>])	TAB !
(skosimp [<i>fnum</i>])	TAB S
(skosimp*)	TAB *
(inst <i>fnum</i> " <i>expr</i> " ..." <i>expr</i> ")	
Eg (inst -1 "2+j!1")	
(inst? [<i>fnum</i>])	TAB ?

General tip: (nearly) always try to (skosimp*), unless you're faced with a universal quantification that you want to prove by induction.

equality

(expand " <i>name</i> " [<i>fnum</i>] [<i>occurrence</i>])	TAB e
TAB e with cursor on formula to expand	
(replace <i>fnum</i> [<i>fnum</i>] RL)	TAB r
(replace <i>fnum</i> [<i>fnum</i>] LR)	TAB r
(rewrite)	TAB R
(assert)	TAB a

induction

(induct " <i>n</i> ")	TAB I
(induct-and-simplify " <i>n</i> ")	TAB CTRL-s

..., n:nat, ...) ...

Goal should be of the form FORALL(

automated proving

(prop)	TAB p
(assert)	TAB a
(grind)	TAB G

misc

(lemma " <i>name</i> ")	TAB l
(apply-extensionality [<i>fnum</i>])	
(decompose-equality [<i>fnum</i>])	TAB =