LEAN TACTICS: ONE PAGE CHEAT SHEET

MARK WILDON

With examples at: `http://www.ma.rhul.ac.uk/~uvah099/Maths/TacticsCheatSheet.lean`

Tactics dealing with construction and deconstruction. Where multiple near equivalents are shown, the first in bold may be most convenient.

Value : Type or **Type**	in **Hypothesis** 'destructor'	in **Goal** 'constructor'
$h : x = y$ **See also:** rw		**rfl**
$h : \exists k, p\ k$	**obtain** $\langle k, h_k\rangle := h$ rcases h with $\langle k, h_k\rangle$ cases h with $\mid$ intro $k\ h_k$	**use** k exists k refine $\langle k, ?\_\rangle$ GOAL NOW $p\ k$ exact Exists.intro $k\ h_k$ NEED $h_k : p\ k$ IN SCOPE
$h : p \to q$ **See also:** rw	**have** $h_q : q := h\ h_p$ NEED $h_p : p$ IN SCOPE 'function application'	**intro** h_p GOAL NOW q, HYPOTHESIS $h_p : p$ 'λ abstraction'
$h : p \leftrightarrow q$	**obtain** $\langle h_{\text{mp}}, h_{\text{mpr}}\rangle := h$ rcases h with $\langle h_{\text{mp}}, h_{\text{mpr}}\rangle$ cases h with $\mid$ intro $h_{\text{mp}}\ h_{\text{mpr}} \implies$ h.mp : $p \to q$, h.mpr : $q \to p$	**constructor** $\cdot\ h_{\text{mp}} : p \to q$ $\cdot\ h_{\text{mpr}} : q \to p$ refine $\langle ?\_, ?\_\rangle$ ... THEN AS constructor exact $\langle h_{\text{mp}}, h_{\text{mpr}}\rangle$ exact Iff.intro $h_{\text{mp}}\ h_{\text{mpr}}$ NEED $h_{\text{mp}} : p \to q$, $h_{\text{mpr}} : q \to p$ IN SCOPE
$h : \forall k, p\ k$	**specialize** $h\ \ell$ have $h_\ell := h\ \ell$ let $h_\ell := h\ \ell$ 'function application'	**intro** ℓ ~~by~~ fun $\ell \implies$ by ... GOAL NOW $p\ l$ 'λ abstraction'
$h : \forall k, \exists \ell, p\ k\ \ell$	**choose** $f\ h'$ **using** h NOW $h' : \forall k, p\ k\ (f\ k)$ 'skolemisation'	
$h : p \wedge q$	**obtain** $\langle h_p, h_q\rangle := h$ rcases h with $\langle h_p, h_q\rangle$ cases h with $\mid$ intro $h_p\ h_q \implies$ h.left : p, h.right : q	**constructor** $\cdot\ h_p : p$ $\cdot\ h_q : q$ refine $\langle ?\_, ?\_\rangle$ $\cdot\ h_p : p$ $\cdot\ h_q : q$ GOALS NOW $h_p : p$ AND $h_q : q$ exact $\langle h_p, h_q\rangle$ exact And.intro $h_p\ h_q$ PROVIDED $h_p : p$, $h_q : q$ IN SCOPE
$p \wedge q \to$	**rintro** $\langle h_p, h_q\rangle$	
$h : p \vee q$	**rcases** h **with** $(h_p \mid h_q)$ $\cdot\ h_p : p$ NOW IN SCOPE $\cdot\ h_q : q$ NOW IN SCOPE obtain $(h_p \mid h_q) := h$ $\cdot\ h_p : p$ NOW IN SCOPE $\cdot\ h_q : q$ NOW IN SCOPE cases h with inl $h_p \implies$ inr $h_q \implies$	**exact Or.inl** $(h_p : p)$ **exact Or.inr** $(h_q : q)$
$p \vee q \to$	**rintro** $(h_p \mid h_q)$ $\cdot\ h_p : p$ NOW IN SCOPE $\cdot\ h_q : q$ NOW IN SCOPE	
$h : \begin{matrix}\neg\forall, p\ x \\ \neg\exists, q\ x\end{matrix}$	**push_neg at** h	**push_neg**

Remarks.

- Compare destructors (for hypothesis) and constructors (for goal) in Haskell and its kin.
- Some tactics change the environment, for example specialize $h\ l$ replaces a hypothesis h of type $\forall k, p\ k$ with its specialization of type $p\ \ell$; use 'have' to duplicate a hypothesis.
- There are subtle differences between 'use', 'refine' and 'exists'. There are cases where the follow up work done by 'use' will close a goal that is left open by 'refine', for instance
  `example (n : Nat) (h : n = 4) : ∃ k, n = 2*k := by ...`
  can be completed by either `use 2` or `refine ⟨2, ?_⟩; rw [h]`.
- 'have' records only that the type is inhabited whereas 'let' remembers the definition. For propositions, by proof-irrelevance, the fact that the type is inhabited is all that can be said, and 'have' is idiomatic. For data values, such as $5 : \mathrm{Nat}$, 'let' should be used.
- push_neg is more general, for instance $\neg\neg p$ transforms to p.

Further tactics. When type signatures are optional they are given here using parentheses. Types can be refined to control where rw matches. Note rw?, apply? and exact? which will *suggest* lemmas from Mathlib. Bold tactics may be most useful.

Tactic	**Use**
apply	apply $(h : p \to q)$: transform goal q to goal p
assumption	close goal of type p if value of type p is in scope
by_contra	by_contra h: change goal to False and introduce $h : \neg p$ as new hypothesis
by_cases	by_cases $h : p$ · WHERE $h : p$ · WHERE $h : \neg p$
calc	calc $x = y :=$ by exact $h : x = y$ $\_ = z :=$ by simp $[(h' : y = z)]$ constructing a value of type $x = z$
change	change q: change goal to q when q is definitionally equivalent to current goal
congr gcongr	use congruence rules for equality (congr) inequality (gcongr): gcongr will replace goal $x + y \le a + b$ with subgoals $x \le a$ and $y \le b$
contradiction	close goal of any type if False is in scope, or any pair of $h : p$ and $h' : \neg p$ values are in scope; compare False.elim $(h : \mathrm{False}) : \star$
convert	souped up refine; matches goal on given pattern leaving subgoals
exact	exact $(h : p)$: close a goal of type p
have	have $h_p : p :=$ by $\ldots$: new hypothesis of type p, do *not* parenthesise ~~$(h_p : p)$~~
induction	induction n with $\mid$ zero $\implies$ by $\ldots$ $\mid$ succ $n\ h_{\mathrm{ind}} \implies$ by $\ldots$
rw	rw $[(h : p \longleftrightarrow q)]$: rewrite p in goal with q (once) rw $[(h : p \longleftrightarrow q), (h' : p' \longleftrightarrow q')]$: same as rw $[h]$; rw $[h']$ rw $[\leftarrow (h : p \longleftrightarrow q)]$: rewrite q in goal with p (once) rw $[(h : x = y)]$: rewrite x in goal as y (once) rw $[\leftarrow (h : x = y)]$: rewrite y in goal as x (once) rw $\ldots$ at k: as above, rewriting hypothesis k (once) rw $\ldots$ at *: as above, rewriting all hypotheses and goal (but still once each)
set/let	set x : Nat := 3: for local bindings
show	show p: state that current goal has type p for clarity in longer proofs
simp	simp: simplify goal using simp lemmas simp [h, h']: simplify goal *also* using specified propositions simp at k: simplify hypothesis k using simp lemmas simp [h, h'] at k: simplify k *also* using specified propositions simp only [h, h'] at k: simplify k *only* using specified propositions
unfold	unfold h: replace h with its definition; it may help to define using 'abbrev'

rwa and simpa mimic rw and simp but fail unless the goal is then closed by applying assumption

Sledgehammer tactics. Try **grind**, **linarith** (inequalities), **nlinarith** (more general inequalities), **omega** and **tauto** (tautologies).