

lean-mode

emacs mode for Lean Theorem Prover

Soonho Kong
soonhok@cs.cmu.edu

Leonardo de Moura
leonardo@microsoft.com

Features

- Show type/overload information at point
- On-the-fly syntax check
- Auto completion
- Jump to definition
- Set Lean options
- Eval Lean commands
- and More to come!

Configuration

(Scenario#1: build lean from source)

```
;; Add the following lines to your emacs configuration (.emacs file for example)
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)

;; Install required/optional packages for lean-mode
(defvar lean-mode-required-packages
  '(company dash dash-functional flycheck f
      fill-column-indicator s lua-mode mmm-mode))
(let ((need-to-refresh t))
  (dolist (p lean-mode-required-packages)
    (when (not (package-installed-p p))
      (when need-to-refresh
        (package-refresh-contents)
        (setq need-to-refresh nil))
      (package-install p)))

;; Set up lean-root path
(setq lean-rootdir "~/projects/lean")
(setq-local lean-emacs-path
            (concat (file-name-as-directory lean-rootdir)
                    (file-name-as-directory "src")
                    "emacs"))
(add-to-list 'load-path (expand-file-name lean-emacs-path))
(require 'lean-mode)
```

Please modify this!

Configuration

(Scenario#2: install lean via apt-get)

```
;; Add the following lines to your emacs configuration (.emacs file for example)
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)

;; Install required/optional packages for lean-mode
(defvar lean-mode-required-packages
  '(company dash dash-functional flycheck f
      fill-column-indicator s lua-mode mmm-mode))
(let ((need-to-refresh t))
  (dolist (p lean-mode-required-packages)
    (when (not (package-installed-p p))
      (when need-to-refresh
        (package-refresh-contents)
        (setq need-to-refresh nil))
      (package-install p))))

;; Set up lean-root path
(setq lean-rootdir "/usr")
(setq-local lean-emacs-path "/usr/share/emacs/site-lisp/lean")
(add-to-list 'load-path (expand-file-name lean-emacs-path))
(require 'lean-mode)
```

Configuration

(Scenario#3: install lean via homebrew)

```
;; Add the following lines to your emacs configuration (.emacs file for example)
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)

;; Install required/optional packages for lean-mode
(defvar lean-mode-required-packages
  '(company dash dash-functional flycheck f
      fill-column-indicator s lua-mode mmm-mode))
(let ((need-to-refresh t))
  (dolist (p lean-mode-required-packages)
    (when (not (package-installed-p p))
      (when need-to-refresh
        (package-refresh-contents)
        (setq need-to-refresh nil))
      (package-install p))))

;; Set up lean-root path
(setq lean-rootdir "/usr/local")
(setq-local lean-emacs-path "/usr/local/share/emacs/site-lisp/lean")
(add-to-list 'load-path (expand-file-name lean-emacs-path))
(require 'lean-mode)
```

Configuration

(Scenario#4: install lean in Windows)

```
;; Add the following lines to your emacs configuration (.emacs file for example)
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)

;; Install required/optional packages for lean-mode
(defvar lean-mode-required-packages
  '(company dash dash-functional flycheck f
      fill-column-indicator s lua-mode mmm-mode))
(let ((need-to-refresh t))
  (dolist (p lean-mode-required-packages)
    (when (not (package-installed-p p))
      (when need-to-refresh
        (package-refresh-contents)
        (setq need-to-refresh nil))
      (package-install p))))

;; Set up lean-root path
(setq lean-rootdir "\\lean-0.2.0-windows")
(setq-local lean-emacs-path "\\lean-0.2.0-windows\\share\\emacs\\site-lisp\\lean")
(add-to-list 'load-path (expand-file-name lean-emacs-path))
(require 'lean-mode)
```

```
-- Copyright (c) 2014 Microsoft Corporation. All rights reserved.
-- Released under Apache 2.0 license as described in the file LICENSE.
-- Author: Leonardo de Moura
```

```
import logic.axioms.hilbert logic.axioms.funext
open eq.ops nonempty inhabited
```

```

-- Diaconescu's theorem
-- Show that Excluded middle follows
-- Hilbert's choice operator, function extensionality and Prop extensionality
context
```

```
hypothesis propext {a b : Prop} : (a → b) → (b → a) → a = b
parameter p : Prop
```

```
private definition u [reducible] := epsilon (λx, x = true ∨ p)
```

```
private definition v [reducible] := epsilon (λx, x = false ∨ p)
```

```
private lemma u_def : u = true ∨ p :=
epsilon_spec (exists.intro true (or.inl rfl))
```

```
private lemma v_def : v = false ∨ p :=
epsilon_spec (exists.intro false (or.inl rfl))
```

```
private lemma uv_implies_p : ¬(u = v) ∨ p :=
or.elim u_def
  (assume Hut : u = true, or.elim v_def
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf⁻¹ ▸ Hut⁻¹ ▸ true_ne_false,
      or.inl Hne)
    (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)
```

```
private lemma p_implies_uv : p → u = v :=
assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
funext (take x : Prop,
  have Hl : (x = true ∨ p) → (x = false ∨ p), from
    assume A, or.inr Hp,
  have Hr : (x = false ∨ p) → (x = true ∨ p), from
    assume B, or.inr Hp.
```

Show information at point
(type, overloading, casting, etc)

```
private definition u [reducible] := epsilon (λx, x = true ∨ p)
```

```
private definition v [reducible] := epsilon (λx, x = false ∨ p)
```

```
private lemma u_def : u = true ∨ p :=
  epsilon_spec (exists.intro true (or.inl rfl))
```

```
private lemma v_def : v = false ∨ p :=
  epsilon_spec (exists.intro false (or.inl rfl))
```

Show type information of a sub-term in parens
(put a cursor on a open-paren)

```
private lemma uv_implies_p : ¬(u = v) ∨ p :=
  or.elim u_def
```

```
  (assume Hut : u = true, or.elim v_def
```

```
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf-1 ∙ Hut-1 ∙ true_ne_false,
      or.inl Hne)
```

```
    (assume Hp : p, or.inr Hp))
```

```
  (assume Hp : p, or.inr Hp)
```

```
private lemma p_implies_uv : p → u = v :=
```

```
  assume Hp : p,
```

```
  have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
```

```
    funext (take x : Prop,
```

```
      have Hl : (x = true ∨ p) → (x = false ∨ p), from
```

```
        assume A, or.inr Hp,
```

```
        have Hr : (x = false ∨ p) → (x = true ∨ p), from
```

```
          assume A, or.inr Hp,
```

```
          show (x = true ∨ p) = (x = false ∨ p), from
```

```
            propext Hl Hr),
```

```
  show u = v, from
```

```
    Hpred ∙ (eq.refl (epsilon (λ x, x = true ∨ p)))
```

```
theorem em : p ∨ ¬p :=
```

```
  have H : ¬(u = v) → ¬p, from mt p_implies_uv,
```

```
  or.elim uv_implies_p
```

```
    (assume Hne : ¬(u = v), or.inr (H Hne))
```

```
    (assume Hp : p, or.inl Hp)
```

```
end
```




```
-- Copyright (c) 2014 Microsoft Corporation. All rights reserved.
-- Released under Apache 2.0 license as described in the file LICENSE.
-- Author: Leonardo de Moura
```

```
import logic.axioms.hilbert logic.axioms.funext
open eq.ops nonempty inhabited
```

On-the-fly syntax check

```

-- Diaconescu's theorem
-- Show that Excluded middle follows from
-- Hilbert's choice operator, function extensionality and Prop extensionality
context
```

```
hypothesis propext {a b : Prop} : (a → b) → (b → a) → a = b
parameter p : Prop
```

```
private definition u [reducible] := epsilon (λx, x = true ∨ p)
```

```
private definition v [reducible] := epsilon (λx, x = false ∨ p)
```

```
private lemma u_def : u = true ∨ p :=
epsilon_spec (exists.intro true (or.inl rfl))
```

```
private lemma v_def : v = false ∨ p :=
epsilon_spec (exists.intro false (or.inl rfl))
```

```
private lemma uv_implies_p : ¬(u = v) ∨ p :=
```

```
!or.elim u
  (assume Hu : u = true, or.elim v_def
    (assume Hv : v = false,
      have Huv : ¬(u = v), from Hvf-1 ∙ Hut-1 ∙ true_ne_false,
      or.inl Hu)
    (assume Hp : p, or.inr Hp))
  (assume Hv : v = false ∨ p, or.inr Hv)
```

```
private lemma p_implies_uv : p → u = v :=
assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
funext (take x : Prop,
  have Hl : (x = true ∨ p) → (x = false ∨ p), from
    assume A, or.inr A,
  have Hr : (x = false ∨ p) → (x = true ∨ p), from
    assume A, or.inl A,
  or.elim A Hp,
  or.inl A,
  or.inr A)
```

```
private definition v [reducible] := epsilon (λx, x = false ∨ p)
```

```
private lemma u_def : u = true ∨ p :=
epsilon_spec (exists.intro true (or.inl rfl))
```

```
private lemma v_def : v = false ∨ p :=
epsilon_spec (exists.intro false (or.inl rfl))
```

```
private lemma uv_implies_p : ¬(u = v) ∨ p :=
or.elim u
```

```
(assume Hut : u = true, or.elim v_def
  (assume Hvf : v = false,
    have Hne : ¬(u = v), from Hvf-1 ∙ Hut-1 ∙ true_ne_false,
    or.inl Hne)
  (assume Hp : p, or.inr Hp))
(assume Hp : p, or.inr Hp)
```

```
private lemma p_implies_uv : p → u = v :=
assume Hp : p,
```

On-the-fly syntax check

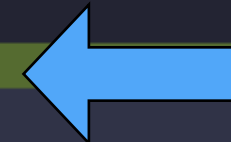
C-c ! 1 : show list of errors

```
U:--- diaconescu.lean 33% (26,9) Git (Lean Hi ElDoc company MMM FlyC:2/0 GitGutter Projectile[lean] MRev guru Fill) 05
```

```
Line Col Level ID Message (Checker)
```

```
26 9 error unknown identifier 'u'... (lean-checker)
```

```
49 11 error unknown identifier 'uv_implies_p'... (lean-checker)
```



```

    or.inl Hne)
  (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

```
private lemma p_implies_uv : p → u = v :=

```

```

assume Hp : p,

```

```

have Hpred : (λ x, x = true v p) = (λ x, x = false v p), from

```

```

funext (take x : Prop,

```

```

  have Hl : (x = true v p) → (x = false v p), from

```

```

    assume A, or.inr Hp,

```

```

  have Hr : (x = false v p) → (x = true v p), from

```

```

    assume A, or.inr Hp,

```

```

  show (x = true v p) = (x = false v p), from

```

```

    propext Hl Hr),

```

```

show u = v, from

```

```

  Hpred ▸ (eq.refl (epsilon (λ x, x = true v p)))

```

```

theorem em : p ∨ ¬p :=

```

```

have H : ¬(u = v) → ¬p, from mt p_implies_uv,

```

```

or.elim uv_implies_p

```

```

  (assume Hne : ¬(u = v), or.inl (H Hne))

```

```

  (assume Hp : p, or.inl Hp)

```

```

end

```

Auto-completion with type
tab



```

    or.inl Hne)
  (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

private lemma p_implies_uv : p → u = v :=
  assume Hp : p,
  have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
    funext (take x : Prop,
      have Hl : (x = true ∨ p) → (x = false ∨ p), from
        assume A, or.inr Hp,
      have Hr : (x = false ∨ p) → (x = true ∨ p), from
        assume A, or.inr Hp,
      show (x = true ∨ p) = (x = false ∨ p), from
        propext Hl Hr),
  show u = v, from
    Hpred ▸ (eq.refl (epsilon (λ x, x = true ∨ p)))

theorem em : p ∨ ¬p :=
  have H : ¬(u = v) → ¬p, from mt p_implies_uv,
  or.elim uv_implies_p
  ! (assume Hne : ¬(u = v), or.inl (H Hne))
  or.inr : b → a ∨ b
end
or.inl : a → a ∨ b
or.intro_left : Π (b : Prop), a → a ∨ b
or.intro_right : ∀ (a : Prop) {b : Prop}, b → a ∨ b
or.induction_on : a ∨ b → (a → C) → (b → C) → C
bool.induction_on : Π (n : bool), C bool.ff → C bool.tt → C n
tactic.expr.induction_on : Π (n : tactic.expr), C tactic.expr.builtin → C n
prod.rprod.intro : Ra a1 a2 → Rb b1 b2 → prod.rprod Ra Rb (prod.mk a1 b1) (prod.mk a2 b2)
char.induction_on : Π (n : char), (Π (a a a a a a a a : bool), C (char.mk a a a a a a a a)) → C n
not.intro : (a → false) → ¬ a
option.induction_on : Π (n : option A), C option.none → (Π (a : A), C (option.some a)) → C n
prod.induction_on : Π (n : prod A B), (Π (pr1 : A) (pr2 : B), C (prod.mk pr1 pr2)) → C n
prod.rprod.induction_on : prod.rprod Ra Rb a a → (Π {a1 : A} {b1 : B} {a2 : A} {b2 : B}, Ra a1 a2 → ...)

```

Auto-completion with type
tab

```

have Hne : ¬(u = v), from Hvf-1 ▸ Hut-1 ▸ true_ne_false,
or.inl Hne)
(assume Hp : p, or.inr Hp))
(assume Hp : p, or.inr Hp)

```

```
private lemma p_implies_uv : p → u = v :=
```

```
assume Hp : p,
```

```
have Hpred : (λ x, x = true v p) = (λ x, x = false v p), from
```

```
funext (take x : Prop,
```

```
have Hl : (x = true v p) → (x = false v p), from
```

```
assume A, or.inr Hp,
```

```
have Hr : (x = false v p) → (x = true v p), from
```

```
assume A, or.inr Hp,
```

```
show (x = true v p) = (x = false v p), from
```

```
propext Hl Hr),
```

```
show u = v, from
```

```
Hpred ▸ (eq.refl (epsilon (λ x, x = true v p)))
```

```
theorem em : p ∨ ¬p :=
```

```
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
```

```
or.elim uv_implies_p
```

```
(assume Hne : ¬(u = v), or.inr (H Hne))
```

```
(assume Hp : p, or.inl Hp)
```

```
end
```

Jump to definition
M-.




```

have Hne : ¬(u = v), from Hvf⁻¹ ▸ Hut⁻¹ ▸ true_ne_false,
or.inl Hne)
  (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

```
private lemma p_implies_uv : p → u = v :=
```

```
assume Hp : p,
```

```
have Hpred : (λ x, x = true v p) = (λ x, x = false v p), from
```

```
funext (take x : Prop,
```

```
have Hl : (x = true v p) → (x = false v p), from
```

```
assume A, or.inr Hp,
```

```
have Hr : (x = false v p) → (x = true v p), from
```

```
assume A, or.inr Hp,
```

```
show (x = true v p) = (x = false v p), from
```

```
propext Hl Hr),
```

```
show u = v, from
```

```
Hpred ▸ (eq.refl (epsilon (λ x, x = true v p)))
```

```
theorem em : p ∨ ¬p :=
```

```
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
```

```
or.elim uv_implies_p
```

```
(assume Hne : ¬(u = v), or.inr (H Hne))
```

```
(assume Hp : p, or.inl Hp)
```

```
end
```

Jump to definition
M-.

Can take a few seconds
for the first time.

```
calc_trans heq.of_heq_of_eq
calc_trans heq.of_eq_of_heq
calc_symm  heq.symm
```

```
/- and -/
```

```
notation a ∧ b := and a b
notation a ∧ b := and a b
```

```
variables {a b c d : Prop}
```

Jump to definition
M-.

```
theorem and.elim (H1 : a ∧ b) (H2 : a → b → c) : c :=
and.rec H2 H1
```

```
/- or -/
```

```
notation a ∨ b :=
notation a ∨ b :=
```

M-* will pop you back!

```
namespace or
```

```
theorem elim (H1 : a ∨ b) (H2 : a → c) (H3 : b → c) : c :=
  rec H2 H3 H1
end or
```

```
/- iff -/
```

```
definition iff (a b : Prop) := (a → b) ∧ (b → a)
```

```
notation a <=> b := iff a b
```

```
notation a ↔ b := iff a b
```

```
namespace iff
```

```
definition intro (H1 : a → b) (H2 : b → a) : a ↔ b :=
and.intro H1 H2
```

```
definition elim (H1 : (a → b) → (b → a) → c) (H2 : a ↔ b) : c :=
and.rec H1 H2
```

```
definition elim_left (H : a ↔ b) : a → b :=
elim (assume H1 H2, H1) H
```

```

or.elim u_def
  (assume Hut : u = true, or.elim v_def
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf-1 > Hut-1 > true_ne_false,
      or.inl Hne)
    (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

Set lean options
C-c C-o

```

private lemma p_implies_uv : p → u = v :=
  assume Hp : p,
  have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from

```

```

  funext (take x : Prop,
    have Hl : (x = true ∨ p) → (x = false ∨ p), from
      assume A, or.inr Hp,
    have Hr : (x = false ∨ p) → (x = true ∨ p), from
      assume A, or.inr Hp,
    show (x = true ∨ p) = (x = false ∨ p), from
      propext Hl Hr),

```

```

  show u = v, from
    Hpred > (eq.refl (epsilon (λ x, x = true ∨ p)))

```

□

```

theorem em : p ∨ ¬p :=
  have H : ¬(u = v) → ¬p, from mt p_implies_uv,
  or.elim uv_implies_p
    (assume Hne : ¬(u = v), or.inr (H Hne))
    (assume Hp : p, or.inl Hp)

```

```

end

```



```

U:--- diaconescu.lean Bot (46,0) Git (Lean ElDoc company LeanDebug*-1 MMM FlyC GitGutter Projectile[lean] MRev guru Fi
Option name: {class.instance_max_depth | class.trace_instances | class.unique_instances | elaborator.calc_assistant | elaborat
stor.fail_if_missing_field | elaborator.flycheck_goals | elaborator.ignore_instances | elaborator.local_instances | find_decl.
expensive | find_decl.max_steps | ...}

```



```

or.elim u_def
  (assume Hut : u = true, or.elim v_def
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf⁻¹ ▸ Hut⁻¹ ▸ true_ne_false,
      or.inl Hne)
    (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

Set lean options
C-c C-o

```

private lemma p_implies_uv : p → u = v :=
  assume Hp : p,
  have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
    funext (take x : Prop,
      have Hl : (x = true ∨ p) → (x = false ∨ p), from
        assume A, or.inr Hp,
      have Hr : (x = false ∨ p) → (x = true ∨ p), from
        assume A, or.inr Hp,
      show (x = true ∨ p) = (x = false ∨ p), from
        propext Hl Hr),
  show u = v, from
    Hpred ▸ (eq.refl (epsilon (λ x, x = true ∨ p)))
□
theorem em : p ∨ ¬p :=
  have H : ¬(u = v) → ¬p, from mt p_implies_uv,
  or.elim uv_implies_p
    (assume Hne : ¬(u = v), or.inr (H Hne))
    (assume Hp : p, or.inl Hp)
end

```



```

or.elim u_def
  (assume Hut : u = true, or.elim v_def
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf⁻¹ ▸ Hut⁻¹ ▸ true_ne_false,
      or.inl Hne)
    (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

Evaluate lean commands
C-c C-e

```

private lemma p_implies_uv : p → u = v :=
  assume Hp : p,
  have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
    funext (take x : Prop,
      have Hl : (x = true ∨ p) → (x = false ∨ p), from
        assume A, or.inr Hp,
      have Hr : (x = false ∨ p) → (x = true ∨ p), from
        assume A, or.inr Hp,
      show (x = true ∨ p) = (x = false ∨ p), from
        propext Hl Hr),
  show u = v, from
    Hpred ▸ (eq.refl (epsilon (λ x, x = true ∨ p)))
□
theorem em : p ∨ ¬p :=
  have H : ¬(u = v) → ¬p, from mt p_implies_uv,
  or.elim uv_implies_p
    (assume Hne : ¬(u = v), or.inr (H Hne))
    (assume Hp : p, or.inl Hp)
end

```



```

or.elim u_def
  (assume Hut : u = true, or.elim v_def
    (assume Hvf : v = false,
      have Hne : ¬(u = v), from Hvf-1 ▸ Hut-1 ▸ true_ne_false,
      or.inl Hne)
    (assume Hp : p, or.inr Hp))
  (assume Hp : p, or.inr Hp)

```

Evaluate lean commands

C-c C-e

```

private lemma p_implies_uv : p → u = v :=
assume Hp : p,
have Hpred : (λ x, x = true v p) = (λ x, x = false v p), from

```

```

  funext (take x : Prop,
    have Hl : (x = true v p) → (x = false v p), from
      assume A, or.inr Hp,
    have Hr : (x = false v p) → (x = true v p), from
      assume A, or.inr Hp,
    show (x = true v p) = (x = false v p), from
      propext Hl Hr),
  show u = v, from
    Hpred ▸ (eq.refl (epsilon (λ x, x = true v p)))

```

```

theorem em : p v ¬p :=
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
or.elim uv_implies_p
  (assume Hne : ¬(u = v), or.inr (H Hne))
  (assume Hp : p, or.inl Hp)
end

```



FAQS

Q: How can I type this
symbol '▸' ?

```

assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
  funext (take x : Prop,
    have H1 : (x = true ∨ p) = (x = false ∨ p), from
      propext (Hr : x = true ↔ x = false),
    show u = v, from
      Hpred ▸ (eq.refl (epsilon (λ x, x = true ∨ p)))

```

```

theorem em : p ∨ ¬p :=
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
or.elim uv_implies_p
  (assume Hne : ¬(u = v), or.inr (H Hne))
  (assume Hp : p, or.inl Hp)
end

```

FAQ#1

A: Press 'C-c C-k'!

```

assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
  funext (take x : Prop,
    have H1 : (x = true ∨ p) = (x = false ∨ p), from
      show (x = true ∨ p) = (x = false ∨ p), from
        propext (Hr),
    show u = v, from
      Hpred (eq.refl (epsilon (λ x, x = true ∨ p)))

```

```

theorem em : p ∨ ¬p :=
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
or.elim uv_implies_p
  (assume Hne : ¬(u = v), or.inr (H Hne))
  (assume Hp : p, or.inl Hp)
end

```



```

assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
  funext (take x : Prop,
    have Hl : (x = true ∨ p) → (x = false ∨ p), from
      assume A, or.inr Hp,
    have Hr : (x = false ∨ p) → (x = true ∨ p), from
      assume A, or.inr Hp,
    show (x = true ∨ p) = (x = false ∨ p), from
      propext Hl Hr),
show u = v, from
  Hpred ▯ (eq.refl (epsilon (λ x, x = true ∨ p)))

```

```

theorem em : p ∨ ¬p :=
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
or.elim uv_implies_p
  (assume Hne : ¬(u = v), or.inr (H Hne))
  (assume Hp : p, or.inl Hp)
end

```

Q: It seems that nothing is working!
What should I do?


```

assume Hp : p,
have Hpred : (λ x, x = true ∨ p) = (λ x, x = false ∨ p), from
  funext (take x : Prop,
    have Hl : (x = true ∨ p) → (x = false ∨ p), from
      assume A, or.inr Hp,
    have Hr : (x = false ∨ p) → (x = true ∨ p), from
      assume A, or.inr Hp,
    show (x = true ∨ p) = (x = false ∨ p), from
      propext Hl Hr),
show u = v, from
  Hpred ▯ (eq.refl (epsilon (λ x, x = true ∨ p)))

```

```

theorem em : p ∨ ¬p :=
have H : ¬(u = v) → ¬p, from mt p_implies_uv,
or.elim uv_implies_p
  (assume Hne : ¬(u = v), or.inr (H Hne))
  (assume Hp : p, or.inl Hp)
end

```

A: Keep calm and run
 “M-x lean-server-restart-process”
 then please file a bug report!
 (with reproducible steps)

Bug Reports, Feature Requests

<https://github.com/leanprover/lean/issues/new>

Contributions are Welcome!