Crash Course Lean

Suzan Erven & Amber Pater

- Theorem example
- Tactics
- Interacting with Lean

Theorem

```
theorem t1 (p q : Prop) (hp : p) (hq : q) : p ∧ q ∧ p :=
...
```

Context *Goal*

- Theorem example
- Tactics
- Interacting with Lean

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;        repeat
<|>      try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Variables

```
variables a b : nat
variables (h : a = b)

include h
example : a = b :=
begin
  apply h
end

omit h
```

```
example (a b : nat)
(h : a = b) : a = b :=
begin
  apply h
end
```

```
variables a b : nat
variables (h : a = b)

example: a = b :=
let h := h in
begin
  apply h
end
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactic combinators

- `foo; bar`
- `repeat foo`
- `foo <|> bar`
- `try foo`

- Performs `bar` on all subgoals created by `foo`
- Performs `foo` zero or more times, until it fails
- Performs `foo`, and if that fails performs `bar` instead
- Performs `foo`, but is allowed to fail. Same as `foo <|> skip`

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;        repeat
<|>      try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

Tactics

```
intro    revert
left     right
assumption
constructor
split
rewrite
simplify
```

```
sorry
```

```
exact
apply    fapply
```

```
reflexivity
transitivity
symmetry
```

```
generalize
cases
existsi
contradiction
```

```
variables
include      omit
let … in …
```

```
have         show
from         by
begin … end
{ … }
```

```
;            repeat
<|>          try
```

```
any_goals
all_goals
focus
goal1
solve1
tactic
done
```

- Theorem example
- Tactics
- Interacting with Lean

Importing Files

- `import foo bar.asdf`
  Imports the files `foo.lean` and `bar/asdf.lean`

- `import .foo ..bar.asdf`
  Imports `foo.lean` from the current directory, and `bar/asdf.lean` relative to the parent directory

- Importing is transitive

Sections

- Defines scope for variables, local definitions etc.
- Allows for parameters / hypothesis / conjecture

```
section foo

...

end foo
```

Namespaces

```
namespace foo
def bar : ℕ := 1
#check bar ✓
end foo
```

```
namespace foo
def bar : ℕ := 1
end foo
#check bar ✗
```

```
namespace foo
def bar : ℕ := 1
end foo
#check foo.bar ✓
```

```
namespace foo
def bar : ℕ := 1
end foo
open foo
#check bar ✓
```

Namespaces

- Alias already in use → overloading
- Use `foo.bar` or `_root_.bar` to point to a specific `bar`
- `protected def` prevents overloading:

```
namespace foo
protected def bar : ...
```

Creates only `foo.bar`, not also `bar`.

Attributes

```
@[simp, refl]
theorem …
…
```

```
attribute [simp, refl]
theorem …
…
```

```
theorem foo …
…

attribute [simp, refl] foo
```

```
section foo
local attribute bar
…
end foo
```

Notation and infix

```
notation `[` a `**` b `]` := a * b + 1

def mul_squares (a b : ℕ) := a * a * b * b

infix `<*>`:50 := mul_squares

#reduce [2 ** 3]
-- [2 ** 3] = 2 * 3 + 1 = 7
#reduce 2 <*> 3
-- 2 <*> 3 = 2 * 2 * 3 * 3 = 36
```

- `infix` is by default left associative, use `infixr` to be right associative

Coercions

Lean can detect & insert coercion on the latter argument:

```
variable n : ℕ
variable i : ℤ

#check i + n     -- i + ↑n : ℤ
```

But it has to be done manually on the first:

```
variable n : ℕ
variable i : ℤ

#check ↑n + i          -- i + ↑n: ℤ
```

Thank you!