SSReflect - Logics & Basic tactics

Laurence Rideau
12 March

MAP INTERNATIONAL SPRING SCHOOL ON FORMALIZATION OF MATHEMATICS 2012

SOPHIA ANTIPOLIS, FRANCE / 12-16 MARCH

Inria informatics mathematics
ihp Institut Henri Poincaré
MICROSOFT RESEARCH INRIA JOINT CENTRE

SSReflect – Reminder

(SSR = *Small Scale Reflection*)

SSReflect: extension of Coq

- developed while formalizing the Four Color Theorem (2004),
- now used for the Odd Order Theorem.

SSReflect – Reminder

(SSR = *Small Scale Reflection*)

SSReflect: extension of Coq

- developed while formalizing the Four Color Theorem (2004),
- now used for the Odd Order Theorem.

Changes with standard Coq:

- Vernacular (Commands) and Gallina are mostly unchanged (e.g., `Definition`, `Lemma`, `forall`, `match with`);
- standard tactics are still available
- some tactics are superseded (e.g., `apply`, `rewrite`)
- new libraries are provided (e.g., `nat`, `seq`)

Design Decisions

- Simplify and generalize the syntax of tactics.

Design Decisions

- Simplify and generalize the syntax of tactics.
- Add some ways to structure the scripts,
  so that breakages are easier to understand.

Design Decisions

- Simplify and generalize the syntax of tactics.
- Add some ways to structure the scripts,
  so that breakages are easier to understand.
- Force the user to explicitly name things.

Design Decisions

- Simplify and generalize the syntax of tactics.
- Add some ways to structure the scripts,
  so that breakages are easier to understand.
- Force the user to explicitly name things.
- Ease the use of boolean reflection.

Outline

Outline

1. Logics
   - First Order Logic
   - Booleans
2. Tactics, Tacticals
3. Proof Structure

Minimal Propositional Logic

- Propositional variables: `P Q R` ...
- Propositions: `(even 4) (x < 10) (7 <= 2)`
- Implication: `->`
- Formulas: `(P -> Q) -> (Q -> R) -> P -> R`

Minimal Propositional Logic

- Propositional variables: `P Q R ...`
- Propositions: `(even 4) (x < 10) (7 <= 2)`
- Implication: `->`
- Formulas: `(P -> Q) -> (Q -> R) -> P -> R`

- Propositional are of sort Prop : `(P : Prop)`.
- Declaring variables: `Variables P Q R :Prop.`

Minimal Propositional Logic

- Propositional variables: `P Q R` ...
- Propositions: `(even 4) (x < 10) (7 <= 2)`
- Implication: `->`
- Formulas: `(P -> Q) -> (Q -> R) -> P -> R`

- Propositional are of sort Prop : `(P : Prop)`.
- Declaring variables: `Variables P Q R :Prop.`
- Any term of type P `(p : P)` is a proof of `P`.

State and Proof a theorem

```
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)
```

State and Proof a theorem

```
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)
```

$$\left.\begin{array}{c}\vdots \\ P : Prop \\ Q : Prop \\ R : Prop\end{array}\right\}$$ named hypotheses (Context)

$$\underbrace{(P \to Q) \to (Q \to R) \to P}_{Assumptions} \to \underbrace{R}_{Conclusion}$$ } current goal

State and Proof a theorem

```
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)
```

$$\left.\begin{array}{c} \vdots \\ P : Prop \\ Q : Prop \\ R : Prop \end{array}\right\} \text{named hypotheses (Context)}$$

$$\underbrace{(P \to Q) \to (Q \to R) \to P}_{\textit{Assumptions}} \to \underbrace{R}_{\textit{Conclusion}} \quad \} \text{ current goal}$$

Tactic: any operation that allows the simplification, decomposition into subgoals, or resolution of a goal.

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq.
```

$$\cfrac{\begin{array}{l} P : Prop \\ Q : Prop \\ R : Prop \\ Hpq : (P \to Q) \end{array}}{(Q \to R) \to P \to R}$$

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
```

$$
\begin{array}{l}
P : Prop \\
Q : Prop \\
R : Prop \\
Hpq : (P \to Q) \\
Hqr : (Q \to R) \\
p : P \\
\hline\hline
R
\end{array}
$$

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.
```

$$
\begin{array}{l}
P : Prop \\
Q : Prop \\
R : Prop \\
Hpq : (P \rightarrow Q) \\
p : P \\
\hline\hline
Q
\end{array}
$$

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.
apply: (Hpq).
```

$$
\begin{array}{l}
P : Prop \\
Q : Prop \\
R : Prop \\
Hpq : (P \to Q) \\
p : P \\
\hline\hline
P
\end{array}
$$

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.
apply: Hpq.
exact: p.
```

Proof completed.

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.
exact: (Hpq p).
```

Proof completed.

Proof

```
Theorem command:
Lemma imp_trans :(P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.
exact: (Hpq p).
Qed.
```

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`
  - as a goal: `move=> P Q R.`

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`
  - as a goal: `move=> P Q R.`
  - as an hypothesis named `H`:
    `apply: H.` `apply: (H A B).` or ...

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`
    - as a goal: `move=> P Q R.`
    - as an hypothesis named `H`:
      `apply: H.` `apply: (H A B).` or ...
- `forall n:nat, 0 <= n`

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`
  - as a goal: `move=> P Q R.`
  - as an hypothesis named `H`:
    `apply: H.` `apply: (H A B).` or ...
- `forall n:nat, 0 <= n`
  - `move=> n.`

Minimal Propositional Logic with universal quantifier

- `forall (P Q R :Prop), (P ->Q)-> (Q -> R) -> P -> R`
  - as a goal: `move=> P Q R.`
  - as an hypothesis named `H`:
    `apply: H.` `apply: (H A B).` or ...
- `forall n:nat, 0 <= n`
  - `move=> n.`
  - `apply: H.` `apply: (H a).`

Propositional Logic, Conjunction

- Conjunction : `A /\ B`

Propositional Logic, Conjunction

- Conjunction : `A /\ B`
  - `case: ab. (* Break the (ab :A /\ B) hypothesis *)`

Propositional Logic, Conjunction

- Conjunction : `A /\ B`
  - `case: ab. (* Break the (ab : A /\ B) hypothesis *)`

$$\begin{array}{c} \texttt{ab : A /\textbackslash\ B} \\ \hline\hline \texttt{G} \end{array} \quad \rightarrow \quad \begin{array}{c} \\ \hline\hline \texttt{A -> B -> G} \end{array}$$

Propositional Logic, Conjunction

- Conjunction : `A /\ B`
  - `case: ab. (* Break the (ab :A /\ B) hypothesis *)`

$$\frac{\texttt{ab : A /\textbackslash\ B}}{\texttt{G}} \quad \rightarrow \quad \frac{}{\texttt{A -> B -> G}}$$

  - `split. (* Prove a conjunction :A /\B *)`

Propositional Logic, Conjunction

- Conjunction : `A /\ B`
  - `case: ab. (* Break the (ab :A /\ B) hypothesis *)`

```
  ab : A /\ B
 ===========        →    ================
      G                   A -> B -> G
```

  - `split. (* Prove a conjunction :A /\B *)`

```
 ========    →   =====     =====
  A /\ B           A         B
```

Propositional Logic, Disjunction

- Disjunction : `A \/ B`

Propositional Logic, Disjunction

- Disjunction : `A \/ B`
  - `case: ab. (* Break the (ab :A \/ B) hypothesis *)`

Propositional Logic, Disjunction

- Disjunction : `A \/ B`
  - `case: ab. (* Break the (ab : A \/ B) hypothesis *)`

```
ab : A \/ B                    
===========   →   ========    ========
     G             A -> G      B -> G
```

Propositional Logic, Disjunction

- Disjunction : `A \/ B`
  - `case: ab. (* Break the (ab :A \/ B) hypothesis *)`

$$\frac{\texttt{ab : A \textbackslash/ B}}{\texttt{G}} \rightarrow \frac{}{\texttt{A -> G}} \qquad \frac{}{\texttt{B -> G}}$$

  - `left. (* Prove a disjunction :A \/ B *)`
    `(* by choosing the left part *)`

Propositional Logic, Disjunction

- Disjunction : `A \/ B`
  - `case: ab. (* Break the (ab :A \/ B) hypothesis *)`

$$\frac{\texttt{ab : A \textbackslash/ B}}{\texttt{G}} \quad \rightarrow \quad \frac{}{\texttt{A -> G}} \qquad \frac{}{\texttt{B -> G}}$$

  - `left. (* Prove a disjunction :A \/ B *)`
    `(* by choosing the left part *)`

$$\frac{}{\texttt{A \textbackslash/ B}} \quad \rightarrow \quad \frac{}{\texttt{A}}$$

Propositional Logic, Disjunction

- Disjunction : `A \/ B`
  - `case: ab. (* Break the (ab :A \/ B) hypothesis *)`

```
  ab : A \/ B
  ===========    →   ========      ========
       G              A -> G        B -> G
```

  - `left. (* Prove a disjunction :A \/ B *)`
    `(* by choosing the left part *)`

```
  ======      →   ===
  A \/ B           A
```

  - `right. (* Prove a disjunction :A \/ B *)`
    `(* by choosing the right part *)`

Propositional Logic, Negation

- Negation : `~B`

Propositional Logic, Negation

- Negation : `~B`
  - `~B` is defined as `(B -> False)`

Propositional Logic, Negation

- Negation : `~B`
  - `~B` is defined as `(B -> False)`
  - `move=> B. (* To prove the goal (~B)*)`

Propositional Logic, Negation

- Negation : `~B`
  - `~B` is defined as `(B -> False)`
  - `move=>B. (* To prove the goal (~B)*)`

$$\frac{\texttt{...}}{\texttt{~B}} \quad \rightarrow \quad \frac{\texttt{b : B}}{\texttt{False}}$$

Propositional Logic, Negation

- Negation : `~B`
  - `~B` is defined as `(B -> False)`
  - `move=> B. (* To prove the goal (~B)*)`

$$\frac{\texttt{...}}{\texttt{\~{}B}} \quad \rightarrow \quad \frac{\texttt{b : B}}{\texttt{False}}$$

  - Then `apply: H. (* for a (H :~C) in the context*)`

Existential Quantifier

Existential: `exists n:nat, P n`

```
(* P is a predicate on nat (P :nat ->Prop)*)
```

Existential Quantifier

Existential: `exists n:nat, P n`

```
(* P is a predicate on nat (P :nat ->Prop)*)
```

- `exists 2. (*To prove an exists, give a witness *)`

Existential Quantifier

Existential: `exists n:nat, P n`

`(* P is a predicate on nat (P :nat ->Prop)*)`

- `exists 2. (*To prove an exists, give a witness *)`

```
        ...                     ...
=====================    →    =======
exists n:nat, P n              P 2
```

Existential Quantifier

Existential: `exists n:nat, P n`

`(* P is a predicate on nat (P :nat ->Prop)*)`

- `exists 2. (*To prove an exists, give a witness *)`

```
        ...                          ...
=====================        →      =====
exists n:nat, P n                    P 2
```

- `case: Hex.`

  `(* To break the (Hex:exists n, P n)hypothesis *)`

  `(* combined with (move=>n Hn.)*)`

Existential Quantifier

Existential: `exists n:nat, P n`

```
(* P is a predicate on nat (P :nat ->Prop)*)
```

- `exists 2. (*To prove an exists, give a witness *)`

$$\frac{\dots}{\texttt{exists n:nat, P n}} \rightarrow \frac{\dots}{\texttt{P 2}}$$

- `case: Hex.`

```
(* To break the (Hex:exists n, P n)hypothesis *)
(* combined with (move=>n Hn.)*)
```

$$\frac{\texttt{Hex: exists n, P n}}{\texttt{G}} \rightarrow \frac{\begin{array}{l}\texttt{n : nat}\\ \texttt{Hn : P n}\end{array}}{\texttt{G}}$$

Outline

1. Logics
   - First Order Logic
   - Booleans
2. Tactics, Tacticals
3. Proof Structure

Booleans

- `Inductive bool := true | false.`

Booleans

- `Inductive bool := true | false.`
- Operators: "`&&`", "`||`", "`~~`", "`==>`", "`(+)`".

Booleans

- `Inductive bool := true | false.`
- Operators: "`&&`", "`||`", "`~~`", "`==>`", "`(+)`".

b_1	b_2	b_1 && b_2	b_1 || b_2	b_1 ==> b_2	b_1 (+) b_2
T	T	T	T	T	F
T	F	F	T	F	T
F	T	F	T	T	T
F	F	F	F	T	F

Booleans

- `Inductive bool := true | false.`
- Operators: "`&&`", "`||`", "`~~`", "`==>`", "`(+)`".

- Some notations
  - `"[ && b1 , b2 , .. , bn & c ]" := (b1 && (b2 && .. (bn && c).. ))`
  - `"[ || b1 , b2 , .. , bn | c ]" := (b1 || (b2 || .. (bn || c).. ))`

Booleans

- `Inductive bool := true | false.`
- Operators: "`&&`", "`||`", "`~~`", "`==>`", "`(+)`".

- `is_true : bool -> Prop`.
  - `fun b : bool => b = true`.
  - Notation : `"x 'is_true'" := (is_true x)`

Booleans in proofs

- Reason by case on a boolean:
  `case: a.`

Booleans in proofs

- Reason by case on a boolean:

```
case: a.
```

```
⋮
a : bool
b : bool
================================
a (+) b = (a && ~~b)|| (~~a && b)
```

$\rightarrow$

```
⋮
b : bool
================================
true (+)b = (true && ~~b)|| (~~true && b)
```

Booleans in proofs

- Reason by case on a boolean:
  `case: a.`

```
⋮
a : bool
b : bool
==================
a (+) b = (a && ~~b)|| (~~a && b)
```

$\rightarrow$

```
⋮
b : bool
==================
true (+)b = (true && ~~b)|| (~~true && b)
⋮
b : bool
==================
false (+)b = (false && ~~b)|| (~~false && b)
```

Booleans in proofs(2)

- Compute, simplify:

```
rewrite /=.
```

Booleans in proofs(2)

- Compute, simplify:
  `rewrite /=.`

```
⋮

b : bool
=====================================
true (+)b = (true && ~~ b)|| (~~ true && b)
```

→

```
⋮
b : bool
=====================
~~ b = ~~ b || false
```

Outline

Tactics / Tacticals

- Tactic: any operation that allows the simplification, decomposition into subgoals, or resolution of a goal.
- Tactical: any function of tactics (eg. ; the composition of two tactics).

Tactics and Tacticals

- `move=>`
- `by` tactical
- `apply:`
- `exact:`
- `case:`
- `elim:`
- `rewrite`

Introduction Tactic

- `move=> a b c.`
  pops the top 3 elements of the goal, and
  it puts them into the context with names *a*, *b*, and *c*.
- `move=> _.`
  pops the first top element of the goal, without putting it in the context.
- `move=> a _ c.`

Tactical `by` and Tactics `apply` / `exact`

- "`by []`" tries to solve the current goal by some trivial means; it fails if it doesn't succeed.

Tactical `by` and Tactics `apply` / `exact`

- "`by []`" tries to solve the current goal by some trivial means; it fails if it doesn't succeed.
- "`by any_tactic`" applies the argument tactic, then tries to solve the current goal.
- "`apply: H`" applies H to the goal.

Tactical `by` and Tactics `apply` / `exact`

- "`by []`" tries to solve the current goal by some trivial means; it fails if it doesn't succeed.
- "`by any_tactic`" applies the argument tactic, then tries to solve the current goal.
- "`apply: H`" applies H to the goal.

$$
\begin{array}{c}
\vdots \\
\texttt{H: P -> Q} \\
\hline\hline
\texttt{Q}
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\vdots \\
\hline\hline
\texttt{P}
\end{array}
$$

Tactical `by` and Tactics `apply` / `exact`

- "`by []`" tries to solve the current goal by some trivial means; it fails if it doesn't succeed.
- "`by any_tactic`" applies the argument tactic, then tries to solve the current goal.
- "`apply: H`" applies H to the goal.

```
  ⋮
 H: P -> Q        →      ⋮
==========             =====
 Q                       P
```

- "`exact:H`" performs "`by apply: H`"

Tactics `case:` / `elim:`

Performs a case analysis / inductive elimination on the element given as an argument.

Tactics `case:` / `elim:`

Performs a case analysis / inductive elimination on the element given as an argument.

```
Inductive nat := O | S of nat
```

Tactics case: / elim:

Performs a case analysis / inductive elimination on the element given as an argument.

```
Inductive nat := O | S of nat
```

```
Lemma P_of_n forall n : nat, P n.
move=>n.
```

Tactics case: / elim:

Performs a case analysis / inductive elimination on the element given as an argument.

```
Inductive nat := O | S of nat
```

```
Lemma P_of_n forall n : nat, P n.
move=>n.
```

```
case:n.
```

1. $\overline{\overline{}}$ `P O`
2. $\overline{\overline{}}$ `forall n : nat, P (S n)`

Tactics case: / elim:

Performs a case analysis / inductive elimination on the element given as an argument.

```
Inductive nat := O | S of nat
```

```
Lemma P_of_n forall n : nat, P n.
move=>n.
```

```
elim:n.
```

1. $\overline{\overline{}}$ `P 0`

2. $\overline{\overline{}}$ `forall n, P n -> P (S n)`

Basic Rewriting tactic

Tactic "`rewrite` *items* ..." modifies subterms of the goal:

- "/*name*" unfolds a definition
- "-/*name*" folds a definition
- "*term*" rewrites (left to right) with a lemma or an hypothesis which conclusion is an equality

Basic Rewriting tactic

Tactic "`rewrite` *items* ..." modifies subterms of the goal:

- "*/name*" unfolds a definition
- "*-/name*" folds a definition
- "*term*" rewrites (left to right) with a lemma or an hypothesis which conclusion is an equality

```
 Eqab: a = b            Eqab: a = b
=============    →     =============
    P  a                P  b
```

Basic Rewriting tactic

Tactic "`rewrite` *items* ..." modifies subterms of the goal:

- "/*name*" unfolds a definition
- "-/*name*" folds a definition
- "*term*" rewrites (left to right) with a lemma or an hypothesis which conclusion is an equality

$$\frac{\texttt{Eqab: a = b}}{\texttt{P a}} \quad \rightarrow \quad \frac{\texttt{Eqab: a = b}}{\texttt{P b}}$$

- "-*term*" rewrites right to left

Multiple Rewriting and Occurrence selection

`rewrite` -*multiplicity term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

Multiple Rewriting and Occurrence selection

`rewrite -`*multiplicity term*

- "`?`": as many times as possible, possibly none,
- "`!`": as many times as possible, at least once,
- "*n*`?`": at most *n* times,
- "*n*`!`": exactly *n* times.

`rewrite -{`*number*`}`*term*

Multiple Rewriting and Occurrence selection

`rewrite` -*multiplicity* *term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

`rewrite` -{*number*}*term*

```
Lemma dbl a b : 2 * (a + b) = (b + a) + (a + b).
```

Multiple Rewriting and Occurrence selection

`rewrite -`*multiplicity* *term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

`rewrite -{`*number*`}`*term*

```
Lemma dbl a b : 2 * (a + b) = (b + a) + (a + b).

Proof.

 a : nat
 b : nat
=================================
 2 * (a + b) = (b + a) + (a + b)
```

Multiple Rewriting and Occurrence selection

`rewrite -`*multiplicity* *term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

`rewrite -{`*number*`}` *term*

```
Lemma dbl a b : 2 * (a + b) = (b + a) + (a + b).

rewrite -!addnA.
```

```
a : nat
b : nat
==============================
2 * (a + b) = b + (a + (a + b))
```

Multiple Rewriting and Occurrence selection

`rewrite -`*multiplicity term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

`rewrite -{`*number*`}`*term*

```
Lemma dbl a b : 2 * (a + b) = (b + a) + (a + b).

rewrite {2}addnC.
```

```
a : nat
b : nat
==============================
2 * (a + b) = (b + a) + (b + a)
```

Multiple Rewriting and Occurrence selection

`rewrite -`*multiplicity term*

- "?": as many times as possible, possibly none,
- "!": as many times as possible, at least once,
- "*n*?": at most *n* times,
- "*n*!": exactly *n* times.

`rewrite -{`*number*`}`*term*

```
Lemma dbl a b : 2 * (a + b) = (b + a) + (a + b).

rewrite -!addnA {2}addnC.
```

```
a : nat
b : nat
=================================
2 * (a + b) = (b + (a + (b + a)))
```

Outline

1. Logics
2. Tactics, Tacticals
3. Proof Structure
   - Forward reasoning
   - Proof control flow
   - Subgoal selectors

Forward Reasoning

- `have`
- `suffices` (`suff`)

Forward Reasoning: `have` / `suffices`

`have H :` *intermediate_goal*

performs a logical cut.

Forward Reasoning: `have` / `suffices`

`have H :` *intermediate_goal*

performs a logical cut.

```
Variable f : nat -> nat.
Variable P : nat -> Prop.

Lemma P_of_3:    P 3.
```

Forward Reasoning: `have` / `suffices`

`have H :` *intermediate_goal*

performs a logical cut.

```
Variable f : nat -> nat.
Variable P : nat -> Prop.

Lemma P_of_3:    P 3.

Proof.
have H: exists x, f x = 3.
```

Forward Reasoning: `have` / `suffices`

`have H :` *intermediate_goal*

performs a logical cut.

```
Variable f : nat -> nat.
Variable P : nat -> Prop.

Lemma P_of_3:   P 3.

Proof.
have H: exists x, f x = 3.
```

1. $\dfrac{}{\texttt{exists x, f x = 3}}$ 2. $\dfrac{\texttt{H: exists x, f x = 3}}{\texttt{P 3}}$

Forward Reasoning: `have` / `suffices`

`have H :` *intermediate_goal*

performs a logical cut.

```
Variable f : nat -> nat.
Variable P : nat -> Prop.

Lemma P_of_3:   P 3.

Proof.
have H: exists x, f x = 3.
```

Tactic "`suff`" also performs a logical cut, but it produces the two subgoals in the opposite order.

Proof control flow

- Tabulation (depending on the number of subgoals number)

Proof control flow

- Tabulation (depending on the number of subgoals number)
- Bullets -, $+$, *

Proof control flow

- Tabulation (depending on the number of subgoals number)
- Bullets -, $+$, *
- Proof terminators : `by` , `exact:`

A proof example

Emacs@zeniba-2

```
case E1: (abezoutn _ _) => [[| k1] [| k2]].
- rewrite !muln0 !gexpn0 mulg1 => H1.
  move/eqP: (sym_equal F0); rewrite -H1 orderg1 eqn_mul1.
  by case/andP; move/eqP.
- rewrite muln0 gexpn0 mulg1 => H1.
  have F1: t %| t * S k2.+1 - 1.
   apply: (@dvdn_trans (orderg x)); first by rewrite F0; exact: dvdn_mull.
   rewrite orderg_dvd; apply/eqP; apply: (mulgI x).
   rewrite -{1}(gexpn1 x) mulg1 gexpn_add leq_add_sub //.
   by move: P1; case t.
  rewrite dvdn_subr in F1; last by exact: dvdn_mulr.
  + rewrite H1 F0 -{2}(muln1 (p ^ l)); congr (_ * _).
    by apply/eqP; rewrite -dvdn1.
  + by move: P1; case: (t) => [| [| s1]].
- rewrite muln0 gexpn0 mul1g => H1.
-1:---  f.v          All L15     (coq Holes)------------------------------------
```

Subgoal Selectors

- Solving one subgoal with a single tactic:
  - *tactic* `; first by` *tactic*
  - *tactic* `; last by` *tactic*

Subgoal Selectors

- Solving one subgoal with a single tactic:
  - *tactic* ; `first` `by` *tactic*
  - *tactic* ; `last` `by` *tactic*
- Changing the order of subgoals:
  - *tactic* ; `first last` (or `last first`)