SSReflect - Logics & Basic tactics

Laurence Rideau
12 March
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- some tactics are superseded (e.g., apply, rewrite)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**.
SSR Tactics Structure

Outline

1. Logics
2. Tactics, Tacticals
3. Proof Structure
Outline

1 Logics
   - First Order Logic
   - Booleans

2 Tactics, Tacticals

3 Proof Structure
Minimal Propositional Logic

- Propositional variables: $P, Q, R, \ldots$
- Propositions: $(\text{even~4}) \land (x < 10) \land (7 \leq 2)$
- Implication: $\rightarrow$
- Formulas: $(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$
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 *)

\[
\begin{aligned}
\vdash \quad \begin{array}{l}
P : Prop \\
Q : Prop \\
R : Prop \\
\end{array}
\end{aligned}
\]

\[
(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R \quad \text{current goal}
\]

\[
\begin{aligned}
\text{Assumptions} & \quad \text{Conclusion}
\end{aligned}
\]
State and Proof a theorem

Lemma imp_trans : (P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R.
Proof. (* start the proof of a Lemma *)

\begin{align*}
& \vdots \\
& P : Prop \\
& Q : Prop \\
& R : Prop \\
\end{align*}
\begin{align*}
\{ & \text{named hypotheses (Context)} \\
& \text{current goal} \\
& \text{Assumptions} & \text{Conclusion} \\
& & \text{current goal} \\
\end{align*}

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.

\[
\begin{align*}
P & : \text{Prop} \\
Q & : \text{Prop} \\
R & : \text{Prop} \\
Hpq & : (P \rightarrow Q) \\
(\therefore Q \rightarrow R) & \rightarrow P \rightarrow R
\end{align*}
\]
Theorem command:  
Lemma imp_trans : (P -> Q) -> (Q -> R) -> P -> R.
Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.

\[ \begin{align*}
P & : \text{Prop} \\
Q & : \text{Prop} \\
R & : \text{Prop} \\
Hpq & : (P \rightarrow Q) \\
Hqr & : (Q \rightarrow R) \\
p & : P \\
\hline
\end{align*} \]

\[ \underline{P} \rightarrow R \]

Laurence Rideau  SSReflect - Logics & Basic tactics
**Proof**

**Theorem command:**

Lemma imp_trans : \((P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R\).

Proof. (* start the proof of a Lemma *)

move=> Hpq Hqr p.
apply: Hqr.

\[
\begin{align*}
P & : \text{Prop} \\
Q & : \text{Prop} \\
R & : \text{Prop} \\
Hpq & : (P \rightarrow Q) \\
p & : P \\
\hline
Q
\end{align*}
\]
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).

\[
\begin{aligned}
P & : Prop \\
Q & : Prop \\
R & : Prop \\
Hpq & : (P \rightarrow Q) \\
p & : P \\
P \\
\end{aligned}
\]
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.
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 \(\rightarrow\)Q) \(\rightarrow\) (Q \(\rightarrow\) R) \(\rightarrow\) P \(\rightarrow\) R
  
  - as a goal:  move=> P Q R.
Minimal Propositional Logic with universal quantifier

- \(\textbf{forall} \ (P \ Q \ R : \text{Prop}), \ (P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R\)
  - as a goal: \textit{move=> P Q R.}
  - as an hypothesis named \(H\):
    \textit{apply: H. \ apply: (H A B). or ...}
Minimal Propositional Logic with universal quantifier

- \( \forall (P, Q, R : \text{Prop}), (P \implies Q) \implies (Q \implies R) \implies P \implies R \)
  - as a goal: \text{move=} P \ Q \ R.
  - as an hypothesis named H:
    \text{apply=} H. \text{apply=} (H \ A \ B). \text{or ...}
- \( \forall n : \text{nat}, 0 \leq n \)
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** \((P \ Q \ R : \text{Prop}), \ (P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R\)
  - as a goal: \textit{move=>P Q R.}
  - as an hypothesis named \(H\):
    - \textit{apply:H.  apply: (H A B). or ...}

- **forall** \(n: \text{nat}, \ 0 \leq n\)
  - \textit{move=>n.}
  - \textit{apply:H.  apply: (H a).}
Propositional Logic, Conjunction

Conjunction : \( A \land B \)
Propositional Logic, Conjunction

- Conjunction : $A \land B$
  - case: $ab$. (* Break the $(ab : A \land B)$ hypothesis *)
Propositional Logic, Conjunction

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

\[
\frac{ab : A \land B}{G} \quad \rightarrow \quad \frac{A \rightarrow B \rightarrow G}{A \rightarrow B \rightarrow G}
\]
Propositional Logic, Conjunction

- Conjunction: \( A \land B \)
  - **case**: \( ab \). (* Break the \( (ab : A \land B) \) hypothesis *)
  
  \[
  \begin{align*}
  ab : A \land B & \quad \rightarrow \quad \frac{G}{A \rightarrow B \rightarrow G}
  \end{align*}
  \]
  - **split**. (* Prove a conjunction: \( A \land B \) *)
Propositional Logic, Conjunction

- **Conjunction**: $A \land B$
  - **case**: ab. (* Break the $(ab : A \land B)$ hypothesis *)

  \[
  \frac{ab : A \land B}{G} \quad \rightarrow \quad \frac{A \rightarrow B \rightarrow G}{A \land B}
  \]

- **split.** (* Prove a conjunction : $A \land B$ *)

  \[
  \frac{A \land B}{A} \quad \rightarrow \quad \frac{B}{A \land B}
  \]
Propositional Logic, Disjunction

- Disjunction: \( A \lor B \)
Propositional Logic, Disjunction

- Disjunction: \( A \lor B \)
  - case: ab. (* Break the \((ab : A \lor B)\) hypothesis *)
**Propositional Logic, Disjunction**

- **Disjunction**: $A \lor B$

  - **case**: $ab$. (* Break the $(ab : A \lor B)$ hypothesis *)

\[
\frac{ab : A \lor B}{G} \quad \rightarrow \quad \frac{A \rightarrow G}{A} \quad \frac{B \rightarrow G}{B}
\]
Propositional Logic, Disjunction

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

\[
\begin{align*}
\text{ab} & : A \lor B \\
\hline
\text{G} & \rightarrow \hspace{1cm} A \rightarrow \text{G} \hspace{1cm} B \rightarrow \text{G}
\end{align*}
\]

- **left.** (* Prove a disjunction :\( A \lor B \) *)
  (* by choosing the left part *)
Propositional Logic, Disjunction

- **Disjunction**: $A \lor B$

  - **case**: $ab$. (* Break the $(ab : A \lor B)$ hypothesis *)

  \[
  \begin{array}{c}
  ab : A \lor B \\
  \hline
  G \\
  \end{array} \quad \rightarrow \quad 
  \begin{array}{c}
  A \rightarrow G \\
  B \rightarrow G \\
  \end{array}
  \]

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

  \[
  \begin{array}{c}
  A \lor B \\
  \hline
  A \\
  \end{array} \quad \rightarrow \quad 
  A
  \]
Propositional Logic, Disjunction

Disjunction: $A \lor B$

- **case**: $ab$. (* Break the $(ab : A \lor B)$ hypothesis *)

\[
\begin{align*}
ab : A \lor B & \quad \Rightarrow \quad A \to G \\
& \quad \Rightarrow \quad B \to G \\
\end{align*}
\]

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

\[
\begin{align*}
A \lor B & \quad \Rightarrow \quad A \\
\end{align*}
\]

- **right.** (* Prove a disjunction: $A \lor B$ *)
  
  (* by choosing the right part *)
Propositional Logic, Negation

- Negation: \( \neg B \)
Propositional Logic, Negation

- Negation: \( \neg B \)
  - \( \neg B \) is defined as (\( B \rightarrow \text{False} \))
Negation : \( \sim B \)

\( \sim B \) is defined as \( (B \rightarrow \text{False}) \)

\( \text{move=} \rightarrow B. \) (* To prove the goal \( (\sim B)\)*)
Negation: \( \neg B \)
- \( \neg B \) is defined as (\( B \rightarrow \text{False} \))
- \text{move=}\( B \). \((* \text{ To prove the goal } (\neg B)*)\)

\[
\begin{array}{c}
\ldots \\
\vdots \\
\neg B \\
\rightarrow \\
b: B \\
\frac{}{\text{False}}
\end{array}
\]
Negation : \( \neg B \)

- \( \neg B \) is defined as \((B \rightarrow \text{False})\)
- move=> B. (* To prove the goal (\( \neg B \))*

\[
\begin{align*}
\therefore \quad \neg B & \rightarrow b : B \\
\hline
\neg B & \rightarrow \text{False}
\end{align*}
\]

- Then apply: H. (* for a (H : \( \neg C \)) in the context*)
Existential Quantifier

Existential: \( \text{exists } n: \text{nat}, \ P \ n \)

(* \( P \) is a predicate on \( \text{nat} \) (\( P : \text{nat} \implies \text{Prop} \))*
Existential Quantifier

Existential: \textit{exists} n: \textit{nat}, P n

(* P is a predicate on nat (P : nat \rightarrow Prop)*)

- \textit{exists} 2. (*To prove an exists, give a witness *)
Existential Quantifier

Existential: $\exists n : \text{nat}, P n$

(* $P$ is a predicate on nat ($P : \text{nat} \rightarrow \text{Prop}$)*

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

\[
\begin{align*}
\cdots & \\
\exists n : \text{nat}, P n & \rightarrow \cdots \\
\rightarrow & \ P 2
\end{align*}
\]
Existential Quantifier

**Existential:** \( \exists n : \text{nat}, \ P n \)

(*\( P \) is a predicate on \( \text{nat} \) (\( P : \text{nat} \rightarrow \text{Prop} \))*

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

\[
\begin{align*}
\ldots & \quad \ldots \\
\exists n : \text{nat}, \ P n & \rightarrow \ P 2
\end{align*}
\]

- **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 *)`

    ...  
    \[ \begin{array}{c} \exists n : \text{nat}, P n \\ \rightarrow \end{array} \]  
    ...  
    \[ \exists n : \text{nat}, P n \rightarrow P 2 \]

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

    \[ \begin{array}{c} \text{Hex: exists } n, P n \\ \rightarrow \end{array} \]  
    \[ \begin{array}{c} n : \text{nat} \\ \rightarrow \end{array} \]  
    \[ \begin{array}{c} Hn : P n \\ \rightarrow \end{array} \]  
    \[ G \]  
    \[ G \]
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**: "&&", "||", "~~", "==>", "(+)".

<table>
<thead>
<tr>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_1 &amp;&amp; b_2$</th>
<th>$b_1 \mid\mid b_2$</th>
<th>$b_1 \implies b_2$</th>
<th>$b_1 (+) b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>
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)
  
  :
  
  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)

  ...

  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:
  
  \[ \text{rewrite} /=. \]

  
  \[ b : \text{bool} \]

  \[ \text{true} (+)b = (\text{true} \&\& \sim b) \| (\sim \text{true} \&\& b) \]

  
  \[ b : \text{bool} \]

  \[ \sim \sim b = \sim b \| \text{false} \]
Outline

1 Logics
   - First Order Logic
   - Booleans

2 Tactics, Tacticals

3 Proof Structure
   - Forward reasoning
   - Proof control flow
   - Subgoal selectors
**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
- 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.
"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.

\[
\begin{align*}
\text{H: } P & \rightarrow Q \\
Q & \rightarrow P
\end{align*}
\]
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.

\[
\begin{align*}
\text{H: } P & \rightarrow \ Q \\
\text{\quad \quad Q} & \rightarrow \ Q \\
\quad \quad P & \rightarrow \ P
\end{align*}
\]

- "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 := 0 | 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 := 0 | S of nat`

**Lemma** `P_of_n` *forall* `n : nat`, `P n`.

`move => n`.

`case : n`.

1. \( P \ 0 \)
2. *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. \[ P 0 \]
2. \[ \text{forall } n, P n \rightarrow P (S n) \]
Basic Rewriting tactic

Tactic "\textit{rewrite items...}" modifies subterms of the goal:

- "\texttt{/name}" unfolds a definition
- "\texttt{-/name}" folds a definition
- "\textit{term}" rewrites (left to right) with a lemma or an hypothesis which conclusion is an equality
Tactic "\texttt{rewrite items}..." modifies subterms of the goal:

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

\[
\begin{align*}
\text{Eqab: } a & = b \\
\frac{P \ a}{P \ b}
\end{align*}
\]
Basic Rewriting tactic

Tactic "\texttt{rewrite items} . . ." modifies subterms of the goal:

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

\[
\begin{align*}
\text{Eqab: } & a = b \quad \Rightarrow \quad \text{Eqab: } a = b \\
\frac{\text{P } a}{\text{P } b}
\end{align*}
\]

- "-/\texttt{term}" rewrites right to left
Multiple Rewriting and Occurrence selection

\texttt{rewrite \textasciitilde 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

\texttt{rewrite \textasciitilde 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.

\texttt{rewrite \textasciitilde \{number\}\_term}
Multiple Rewriting and Occurrence selection

\[\text{rewrite } -\text{multiplicity term}\]

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

\[\text{rewrite } -\{\text{number}\} \text{ term}\]

\textbf{Lemma} \(\text{dbl a b : } 2 \ast (a + b) = (b + a) + (a + b)\).
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 \(-\{\text{number}\}\)term

**Lemma** dbl a b : $2 \times (a + b) = (b + a) + (a + b)$.

**Proof**.

\[
\begin{align*}
a : \text{nat} \\
b : \text{nat} \\
2 \times (a + b) &= (b + a) + (a + b)
\end{align*}
\]
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 \(-\{\text{number}\}\ term\)

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

rewrite \(!\ addnA\).

\[
\begin{align*}
a & : \text{nat} \\
b & : \text{nat} \\
2 * (a + b) &= b + (a + (a + b))
\end{align*}
\]
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.

\[\begin{align*}
a & : \text{nat} \\
b & : \text{nat} \\
2 \(*\) (a + b) = (b + a) + (b + a)
\end{align*}\]
Multiple Rewriting and Occurrence selection

```
rewrite \textit{-multiplicity} \textit{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 \textit{-\{number\}} \textit{term}

Lemma \textit{dbl a b : } 2 \cdot (a + b) = (b + a) + (a + b).
```

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

\begin{align*}
a : \text{nat} \\
\qquad b : \text{nat} \\
\begin{array}{l}
2 \cdot (a + b) = (b + (a + (b + a))
\end{array}
\end{align*}
Outline

1. Logics

2. Tactics, Tacticals

3. Proof Structure
   - Forward reasoning
   - Proof control flow
   - Subgoal selectors
Forward Reasoning

- have
- suffices (suff)
Forward Reasoning: \texttt{have / suffices}

\texttt{have H : intermediate\_goal} performs a \texttt{logical cut}.
Forward Reasoning: have / suffices

```ocaml
have H : intermediate_goal
performs a logical cut.

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

```
Forward Reasoning: have / suffices

have H : \textit{intermediate}\_goal
performs a logical cut.

Variable f : nat \rightarrow nat.
Variable P : nat \rightarrow \text{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.


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

1. exists x, f x = 3
2. H: exists x, f x = 3
   P 3
Forward Reasoning: have / suffices

have H : intermediate_goal
performs a logical cut.

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


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

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_mul1.
  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 ^ 1)); congr (_ * _).
  by apply/eqP; rewrite -dvdn1.
+ by move: P1; case: (t) => [ | [ | s1]].
- rewrite muln0 gexpn0 mulg => H1.
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)