Automating Natural Deduction 2

Simon Foster   Jim Woodcock
University of York

18th August 2022

Overview

1. Universal and Existential Quantifiers
2. Fixed and Schematic Variables
3. Deduction Rules
4. Isar Proofs with Quantifiers

Outline

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall x.\ \exists y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard
- We often need to guide a proof
- Even powerful tools like `blast` and `auto` often stop at quantifiers
- They usually highlight the more creative aspects needed for a proof
- Particularly important for inductive and set theoretic proofs
- First, we need to consider the role of logical variables in proof

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall x.\ \exists y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Proof with Quantifiers

- Universal and existential quantifiers:

$$\forall\, x.\ \exists\, y.\ P(x, y)$$

- Reasoning with quantifiers is intrinsically hard.
- We often need to guide a proof.
- Even powerful tools like `blast` and `auto` often stop at quantifiers.
- They usually highlight the more creative aspects needed for a proof.
- Particularly important for inductive and set theoretic proofs.
- First, we need to consider the role of logical variables in proof.

Outline

Free and Fixed Variables in Theorem Specifications

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x.P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.
- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.

Example

- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.

Example

```
lemma nat_min: "⟦ ⋀ x::nat. y ≤ x ⟧ ⟹ y = 0"
```

- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Free and Fixed Variables in Theorem Specifications

Example

```
theorem square_greater_zero:
  fixes x :: nat assumes "x > 0" shows "square x > 0"
```

- Make them arbitrary but fixed during the proof.
- All we know about fixed variables is their type and assumptions.
- Analogy: Fixed variables and assumptions inputs. Conclusion output.
- Captured by meta-quantification in the proof state.
- $\bigwedge x. P(x)$ means P is valid for any value x.

Example

```
lemma nat_min: "⟦ ⋀ x::nat. y ≤ x ⟧ ⟹ y = 0"
```

- During proof x (bound) can take an arbitary value, but y (free) is fixed.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.
- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"] = (0 < 3 ⟹ 0 < square 3)`.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.
- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"]` = $(0 < 3 \Longrightarrow 0 <$ `square 3`$)$.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.
- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"] = (0 < 3 ⟹ 0 < square 3)`.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

- We can instantiate a schematic variable x manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"]` = $(0 < 3 \Longrightarrow 0 < \text{square } 3)$.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

square_greater_zero: $0 < ?x \Longrightarrow 0 < \text{square } ?x$

nat_min: $\bigwedge x.\ ?y \leq x \Longrightarrow ?y = 0$

- We can instantiate a schematic variable x manually in several ways:
- thmname[where x="val"] and (rule_tac x="val" in thmname).
- square_greater_zero[where x="3"] = ($0 < 3 \Longrightarrow 0 <$ square 3).
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

`square_greater_zero: 0 < ?x` $\Longrightarrow$ `0 < square ?x`

`nat_min:` $\bigwedge$ `x. ?y` $\leq$ `x` $\Longrightarrow$ `?y = 0`

- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"]` = `(0 < 3` $\Longrightarrow$ `0 < square 3)`.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

square_greater_zero: 0 < ?x $\Longrightarrow$ 0 < square ?x
nat_min: $\bigwedge$ x. ?y $\leq$ x $\Longrightarrow$?y = 0

- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"]` = (0 < 3 $\Longrightarrow$ 0 < square 3).
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

`square_greater_zero:` $0 < ?x \Longrightarrow 0 < \texttt{square}\ ?x$

`nat_min:` $\bigwedge x.\ ?y \leq x \Longrightarrow ?y = 0$

- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"]` $= (0 < 3 \Longrightarrow 0 < \texttt{square}\ 3)$.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

`square_greater_zero: 0 < ?x ⟹ 0 < square ?x`

`nat_min: ⋀ x. ?y ≤ x ⟹ ?y = 0`

- We can instantiate a schematic variable `x` manually in several ways:
- `thmname[where x="val"]` and `(rule_tac x="val" in thmname)`.
- `square_greater_zero[where x="3"] = (0 < 3 ⟹ 0 < square 3)`.
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Schematic Variables

- $?x$: schematic variable ("unknown") in term that can be instantiated.
- When a theorem is proved, each free variable becomes a schematic.

Example

square_greater_zero: 0 < ?x $\Longrightarrow$ 0 < square ?x
nat_min: $\bigwedge$ x. ?y $\leq$ x $\Longrightarrow$?y = 0

- We can instantiate a schematic variable x manually in several ways:
- thmname[where x="val"] and (rule_tac x="val" in thmname).
- square_greater_zero[where x="3"] = (0 < 3 $\Longrightarrow$ 0 < square 3).
- Schematic variables can be shared among subgoals in the proof state.
- Fixed and schematic variables important for quantifier deduction rules.

Outline

Quantifier Deduction Rules

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$x \notin \mathit{fv}(\Gamma)\ \frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall\, a.\, P(a)}\ \texttt{allI} \qquad \frac{P(t), \Gamma \vdash R}{\forall\, a.\, P(a), \Gamma \vdash R}\ \texttt{allE}$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists\, a.\, P(a)}\ \texttt{exI} \qquad x \notin \mathit{fv}(\Gamma, Q)\ \frac{P(x), \Gamma \vdash Q}{\exists\, a.\, P(a), \Gamma \vdash Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$x \notin \mathit{fv}(\Gamma)\ \frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall\, a.\, P(a)}\ \texttt{allI} \qquad \frac{P(t), \Gamma \vdash R}{\forall\, a.\, P(a), \Gamma \vdash R}\ \texttt{allE}$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists\, a.\, P(a)}\ \texttt{exI} \qquad x \notin \mathit{fv}(\Gamma, Q)\ \frac{P(x), \Gamma \vdash Q}{\exists\, a.\, P(a), \Gamma \vdash Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$x \notin fv(\Gamma)\ \frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall\, a.\, P(a)}\ \texttt{allI} \qquad \frac{P(t), \Gamma \vdash R}{\forall\, a.\, P(a), \Gamma \vdash R}\ \texttt{allE}$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists\, a.\, P(a)}\ \texttt{exI} \qquad x \notin fv(\Gamma, Q)\ \frac{P(x), \Gamma \vdash Q}{\exists\, a.\, P(a), \Gamma \vdash Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$x \notin fv(\Gamma)\ \frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall\, a.\, P(a)}\ \texttt{allI} \qquad \frac{P(t), \Gamma \vdash R}{\forall\, a.\, P(a), \Gamma \vdash R}\ \texttt{allE}$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists\, a.\, P(a)}\ \texttt{exI} \qquad x \notin fv(\Gamma, Q)\ \frac{P(x), \Gamma \vdash Q}{\exists\, a.\, P(a), \Gamma \vdash Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$x \notin \mathit{fv}(\Gamma)\ \frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall\, a.\, P(a)}\ \texttt{allI} \qquad \frac{P(t), \Gamma \vdash R}{\forall\, a.\, P(a), \Gamma \vdash R}\ \texttt{allE}$$

$$\frac{\Gamma \vdash P(t)}{\Gamma \vdash \exists\, a.\, P(a)}\ \texttt{exI} \qquad x \notin \mathit{fv}(\Gamma, Q)\ \frac{P(x), \Gamma \vdash Q}{\exists\, a.\, P(a), \Gamma \vdash Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\, x}{\forall a.\, P\, a}\ \texttt{allI} \qquad \frac{\forall a.\, P\, a \qquad P\, t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall a.\, P\, a}{P\, t}\ \texttt{spec}$$

$$\frac{P\, t}{\exists a.\, P\, a}\ \texttt{exI} \qquad \frac{\exists a.\, P\, a \qquad \bigwedge x.\, P\, x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall a.\ P\ a}\ \texttt{allI} \qquad \frac{\forall a.\ P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall a.\ P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists a.\ P\ a}\ \texttt{exI} \qquad \frac{\exists a.\ P\ a \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\, x}{\forall\, a.\, P\, a}\ \texttt{allI} \qquad \frac{\forall\, a.\, P\, a \qquad P\, t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\, a.\, P\, a}{P\, t}\ \texttt{spec}$$

$$\frac{P\, t}{\exists\, a.\, P\, a}\ \texttt{exI} \qquad \frac{\exists\, a.\, P\, a \qquad \bigwedge x.\, P\, x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\ x}{\forall\, a.\, P\ a}\ \texttt{allI} \qquad \frac{\forall\, a.\, P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\, a.\, P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists\, a.\, P\ a}\ \texttt{exI} \qquad \frac{\exists\, a.\, P\ a \qquad \bigwedge x.\, P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall\ a.\ P\ a}\ \texttt{allI} \qquad \frac{\forall\ a.\ P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\ a.\ P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists\ a.\ P\ a}\ \texttt{exI} \qquad \frac{\exists\ a.\ P\ a \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall a.\ P\ a}\ \texttt{allI} \qquad \frac{\forall a.\ P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall a.\ P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists a.\ P\ a}\ \texttt{exI} \qquad \frac{\exists a.\ P\ a \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall\ a.\ P\ a}\ \texttt{allI} \qquad \frac{\forall\ a.\ P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\ a.\ P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists\ a.\ P\ a}\ \texttt{exI} \qquad \frac{\exists\ a.\ P\ a \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall a.\ P\ a}\ \texttt{allI} \qquad \frac{\forall a.\ P\ a \qquad P\ t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall a.\ P\ a}{P\ t}\ \texttt{spec}$$

$$\frac{P\ t}{\exists a.\ P\ a}\ \texttt{exI} \qquad \frac{\exists a.\ P\ a \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\, x}{\forall\, a.\, P\, a}\ \texttt{allI} \qquad \frac{\forall\, a.\, P\, a \qquad P\, t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\, a.\, P\, a}{P\, t}\ \texttt{spec}$$

$$\frac{P\, t}{\exists\, a.\, P\, a}\ \texttt{exI} \qquad \frac{\exists\, a.\, P\, a \qquad \bigwedge x.\, P\, x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\, x}{\forall\, a.\, P\, a}\ \mathtt{allI} \qquad \frac{\forall\, a.\, P\, a \qquad P\, t \Longrightarrow R}{R}\ \mathtt{allE} \qquad \frac{\forall\, a.\, P\, a}{P\, t}\ \mathtt{spec}$$

$$\frac{P\, t}{\exists\, a.\, P\, a}\ \mathtt{exI} \qquad \frac{\exists\, a.\, P\, a \qquad \bigwedge x.\, P\, x \Longrightarrow Q}{Q}\ \mathtt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Quantifier Deduction Rules

Quantifiers

$$\frac{\bigwedge x.\, P\, x}{\forall\, a.\, P\, a}\ \texttt{allI} \qquad \frac{\forall\, a.\, P\, a \qquad P\, t \Longrightarrow R}{R}\ \texttt{allE} \qquad \frac{\forall\, a.\, P\, a}{P\, t}\ \texttt{spec}$$

$$\frac{P\, t}{\exists\, a.\, P\, a}\ \texttt{exI} \qquad \frac{\exists\, a.\, P\, a \qquad \bigwedge x.\, P\, x \Longrightarrow Q}{Q}\ \texttt{exE}$$

- Note that P is a polymorphic predicate of type `'a ⇒ bool`.
- For `allI`, demonstrate $P\ x$ for an arbitrary but fixed value x.
- For `exI`, demonstrate $P\ x$ for a particular value t (that we supply).
- For `allE`, assume $P\ x$ holds for a particular value t.
- For `exE`, assume $P\ x$ holds for an arbitrary but fixed value x.

Outline

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.
  - The quantifier for y isn't necessary—free variables are universal.

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{}{y \in \mathbb{N} \vdash y + 1 > y}\ \text{arithmetic}}{y \in \mathbb{N} \vdash \exists a \in \mathbb{N}.\ a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall b \in \mathbb{N}.\ \exists a \in \mathbb{N}.\ a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

arithmetic

$y \in \mathbb{N} \vdash y + 1 > y$

exI, $a = y + 1$

$y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y$

allI

$\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{}{y \in \mathbb{N} \vdash y + 1 > y}\text{arithmetic}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\,\texttt{exI},\ a = y+1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\,\texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{}{y \in \mathbb{N} \vdash y + 1 > y}\ \text{arithmetic}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\overset{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\ \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

arithmetic

$$\dfrac{\dfrac{y \in \mathbb{N} \vdash y + 1 > y}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$
\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}\ \texttt{exI},\ a = y + 1}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{allI}}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}
$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

```
lemma "∃ x::nat. x > y"
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

```
lemma "∃ x::nat. x > y"
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\text{arithmetic}}{y \in \mathbb{N} \vdash y + 1 > y}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\;\texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\;\texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

```
lemma "∃ x::nat. x > y"
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

Example: Supplying a Witness (Explicit)

- Aim is to prove $\forall\, y :: nat.\, \exists\, x.\, x > y$.

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{}{y \in \mathbb{N} \vdash y + 1 > y}\,\text{arithmetic}}{y \in \mathbb{N} \vdash \exists\, a \in \mathbb{N}.\, a > y}\ \texttt{exI},\ a = y + 1}{\vdash \forall\, b \in \mathbb{N}.\, \exists\, a \in \mathbb{N}.\, a > b}\ \texttt{allI}$$

Isabelle Proof

```
lemma "∀ b::nat. ∃ a. a > b"
  apply (rule allI)
  apply (rename_tac y)
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

- The quantifier for y isn't necessary—free variables are universal:

```
lemma "∃ x::nat. x > y"
  apply (rule_tac x="y+1" in exI)
  apply (fact less_add_one)
  done
```

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.
- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Supplying a Witness (Implicit)

- Invoking `exI` without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation

Example: Supplying a Witness (Implicit)

- Invoking exI without a witness leads to creation of a schematic variable.

```
lemma "∃ x::nat. 0 < x ∧ x < 10"
proof (rule exI) (* Goal: 0 < ?x ∧ ?x < 10 *)
  let ?v = "5::nat"
  (* Supply witness and specialise goal *)
  show "0 < ?v ∧ ?v < 10"
  proof (rule conjI)
    show "0 < ?v" by simp
    show "?v < 10" by simp
  qed
qed
```

- Schematic variables shared by subgoals: simultaneous instantiation.

Example: Fixed Variables in Isar

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\,\texttt{impI}}{\vdash \forall n.\; n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\,\texttt{allI}$$

Isar Proof

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\vdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\,\texttt{impI}}{\vdash \forall\, n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\,\texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall\, n.\ n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\cfrac{\cfrac{\begin{array}{c}\cdots\\ x \bmod 2 \neq 0 \vdash x \bmod 2 = 1\end{array}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall\, n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\vdots$$

$$\dfrac{\dfrac{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall\, n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\begin{array}{c}\cdots \\ x \bmod 2 \neq 0 \vdash x \bmod 2 = 1\end{array}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall n.\ n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\;\texttt{impI}}{\vdash \forall n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\;\texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\,\texttt{impI}}{\vdash \forall n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\,\texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall n.\ n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Fixed Variables in Isar

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\cdots}{x \bmod 2 \neq 0 \vdash x \bmod 2 = 1}}{\vdash x \bmod 2 \neq 0 \longrightarrow x \bmod 2 = 1}\ \texttt{impI}}{\vdash \forall n.\, n \bmod 2 \neq 0 \longrightarrow n \bmod 2 = 1}\ \texttt{allI}$$

Isar Proof

```
lemma "∀ n::int. n mod 2 ≠ 0 ⟶ n mod 2 = 1"
proof (rule allI, rule impI)
  fix x::int
  assume "x mod 2 ≠ 0"
  thus "x mod 2 = 1"
    by (simp only: not_mod_2_eq_0_eq_1)
qed
```

Example: Syllogism in Isar

Example: Syllogism in Isar

Example

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Example: Syllogism in Isar

Example

```
lemma Socrates_syllogism:
  assumes "∀ x. man(x) ⟶ mortal(x)" "man(Socrates)"
  shows "mortal(Socrates)"
proof -
  (* Once proved, f contains a schematic variable *)
  from assms(1) have f: "⋀ x. man(x) ⟹ mortal(x)"
    by simp
  (* Instantiate fact f *)
  have "man(Socrates) ⟹ mortal(Socrates)"
    by (rule f[where x="Socrates"])
  with assms(2) show "mortal(Socrates)"
    by simp
qed
```

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\, P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\ P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\ P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\ P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. $\mathtt{pn}$) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\, P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule $\mathtt{exE}$.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\ P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\, P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Existential Elimination in Isar Proofs

- We can introduce existential properties from assumptions.

```
obtain x :: T where pn: "P x"...
```

- Creates a fixed local variable x that satisfies P
- Provided some such x exists.
- The properties of x (e.g. `pn`) become available as local assumptions.
- Requires that we prove $(\bigwedge n.\ P\ n \Longrightarrow Q) \Longrightarrow Q$ for arbitrary Q.
- This corresponds to the existential elimination rule `exE`.

Obtain Example: Even Numbers

Natural Deduction Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{x \bmod 2 = 0, x = 2 * y \vdash x = 2 * y}\,\texttt{asm}}{x \bmod 2 = 0, x = 2 * y \vdash (\exists\, b.\, x = 2 * b)}\,\texttt{exI}}{x \bmod 2 = 0 \vdash (\exists\, b.\, x = 2 * b)}\,\texttt{exE}}{\vdash \forall\, a.\, a \bmod 2 = 0 \longrightarrow (\exists\, b.\, a = 2 * b)}\,\texttt{allI}, \texttt{impI}$$

```
lemma "∀ a::nat. a mod 2 = 0 ⟶ (∃ b. a = 2 * b)"
proof (rule allI, rule impI)
  fix x :: nat
  assume "x mod 2 = 0"
  then obtain y where "x = 2 * y" using mod_eq_0D by blast
  thus "∃ b. x = 2 * b"
    by (rule_tac x="y" in exI, assumption)
qed
```

Obtain Example: No greatest natural number

- `notI: (?P` $\Longrightarrow$ `False)` $\Longrightarrow$ $\neg$ `?P`.
- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- `notI: (?P ⟹ False) ⟹ ¬ ?P`.
- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P ⟹ False) ⟹ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P ⟹ False) ⟹ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P ⟹ False) ⟹ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P ⟹ False) ⟹ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P ⟹ False) ⟹ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ $\neg$?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Obtain Example: No greatest natural number

- notI: (?P $\Longrightarrow$ False) $\Longrightarrow$ ¬ ?P.

```
lemma no_ge_nat: "∄ y::nat. ∀ x. y ≥ x"
proof (rule notI)
  assume "∃ y::nat. ∀x. y ≥ x"
  (* Obtain a number n greater than any number x *)
  then obtain n::nat where n: "∀ x. n ≥ x" by auto
  (* Prove that Suc n is greater than n *)
  have 1: "Suc n > n" by (fact Nat.lessI)
  (* Prove n is greater than or equal to Suc n *)
  from n have 2: "n ≥ Suc n" by auto
  (* Contradiction *)
  from 1 2 show False by auto
qed
```

- Facts 1 and 2 are contradictory, so no such y can exist.

Example: One Point Rule

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Example: One Point Rule

```
lemma one_point: "(∃ x. x = v ∧ P x) ⟷ P v"
proof (rule iffI)
  assume "(∃ x. x = v ∧ P x)" thus "P v"
    by (erule_tac exE, simp)
next
  assume a: "P v" show "(∃ x. x = v ∧ P x)"
  (* Subgoals: ?x = v and P ?x *)
  proof (rule_tac exI, rule_tac conjI)
    (* This specialises ?x = v *)
    show "v = v" by (rule refl)
  next
    show "P v" by (rule a)
  qed
qed
```

Conclusion

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Next Lecture

- Sets and types.

Conclusion

This Lecture

- Fixed and schematic variables.
- Natural deduction rules for quantifiers.
- Reasoning with universal and existential quantifiers.

Next Lecture

- Sets and types.