



NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications



CONTENT

- Intro & motivation, getting started with Isabelle
- **Foundations & Principles**
 - Lambda Calculus
 - Higher Order Logic, natural deduction
 - **Term rewriting**
- Proof & Specification Techniques
 - Inductively defined sets, rule induction
 - Datatypes, recursion, induction
 - Calculational reasoning, mathematics style proofs
 - Hoare logic, proofs about programs

LAST TIME

→ Introducing new Types

LAST TIME

- Introducing new Types
- Equations and Term Rewriting

LAST TIME

- Introducing new Types
- Equations and Term Rewriting
- Confluence and Termination of reduction systems

LAST TIME

- Introducing new Types
- Equations and Term Rewriting
- Confluence and Termination of reduction systems
- Term Rewriting in Isabelle

LAST TIME

- Introducing new Types
- Equations and Term Rewriting
- Confluence and Termination of reduction systems
- Term Rewriting in Isabelle
- First structured proofs (Isar)

APPLYING A REWRITE RULE

→ $l \longrightarrow r$ **applicable** to term $t[s]$

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$
- **Result:** $t[\sigma r]$

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$
- **Result:** $t[\sigma r]$
- **Equationally:** $t[s] = t[\sigma r]$

Example:

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$
- **Result:** $t[\sigma r]$
- **Equationally:** $t[s] = t[\sigma r]$

Example:

Rule: $0 + n \longrightarrow n$

Term: $a + (0 + (b + c))$

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$
- **Result:** $t[\sigma r]$
- **Equationally:** $t[s] = t[\sigma r]$

Example:

Rule: $0 + n \longrightarrow n$

Term: $a + (0 + (b + c))$

Substitution: $\sigma = \{n \mapsto b + c\}$

APPLYING A REWRITE RULE

- $l \longrightarrow r$ **applicable** to term $t[s]$
if there is substitution σ such that $\sigma l = s$
- **Result:** $t[\sigma r]$
- **Equationally:** $t[s] = t[\sigma r]$

Example:

Rule: $0 + n \longrightarrow n$

Term: $a + (0 + (b + c))$

Substitution: $\sigma = \{n \mapsto b + c\}$

Result: $a + (b + c)$

CONDITIONAL TERM REWRITING

Rewrite rules can be conditional:

$$\llbracket P_1 \dots P_n \rrbracket \Longrightarrow l = r$$

CONDITIONAL TERM REWRITING

Rewrite rules can be conditional:

$$\llbracket P_1 \dots P_n \rrbracket \Longrightarrow l = r$$

is **applicable** to term $t[s]$ with σ if

- $\sigma l = s$ and
- $\sigma P_1, \dots, \sigma P_n$ are provable by rewriting.

REWRITING WITH ASSUMPTIONS

Last time: Isabelle uses assumptions in rewriting.

REWRITING WITH ASSUMPTIONS

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

lemma " $f\ x = g\ x \wedge g\ x = f\ x \implies f\ x = 2$ "

REWRITING WITH ASSUMPTIONS

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

lemma " $f\ x = g\ x \wedge g\ x = f\ x \implies f\ x = 2$ "

simp	use and simplify assumptions
(simp (no_asm))	ignore assumptions
(simp (no_asm_use))	simplify , but do not use assumptions
(simp (no_asm_simp))	use , but do not simplify assumptions

PREPROCESSING

Preprocessing (recursive) for maximal simplification power:

$$\begin{aligned}\neg A &\mapsto A = \textit{False} \\ A \longrightarrow B &\mapsto A \implies B \\ A \wedge B &\mapsto A, B \\ \forall x. A x &\mapsto A ?x \\ A &\mapsto A = \textit{True}\end{aligned}$$

PREPROCESSING

Preprocessing (recursive) for maximal simplification power:

$$\neg A \mapsto A = \textit{False}$$

$$A \longrightarrow B \mapsto A \implies B$$

$$A \wedge B \mapsto A, B$$

$$\forall x. A x \mapsto A ?x$$

$$A \mapsto A = \textit{True}$$

Example:

$$(p \longrightarrow q \wedge \neg r) \wedge s$$

\mapsto

PREPROCESSING

Preprocessing (recursive) for maximal simplification power:

$$\begin{aligned}\neg A &\mapsto A = \textit{False} \\ A \longrightarrow B &\mapsto A \implies B \\ A \wedge B &\mapsto A, B \\ \forall x. A x &\mapsto A ?x \\ A &\mapsto A = \textit{True}\end{aligned}$$

Example:

$$\begin{aligned}(p \longrightarrow q \wedge \neg r) \wedge s \\ \mapsto \\ p \implies q = \textit{True} \quad r = \textit{False} \quad s = \textit{True}\end{aligned}$$

DEMO

CASE SPLITTING WITH SIMP

$$\begin{aligned} &P \text{ (if } A \text{ then } s \text{ else } t) \\ &= \\ &(A \longrightarrow P s) \wedge (\neg A \longrightarrow P t) \end{aligned}$$

CASE SPLITTING WITH SIMP

$$P \text{ (if } A \text{ then } s \text{ else } t)$$
$$=$$
$$(A \longrightarrow P s) \wedge (\neg A \longrightarrow P t)$$

Automatic

CASE SPLITTING WITH SIMP

$$\begin{aligned} &P (\text{if } A \text{ then } s \text{ else } t) \\ &= \\ &(A \longrightarrow P s) \wedge (\neg A \longrightarrow P t) \end{aligned}$$

Automatic

$$\begin{aligned} &P (\text{case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) \\ &= \\ &(e = 0 \longrightarrow P a) \wedge (\forall n. e = \text{Suc } n \longrightarrow P b) \end{aligned}$$

CASE SPLITTING WITH SIMP

$$\begin{aligned} &P \text{ (if } A \text{ then } s \text{ else } t) \\ &= \\ &(A \longrightarrow P s) \wedge (\neg A \longrightarrow P t) \end{aligned}$$

Automatic

$$\begin{aligned} &P \text{ (case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) \\ &= \\ &(e = 0 \longrightarrow P a) \wedge (\forall n. e = \text{Suc } n \longrightarrow P b) \end{aligned}$$

Manually: apply (simp split: nat.split)

CASE SPLITTING WITH SIMP

$$\begin{aligned} P (\text{if } A \text{ then } s \text{ else } t) \\ = \\ (A \longrightarrow P s) \wedge (\neg A \longrightarrow P t) \end{aligned}$$

Automatic

$$\begin{aligned} P (\text{case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) \\ = \\ (e = 0 \longrightarrow P a) \wedge (\forall n. e = \text{Suc } n \longrightarrow P b) \end{aligned}$$

Manually: apply (simp split: nat.split)

Similar for any data type **t**: **t.split**

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

For \Longrightarrow hardwired (assumptions used in rewriting)

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

For \Longrightarrow hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Read: to simplify $P \longrightarrow Q$

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

For \Longrightarrow hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Read: to simplify $P \longrightarrow Q$

→ first simplify P to P'

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

For \Longrightarrow hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Read: to simplify $P \longrightarrow Q$

- first simplify P to P'
- then simplify Q to Q' using P' as assumption

CONGRUENCE RULES

congruence rules are about using context

Example: in $P \longrightarrow Q$ we could use P to simplify terms in Q

For \Longrightarrow hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Read: to simplify $P \longrightarrow Q$

- first simplify P to P'
- then simplify Q to Q' using P' as assumption
- the result is $P' \longrightarrow Q'$

MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

conj_cong: $\llbracket P = P'; P' \implies Q = Q' \rrbracket \implies (P \wedge Q) = (P' \wedge Q')$

MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

conj_cong: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \wedge Q) = (P' \wedge Q')$

Context for if-then-else:

if_cong: $\llbracket b = c; c \Longrightarrow x = u; \neg c \Longrightarrow y = v \rrbracket \Longrightarrow$
 $(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$

MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

conj_cong: $\llbracket P = P'; P' \Longrightarrow Q = Q' \rrbracket \Longrightarrow (P \wedge Q) = (P' \wedge Q')$

Context for if-then-else:

if_cong: $\llbracket b = c; c \Longrightarrow x = u; \neg c \Longrightarrow y = v \rrbracket \Longrightarrow$
 $(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$

Prevent rewriting inside then-else (default):

if_weak_cong: $b = c \Longrightarrow (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)$

MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

conj_cong: $\llbracket P = P'; P' \implies Q = Q' \rrbracket \implies (P \wedge Q) = (P' \wedge Q')$

Context for if-then-else:

if_cong: $\llbracket b = c; c \implies x = u; \neg c \implies y = v \rrbracket \implies$
 $(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$

Prevent rewriting inside then-else (default):

if_weak_cong: $b = c \implies (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)$

→ declare own congruence rules with **[cong]** attribute

MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

conj_cong: $\llbracket P = P'; P' \implies Q = Q' \rrbracket \implies (P \wedge Q) = (P' \wedge Q')$

Context for if-then-else:

if_cong: $\llbracket b = c; c \implies x = u; \neg c \implies y = v \rrbracket \implies$
 $(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)$

Prevent rewriting inside then-else (default):

if_weak_cong: $b = c \implies (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)$

- declare own congruence rules with **[cong]** attribute
- delete with **[cong del]**

ORDERED REWRITING

Problem: $x + y \longrightarrow y + x$ does not terminate

ORDERED REWRITING

Problem: $x + y \longrightarrow y + x$ does not terminate

Solution: use permutative rules only if term becomes lexicographically smaller.

Example:

ORDERED REWRITING

Problem: $x + y \longrightarrow y + x$ does not terminate

Solution: use permutative rules only if term becomes lexicographically smaller.

Example: $b + a \rightsquigarrow a + b$ but not $a + b \rightsquigarrow b + a$.

ORDERED REWRITING

Problem: $x + y \longrightarrow y + x$ does not terminate

Solution: use permutative rules only if term becomes lexicographically smaller.

Example: $b + a \rightsquigarrow a + b$ but not $a + b \rightsquigarrow b + a$.

For types `nat`, `int` etc:

- lemmas **add_ac** sort any sum (+)
- lemmas **times_ac** sort any product (*)

Example: `apply (simp add: add_ac)` yields
 $(b + c) + a \rightsquigarrow \dots \rightsquigarrow a + (b + c)$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

Example: $(z \odot x) \odot (y \odot v)$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

Example: $(z \odot x) \odot (y \odot v)$

We want: $(z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

Example: $(z \odot x) \odot (y \odot v)$

We want: $(z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))$

We get: $(z \odot x) \odot (y \odot v) = v \odot (y \odot (x \odot z))$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

Example: $(z \odot x) \odot (y \odot v)$

We want: $(z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))$

We get: $(z \odot x) \odot (y \odot v) = v \odot (y \odot (x \odot z))$

We need: AC rule $x \odot (y \odot z) = y \odot (x \odot z)$

AC RULES

Example for associative-commutative rules:

Associative: $(x \odot y) \odot z = x \odot (y \odot z)$

Commutative: $x \odot y = y \odot x$

These 2 rules alone get stuck too early (not confluent).

Example: $(z \odot x) \odot (y \odot v)$

We want: $(z \odot x) \odot (y \odot v) = v \odot (x \odot (y \odot z))$

We get: $(z \odot x) \odot (y \odot v) = v \odot (y \odot (x \odot z))$

We need: AC rule $x \odot (y \odot z) = y \odot (x \odot z)$

If these 3 rules are present for an AC operator
Isabelle will order terms correctly

DEMO

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

Definition:

Let $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ be two rules with disjoint variables.

They form a **critical pair** if a non-variable subterm of l_1 unifies with l_2 .

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

Definition:

Let $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ be two rules with disjoint variables.

They form a **critical pair** if a non-variable subterm of l_1 unifies with l_2 .

Example:

Rules: (1) $f x \longrightarrow a$ (2) $g y \longrightarrow b$ (3) $f (g z) \longrightarrow b$

Critical pairs:

BACK TO CONFLUENCE

Last time: confluence in general is undecidable.

But: confluence for terminating systems is decidable!

Problem: overlapping lhs of rules.

Definition:

Let $l_1 \longrightarrow r_1$ and $l_2 \longrightarrow r_2$ be two rules with disjoint variables.

They form a **critical pair** if a non-variable subterm of l_1 unifies with l_2 .

Example:

Rules: (1) $f x \longrightarrow a$ (2) $g y \longrightarrow b$ (3) $f (g z) \longrightarrow b$

Critical pairs:

$$(1)+(3) \quad \{x \mapsto g z\} \quad a \xleftarrow{(1)} f g t \xrightarrow{(3)} b$$

$$(3)+(2) \quad \{z \mapsto y\} \quad b \xleftarrow{(3)} f g t \xrightarrow{(2)} b$$

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

,

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

But it can be made confluent by adding rules!

,

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

,

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

$$(1)+(3) \quad \{x \mapsto g z\} \quad a \xleftarrow{(1)} f g t \xrightarrow{(3)} b$$

shows that $a = b$ (because $a \xleftarrow{*} b$),

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

$$(1)+(3) \quad \{x \mapsto g z\} \quad a \xleftarrow{(1)} f g t \xrightarrow{(3)} b$$

shows that $a = b$ (because $a \xleftarrow{*} b$), so we add $a \longrightarrow b$ as a rule

COMPLETION

$$(1) f x \longrightarrow a \quad (2) g y \longrightarrow b \quad (3) f (g z) \longrightarrow b$$

is not confluent

But it can be made confluent by adding rules!

How: join all critical pairs

Example:

$$(1)+(3) \quad \{x \mapsto g z\} \quad a \xleftarrow{(1)} f g t \xrightarrow{(3)} b$$

shows that $a = b$ (because $a \xleftarrow{*} b$), so we add $a \longrightarrow b$ as a rule

This is the main idea of the Knuth-Bendix completion algorithm.

DEMO: WALDMEISTER

ORTHOGONAL REWRITING SYSTEMS

Definitions:

ORTHOGONAL REWRITING SYSTEMS

Definitions:

A rule $l \longrightarrow r$ is **left-linear** if no variable occurs twice in l .

ORTHOGONAL REWRITING SYSTEMS

Definitions:

A rule $l \longrightarrow r$ is **left-linear** if no variable occurs twice in l .

A **rewrite system** is **left-linear** if all rules are.

ORTHOGONAL REWRITING SYSTEMS

Definitions:

A rule $l \longrightarrow r$ is **left-linear** if no variable occurs twice in l .

A **rewrite system** is **left-linear** if all rules are.

A system is **orthogonal** if it is left-linear and has no critical pairs.

ORTHOGONAL REWRITING SYSTEMS

Definitions:

A rule $l \longrightarrow r$ is **left-linear** if no variable occurs twice in l .

A **rewrite system** is **left-linear** if all rules are.

A system is **orthogonal** if it is left-linear and has no critical pairs.

Orthogonal rewrite systems are confluent

ORTHOGONAL REWRITING SYSTEMS

Definitions:

A **rule** $l \longrightarrow r$ is **left-linear** if no variable occurs twice in l .

A **rewrite system** is **left-linear** if all rules are.

A system is **orthogonal** if it is left-linear and has no critical pairs.

Orthogonal rewrite systems are confluent

Application: functional programming languages

LAST TIME ON ISAR

- basic syntax
- proof and qed
- assume and show
- from and have
- the three modes of Isar

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

→ proof picks an intro rule automatically

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB have "..." proof

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "..." **proof**

- now **proof** picks an **elim** rule automatically

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB have "..." proof

- now **proof** picks an **elim** rule automatically
- triggered by **from**

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB have "..." proof

- now **proof** picks an **elim** rule automatically
- triggered by **from**
- first assumption of rule must unify with AB

BACKWARD AND FORWARD

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "..." **proof**

- now **proof** picks an **elim** rule automatically
- triggered by **from**
- first assumption of rule must unify with AB

General case: from $A_1 \dots A_n$ have R proof

- first n assumptions of rule must unify with $A_1 \dots A_n$
- conclusion of rule must unify with R

FIX AND OBTAIN

fix $v_1 \dots v_n$

FIX AND OBTAIN

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

FIX AND OBTAIN

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

obtain $v_1 \dots v_n$ **where** $\langle \text{prop} \rangle \langle \text{proof} \rangle$

FIX AND OBTAIN

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

obtain $v_1 \dots v_n$ **where** $\langle \text{prop} \rangle \langle \text{proof} \rangle$

Introduces new variables together with property

DEMO

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

then = **from this**

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

then = **from this**

thus = **then show**

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

then = **from this**

thus = **then show**

hence = **then have**

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

then = **from this**

thus = **then show**

hence = **then have**

with $A_1 \dots A_n$ = **from** $A_1 \dots A_n$ **this**

FANCY ABBREVIATIONS

this = the previous fact proved or assumed

then = **from this**

thus = **then show**

hence = **then have**

with $A_1 \dots A_n$ = **from** $A_1 \dots A_n$ **this**

?thesis = the last enclosing goal statement

MOREOVER AND ULTIMATELY

have $X_1: P_1 \dots$

have $X_2: P_2 \dots$

\vdots

have $X_n: P_n \dots$

from $X_1 \dots X_n$ **show** \dots

MOREOVER AND ULTIMATELY

have $X_1: P_1 \dots$

have $X_2: P_2 \dots$

\vdots

have $X_n: P_n \dots$

from $X_1 \dots X_n$ **show** \dots

wastes lots of brain power

on names $X_1 \dots X_n$

MOREOVER AND ULTIMATELY

have $X_1: P_1 \dots$

have $X_2: P_2 \dots$

\vdots

have $X_n: P_n \dots$

from $X_1 \dots X_n$ **show** \dots

have $P_1 \dots$

moreover have $P_2 \dots$

\vdots

moreover have $P_n \dots$

ultimately show \dots

wastes lots of brain power

on names $X_1 \dots X_n$

GENERAL CASE DISTINCTIONS

show *formula*

proof -

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by blast**

qed

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by blast**

qed

{ ... } is a proof block similar to **proof ... qed**

GENERAL CASE DISTINCTIONS

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by blast**

qed

{ ... } is a proof block similar to **proof ... qed**

{ **assume** P_1 ... **have** P <proof> }

stands for $P_1 \implies P$

MIXING PROOF STYLES

from ...

have ...

apply - make incoming facts assumptions

apply (...)

:

apply (...)

done

DEMO

WE HAVE LEARNED TODAY ...

→ Conditional term rewriting

WE HAVE LEARNED TODAY ...

- Conditional term rewriting
- Congruence and AC rules

WE HAVE LEARNED TODAY ...

- Conditional term rewriting
- Congruence and AC rules
- More on confluence

WE HAVE LEARNED TODAY ...

- Conditional term rewriting
- Congruence and AC rules
- More on confluence
- Completion

WE HAVE LEARNED TODAY ...

- Conditional term rewriting
- Congruence and AC rules
- More on confluence
- Completion
- Isar: fix, obtain, abbreviations, moreover, ultimately

EXERCISES

- Find critical pairs for your DNF solution from last time
- Complete rules to a terminating, confluent system
- Add AC rules for \wedge and \vee
- Decide $((C \vee B) \wedge A) = (\neg(A \wedge B) \longrightarrow C \wedge A)$ with these simp-rules
- Give an Isar proof of the rich grandmother theorem
(automated methods allowed, but proof must be explaining)