NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications
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

⇒ \( l \rightarrow r \) applicable to term \( t[s] \)
Applying a Rewrite Rule

$\rightarrow l \rightarrow r \text{ applicable to term } t[s]$

if there is substitution $\sigma$ such that $\sigma l = s$
Applying a Rewrite Rule

→ l → r applicable to term t[s]
   if there is substitution σ such that σ l = s

→ Result: t[σ r]
Applying a Rewrite Rule

$\Rightarrow l \rightarrow r$ applicable to term $t[s]$ if there is substitution $\sigma$ such that $\sigma l = s$

$\Rightarrow$ Result: $t[\sigma r]$

$\Rightarrow$ Equationally: $t[s] = t[\sigma r]$

Example:
APPPLYING A REWRITE RULE

→ *l* → *r* applicable to term *t*[s]
    if there is substitution σ such that σ *l* = *s*

→ Result: *t*[σ *r*]

→ Equationally: *t*[s] = *t*[σ *r*]

Example:

Rule: 0 + *n* → *n*

Term: *a* + (0 + (b + c))
Applying a Rewrite Rule

→ \( l \rightarrow r \) applicable to term \( t[s] \)
  if there is substitution \( \sigma \) such that \( \sigma \ l = s \)

→ **Result:** \( t[\sigma \ r] \)

→ **Equationally:** \( t[s] = t[\sigma \ r] \)

**Example:**

**Rule:** \( 0 + n \rightarrow n \)

**Term:** \( a + (0 + (b + c)) \)

**Substitution:** \( \sigma = \{ n \mapsto b + c \} \)
Applying a Rewrite Rule

$\rightarrow \quad l \longrightarrow r$ applicable to term $t[s]$ if there is substitution $\sigma$ such that $\sigma \; l = s$

$\rightarrow \quad$ Result: $t[\sigma \; r]$

$\rightarrow \quad$ Equationally: $t[s] = t[\sigma \; r]$

Example:

Rule: $0 + n \longrightarrow n$

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

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

Result: $a + (b + c)$
Rewrite rules can be conditional:

$$\left[ P_1 \ldots P_n \right] \implies l = r$$
Rewrite rules can be conditional:

\[
[P_1 \ldots P_n] \quad \Rightarrow \quad l = r
\]

is **applicable** to term \( t[s] \) with \( \sigma \) if

\(\sigma l = s \) and

\(\sigma P_1, \ldots, \sigma P_n \) are provable by rewriting.
Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.
Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

\[ \text{lemma} \quad \left( f \ x = g \ x \land g \ x = f \ x \right) \implies f \ x = 2 \]
Rewriting with Assumptions

Last time: Isabelle uses assumptions in rewriting.

Can lead to non-termination.

Example:

\[ \text{lemma} \quad " f\ x = g\ x \land g\ x = f\ x \implies f\ x = 2" \]

simp  \quad \text{use and simplify assumptions}

(simp (no_asm))  \quad \text{ignore assumptions}

(simp (no_asm_use))  \quad \text{simplify, but do not use assumptions}

(simp (no_asm_simp))  \quad \text{use, but do not simplify assumptions}
Preprocessing (recursive) for maximal simplification power:

\[-A \iff A = False\]

\[A \rightarrow B \iff A \implies B\]

\[A \land B \iff A, B\]

\[\forall x. \ A \ x \iff A \ ?x\]

\[A \iff A = True\]
Preprocessing (recursive) for maximal simplification power:

\[ \neg A \iff A = False \]
\[ A \rightarrow B \iff A \implies B \]
\[ A \land B \iff A, B \]
\[ \forall x. A \ x \iff A \ ?x \]
\[ A \iff A = True \]

Example:
\[ (p \rightarrow q \land \neg r) \land s \]
\[ \rightarrow \]
Preprocessing (recursive) for maximal simplification power:

\[ \neg A \iff A = False \]
\[ A \rightarrow B \iff A \rightarrowrightarrow B \]
\[ A \land B \iff A, B \]
\[ \forall x. A x \iff A ?x \]
\[ A \iff A = True \]

Example:

\[ (p \rightarrowrightarrow q \land \neg r) \land s \]

\[ \iff \]

\[ p \rightarrowrightarrow q = True \quad r = False \quad s = True \]
DEMO
CASE SPLITTING WITH SIMP

\[ P (\text{if } A \text{ then } s \text{ else } t) = (A \rightarrow P \ s) \land (\neg A \rightarrow P \ t) \]
CASE SPLITTING WITH SIMP

\[ P \text{ (if } A \text{ then } s \text{ else } t) = (A \rightarrow P \ s) \land (\neg A \rightarrow P \ t) \]

Automatic
**Case splitting with simp**

\[ P \ (\text{if } A \text{ then } s \text{ else } t) \]
\[ = \]
\[ (A \rightarrow P \ s) \land (\neg A \rightarrow P \ t) \]

**Automatic**

\[ P \ (\text{case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) \]
\[ = \]
\[ (e = 0 \rightarrow P \ a) \land (\forall n. \ e = \text{Suc } n \rightarrow P \ b) \]
CASE SPLITTING WITH SIMP

\[ P \ (\text{if } A \text{ then } s \text{ else } t) = (A \to P \ s) \land (\neg A \to P \ t) \]

Automatic

\[ P \ (\text{case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) = (e = 0 \to P \ a) \land (\forall n. \ e = \text{Suc } n \to P \ b) \]

Manually: apply (simp split: nat.split)
CASE SPLITTING WITH SIMP

\[ P \ (\text{if } A \text{ then } s \text{ else } t) = (A \rightarrow P \ s) \land (\neg A \rightarrow P \ t) \]

Automatic

\[ P \ (\text{case } e \text{ of } 0 \Rightarrow a \mid \text{Suc } n \Rightarrow b) = (e = 0 \rightarrow P \ a) \land (\forall n. \ e = \text{Suc } n \rightarrow P \ b) \]

Manually: apply (simp split: nat.split)

Similar for any data type t: t.split
congruence rules are about using context

**Example**: in $P \rightarrow Q$ we could use $P$ to simplify terms in $Q$
CONGRUENCE RULES

congruence rules are about using context

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

For $\Rightarrow$ hardwired (assumptions used in rewriting)
Congruence Rules

Congruence rules are about using context.

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

For $\equiv$ hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

**Example:** $[P = P'; P' \Rightarrow Q = Q'] \Rightarrow (P \rightarrow Q) = (P' \rightarrow Q')$

**Read:** to simplify $P \rightarrow Q$
Congruence Rules

Congruence rules are about using context.

**Example:** in $P \rightarrow Q$ we could use $P$ to simplify terms in $Q$.

For $\iff$ hardwired (assumptions used in rewriting).

For other operators expressed with conditional rewriting.

**Example:** $[P = P'; P' \iff Q = Q'] \iff (P \rightarrow Q) = (P' \rightarrow Q')$.

**Read:** to simplify $P \rightarrow Q$.

$\rightarrow$ first simplify $P$ to $P'$.
**Congruence Rules**

Congruence rules are about using context

**Example:** in \( P \rightarrow Q \) we could use \( P \) to simplify terms in \( Q \)

For \( \rightarrow \) hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

**Example:** \([P = P'; P' \rightarrow Q = Q'] \rightarrow (P \rightarrow Q) = (P' \rightarrow Q')\)

**Read:** to simplify \( P \rightarrow Q \)

→ first simplify \( P \) to \( P' \)

→ then simplify \( Q \) to \( Q' \) using \( P' \) as assumption
CONGRUENCE RULES

congruence rules are about using context

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

For $\Rightarrow$ hardwired (assumptions used in rewriting)

For other operators expressed with conditional rewriting.

Example: \[ [P = P'; P' \Rightarrow Q = Q'] \Rightarrow (P \rightarrow Q) = (P' \rightarrow Q') \]

Read: to simplify $P \rightarrow Q$

$\Rightarrow$ first simplify $P$ to $P'$
$\Rightarrow$ then simplify $Q$ to $Q'$ using $P'$ as assumption
$\Rightarrow$ the result is $P' \rightarrow Q'$
Sometimes useful, but not used automatically (slowdown):

\texttt{conj\_cong}: \( [P = P'; P' \implies Q = Q'] \implies (P \land Q) = (P' \land Q') \)
\textbf{MORE CONGRUENCE}

Sometimes useful, but not used automatically (slowdown):

\textbf{conj\_cong}: \([P = P'; P' \implies Q = Q'] \implies (P \land Q) = (P' \land Q')\)

Context for if-then-else:

\textbf{if\_cong}: \([b = c; c \implies x = u; \neg c \implies y = v] \implies \)

\((\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)\)
MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

**conj_cong:** \[ P = P'; P' \implies Q = Q' \] \implies (P \land Q) = (P' \land Q')

Context for if-then-else:

**if_cong:** \[ [b = c; c \implies x = u; \neg c \implies y = v] \implies \]

\[(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)\]

Prevent rewriting inside then-else (default):

**if_weak_cong:** \[ b = c \implies (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } x \text{ else } y)\]
MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

**conj_cong:** \[ P = P'; P' \implies Q = Q' \] \implies (P \land Q) = (P' \land Q')

Context for if-then-else:

**if_cong:** \[ b = c; c \implies x = u; \neg c \implies y = v \] \implies (if b then x else y) = (if c then u else v)

Prevent rewriting inside then-else (default):

**if_weak_cong:** b = c \implies (if b then x else y) = (if c then x else y)

→ declare own congruence rules with [cong] attribute
MORE CONGRUENCE

Sometimes useful, but not used automatically (slowdown):

**conj_cong:** \([P = P'; P' \implies Q = Q'] \implies (P \land Q) = (P' \land Q')\)

Context for if-then-else:

**if_cong:** \([b = c; c \implies x = u; \neg c \implies y = v] \implies (\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)\)

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]`
Problem: $x + y \rightarrow y + x$ does not terminate
Problem: $x + y \rightarrow y + x$ does not terminate

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

Example:
Problem: \( x + y \rightarrow y + x \) does not terminate

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

Example: \( b + a \not\rightarrow a + b \) but not \( a + b \not\rightarrow b + a \).
ORDERED REWRITING

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

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

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

For types nat, int etc:

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

Example: apply (simp add: add_ac) yields

$$(b + c) + a \sim \cdots \sim a + (b + c)$$
AC Rules

Example for associative-commutative rules:

Associative: \[(x \mathbin{\circ} y) \mathbin{\circ} z = x \mathbin{\circ} (y \mathbin{\circ} z)\]

Commutative: \[x \mathbin{\circ} y = y \mathbin{\circ} x\]
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: \((z \odot x) \odot (y \odot v)\)
Example for associative-commutative rules:

**Associative:** \((x \circ y) \circ z = x \circ (y \circ z)\)

**Commutative:** \(x \circ y = y \circ x\)

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

**Example:** \((z \circ x) \circ (y \circ v)\)

**We want:** \((z \circ x) \circ (y \circ v) = v \circ (x \circ (y \circ z))\)
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:** \((x \circ y) \circ z = x \circ (y \circ z)\)

**Commutative:** \(x \circ y = y \circ x\)

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

Example: \((z \circ x) \circ (y \circ v)\)

We want: \((z \circ x) \circ (y \circ v) = v \circ (x \circ (y \circ z))\)

We get: \((z \circ x) \circ (y \circ v) = v \circ (y \circ (x \circ z))\)

We need: **AC rule** \(x \circ (y \circ z) = y \circ (x \circ z)\)
Example for associative-commutative rules:

**Associative:** \[(x \circ y) \circ z = x \circ (y \circ z)\]

**Commutative:** \[x \circ y = y \circ x\]

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

Example: \[(z \circ x) \circ (y \circ v)\]

We want: \[(z \circ x) \circ (y \circ v) = v \circ (x \circ (y \circ z))\]

We get: \[(z \circ x) \circ (y \circ v) = v \circ (y \circ (x \circ z))\]

We need: **AC rule** \[x \circ (y \circ z) = y \circ (x \circ z)\]

If these 3 rules are present for an AC operator

Isabelle will order terms correctly
Last time: confluence in general is undecidable.
Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!
Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!
Problem: overlapping lhs of rules.
Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!
Problem: overlapping lhs of rules.

Definition:
Let $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ be two rules with disjoint variables.
They form a critical pair if a non-variable subterm of $l_1$ unifies with $l_2$. 
**Last time:** confluence in general is undecidable.

**But:** confluence for terminating systems is decidable!

**Problem:** overlapping lhs of rules.

**Definition:**

Let \( l_1 \rightarrow r_1 \) and \( l_2 \rightarrow r_2 \) 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: \( (1) \ f \ x \rightarrow a \quad (2) \ g \ y \rightarrow b \quad (3) \ f \ (g \ z) \rightarrow b \)

Critical pairs:
Last time: confluence in general is undecidable.
But: confluence for terminating systems is decidable!
Problem: overlapping lhs of rules.

Definition:
Let \( l_1 \rightarrow r_1 \) and \( l_2 \rightarrow r_2 \) 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) \( f \ x \rightarrow a \) (2) \( g \ y \rightarrow b \) (3) \( f \ (g \ z) \rightarrow b \)
Critical pairs:
\( (1)+(3) \) \( \{ x \leftarrow g \ z \} \) \( a \overset{(1)}{\leftarrow} f \ g \ t \overset{(3)}{\rightarrow} b \)
\( (3)+(2) \) \( \{ z \leftarrow y \} \) \( b \overset{(3)}{\leftarrow} f \ g \ t \overset{(2)}{\rightarrow} b \)
(1) $f \ x \rightarrow a$    (2) $g \ y \rightarrow b$    (3) $f \ (g \ z) \rightarrow b$

is not confluent
COMPLETION

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

is not confluent

But it can be made confluent by adding rules!
(1) \( f \ x \rightarrow a \)  (2) \( g \ y \rightarrow b \)  (3) \( f \ (g \ z) \rightarrow b \)

is not confluent

But it can be made confluent by adding rules!

**How:** join all critical pairs
(1) \( f \ x \rightarrow a \)  (2) \( g \ y \rightarrow b \)  (3) \( f \ (g \ z) \rightarrow b \)

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 \( a \xleftrightarrow{*} b \)),

\[\]
COMPLETION

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

is not confluent

But it can be made confluent by adding rules!

**How:** join all critical pairs

Example:

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

shows that $a = b$ (because $a \xrightarrow{*} b$), so we add $a \rightarrow b$ as a rule
But it can be made confluent by adding rules!

How: join all critical pairs

Example:

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

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

This is the main idea of the Knuth-Bendix completion algorithm.
DEMO: WALDMEISTER
Definitions:
Definitions:

A rule \( l \rightarrow r \) is **left-linear** if no variable occurs twice in \( l \).
Definitions:
A rule \( l \rightarrow r \) 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 $l \rightarrow r$ 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.
Definitions:
A rule \( l \rightarrow r \) 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 $l \rightarrow r$ 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 reasoning: ... have "$A \land B$" proof
Backward reasoning: ... have \( A \land B \) proof

\[\Rightarrow\] proof picks an intro rule automatically
Backward reasoning: ... have \( A \land B \) proof

- proof picks an intro rule automatically
- conclusion of rule must unify with \( A \land B \)
Backward and Forward

Backward reasoning: ... have \( A \land B \) proof

\( \rightarrow \) proof picks an intro rule automatically
\( \rightarrow \) conclusion of rule must unify with \( A \land B \)

Forward reasoning: ...

assume AB: \( A \land B \)
from AB have "..." proof
Backward reasoning: . . . have "\( A \land B \)" proof

- \( \text{proof} \) picks an \text{intro} rule automatically
- conclusion of rule must unify with \( A \land B \)

Forward reasoning: . . .

\text{assume} \ AB: "\( A \land B \)"

\text{from} \ AB \ have "\ldots" \text{proof}

- now \( \text{proof} \) picks an \text{elim} rule automatically
BACKWARD AND FORWARD

Backward reasoning: . . . have "A ∧ B" proof

→ proof picks an intro rule automatically
→ conclusion of rule must unify with A ∧ B

Forward reasoning: . . .

assume AB: "A ∧ B"

from AB have " . . ." proof

→ now proof picks an elim rule automatically
→ triggered by from
Backward and Forward

Backward reasoning: . . . have ”$A \land B$” proof

→ proof picks an intro rule automatically
→ conclusion of rule must unify with $A \land B$

Forward reasoning: . . .

   assume AB: ”$A \land B$”
   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 "\(A \land B\)" proof

\[ \rightarrow \text{proof} \text{ picks an intro rule automatically} \]
\[ \rightarrow \text{conclusion of rule must unify with } A \land B \]

Forward reasoning: . . .

\[ \text{assume } AB: "A \land B" \]
\[ \text{from } AB \text{ have } "\ldots" \text{ proof} \]

\[ \rightarrow \text{now proof picks an elim rule automatically} \]
\[ \rightarrow \text{triggered by from} \]
\[ \rightarrow \text{first assumption of rule must unify with } AB \]

General case: from \(A_1 \ldots A_n\) have \(R\) proof

\[ \rightarrow \text{first } n \text{ assumptions of rule must unify with } A_1 \ldots A_n \]
\[ \rightarrow \text{conclusion of rule must unify with } R \]
**Fix and Obtain**

\[ \text{fix } v_1 \ldots v_n \]
\textbf{Fix and Obtain}

\texttt{fix } \nu_1 \ldots \nu_n

Introduces new arbitrary but fixed variables
\( (\sim \text{parameters, } \land ) \)
**Fix and Obtain**

$$\textbf{fix } v_1 \ldots v_n$$

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

$$\textbf{obtain } v_1 \ldots v_n \textbf{ where } \langle \text{prop} \rangle \langle \text{proof} \rangle$$
**Fix and Obtain**

**fix** $v_1 \ldots v_n$

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

**obtain** $v_1 \ldots v_n$ where $\langle$prop$\rangle$ $\langle$proof$\rangle$

Introduces new variables together with property
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 \( A_1 \ldots A_n \) = from \( A_1 \ldots A_n \) this
**Fancy Abbreviations**

- **this** = the previous fact proved or assumed
- **then** = from this
- **thus** = then show
- **hence** = then have
- **with** \( A_1 \ldots A_n \) = from \( A_1 \ldots A_n \) this
- **?thesis** = the last enclosing goal statement
Moreover and Ultimately

have $X_1: P_1 \ldots$

have $X_2: P_2 \ldots$

$\vdots$

have $X_n: P_n \ldots$

from $X_1 \ldots X_n$ show $\ldots$
Moreover and Ultimately

have $X_1$: $P_1$ 

have $X_2$: $P_2$ 

... 

have $X_n$: $P_n$ 

from $X_1 \ldots X_n$ show 

wastes lots of brain power 

on names $X_1 \ldots X_n$
Moreover and Ultimately

\[
\begin{align*}
\text{have } & \ X_1: P_1 \ldots \\
\text{have } & \ X_2: P_2 \ldots \\
\vdots & \\
\text{have } & \ X_n: P_n \ldots \\
\text{from } & \ X_1 \ldots X_n \text{ show } \\
\text{wastes lots of brain power} \\
\text{on names } & \ X_1 \ldots X_n \\
\text{have } & \ P_1 \ldots \\
\text{moreover have } & \ P_2 \ldots \\
\vdots & \\
\text{moreover have } & \ P_n \ldots \\
\text{ultimately show } & \\
\end{align*}
\]
GENERAL CASE DISTINCTIONS

show \( \text{formula} \)
proof -
**General Case Distinctions**

**show** \( \text{formula} \)  
**proof** -  
**have** \( P_1 \lor P_2 \lor P_3 \)  
<proof>
show formula

proof -

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

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

ultimately show ?thesis by blast
**General Case Distinctions**

**show** *formula*

**proof** -

**have** \( P_1 \lor P_2 \lor P_3 \) *<proof>*

moreover \{ **assume** \( P_1 \) \ldots **have** ?thesis *<proof>* \} 

moreover \{ **assume** \( P_2 \) \ldots **have** ?thesis *<proof>* \} 

ultimately show ?thesis by blast 

qed
**GENERAL CASE DISTINCTIONS**

show $formula$

proof -

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

moreover \[ \{ \text{assume } P_1 \ldots \text{ have } ?\text{thesis} \ <\text{proof}> \} \]

moreover \[ \{ \text{assume } P_2 \ldots \text{ have } ?\text{thesis} \ <\text{proof}> \} \]

moreover \[ \{ \text{assume } P_3 \ldots \text{ have } ?\text{thesis} \ <\text{proof}> \} \]
show \( formula \)

proof -

\begin{align*}
\text{have } P_1 \lor P_2 \lor P_3 & \quad \text{<proof> } \\
\text{moreover } \{ \text{ assume } P_1 \ldots \text{ have } ?\text{thesis} \text{ <proof> } \} \\
\text{moreover } \{ \text{ assume } P_2 \ldots \text{ have } ?\text{thesis} \text{ <proof> } \} \\
\text{moreover } \{ \text{ assume } P_3 \ldots \text{ have } ?\text{thesis} \text{ <proof> } \} \\
\text{ultimately show } ?\text{thesis by blast} \\
\end{align*}

qed
show \textit{formula}

\textbf{proof -}

\textbf{have} $P_1 \lor P_2 \lor P_3$ \textless proof\textgreater

moreover \quad \{ \textbf{assume} \hspace{0.5em} P_1 \hspace{1em} \ldots \hspace{1em} \textbf{have} \hspace{0.5em} \textit{thesis} \hspace{0.5em} \textless proof\textgreater \}

moreover \quad \{ \textbf{assume} \hspace{0.5em} P_2 \hspace{1em} \ldots \hspace{1em} \textbf{have} \hspace{0.5em} \textit{thesis} \hspace{0.5em} \textless proof\textgreater \}

moreover \quad \{ \textbf{assume} \hspace{0.5em} P_3 \hspace{1em} \ldots \hspace{1em} \textbf{have} \hspace{0.5em} \textit{thesis} \hspace{0.5em} \textless proof\textgreater \}

\textbf{ultimately show} \hspace{0.5em} \textit{thesis} \hspace{0.5em} \textbf{by blast}

\textbf{qed}

\{ \ldots \} \text{ is a proof block similar to} \textbf{proof} \ldots \textbf{qed}
**GENERAL CASE DISTINCTIONS**

show formula

proof -

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

moreover \{ assume $P_1$ \ldots have ?thesis <proof> \}

moreover \{ assume $P_2$ \ldots have ?thesis <proof> \}

moreover \{ assume $P_3$ \ldots have ?thesis <proof> \}

ultimately show ?thesis by blast

qed

\{ \ldots \} is a proof block similar to proof ... qed

\{ assume $P_1$ \ldots 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
→ Find critical pairs for your DNF solution from last time

→ Complete rules to a terminating, confluent system

→ Add AC rules for $\land$ and $\lor$

→ Decide $((C \lor B) \land A) = (\neg(A \land B) \rightarrow C \land A)$ with these simp-rules

→ Give an Isar proof of the rich grandmother theorem
  (automated methods allowed, but proof must be explaining)