NICTA Advanced Course

Theorem Proving

Principles, Techniques, Applications

$$a = b \leq c \leq \ldots$$

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
  - **More recursion, Calculational reasoning**
  - Hoare logic, proofs about programs
  - Locales, Presentation

LAST WEEK

➜ Constructive Logic & Curry-Howard-Isomorphism

Last Week

- ➜ Constructive Logic & Curry-Howard-Isomorphism
- ➜ The Coq System

LAST WEEK

- ➜ Constructive Logic & Curry-Howard-Isomorphism
- ➜ The Coq System
- ➜ The HOL4 system

LAST WEEK

➜ Constructive Logic & Curry-Howard-Isomorphism

➜ The Coq System

➜ The HOL4 system

➜ Before that: datatypes, recursion, induction

The Choice

The Choice

➜ Limited expressiveness, automatic termination

- primrec

The Choice

➜ Limited expressiveness, automatic termination

- primrec

➜ High expressiveness, prove termination manually

- recdef

consts sep :: "'a $\times$ 'a list $\Rightarrow$ 'a list"

recdef sep "measure (λ(a, xs). size xs)"

"sep (a, x # y # zs) = x # a # sep (a, y # zs)"

"sep (a, xs) = xs"

RECDEF — EXAMPLES

consts sep :: ”’a $\times$ ’a list $\Rightarrow$ ’a list”

recdef sep ”measure (λ(a, xs). size xs)”

”sep (a, x # y # zs) = x # a # sep (a, y # zs)”

”sep (a, xs) = xs”

consts ack :: ”nat $\times$ nat $\Rightarrow$ nat”

recdef ack ”measure (λm. m) $<$*lex*$>$ measure (λn. n)”

”ack (0, n) = Suc n”

”ack (Suc m, 0) = ack (m, 1)”

”ack (Suc m, Suc n) = ack (m, ack (Suc m, n))”

RECDEF

➜ The definiton:

- one parameter
- free pattern matching, order of rules important
- termination relation
  (**measure** sufficient for most cases)

RECDEF

➜ The definiton:

- one parameter
- free pattern matching, order of rules important
- termination relation
  (**measure** sufficient for most cases)

➜ Termination relation:

- must decrease for each recursive call
- must be well founded

RECDEF

➜ The definiton:

- one parameter
- free pattern matching, order of rules important
- termination relation
  (**measure** sufficient for most cases)

➜ Termination relation:

- must decrease for each recursive call
- must be well founded

➜ Generates own induction principle

RECDEF — INDUCTION PRINCIPLE

➜ Each **recdef** definition induces an induction principle

RECDEF — INDUCTION PRINCIPLE

➜ Each **recdef** definition induces an induction principle

➜ For each equation:

show that the property holds for the lhs provided it holds for each recursive call on the rhs

- Each **recdef** definition induces an induction principle

- For each equation:

  show that the property holds for the lhs provided it holds for each recursive call on the rhs

- Example **sep.induct**:

$$
\begin{array}{l}
[\![\ \bigwedge a.\ P\ a\ [];\\
\quad \bigwedge a\ w.\ P\ a\ [w]\\
\quad \bigwedge a\ x\ y\ zs.\ P\ a\ (y\#zs) \Longrightarrow P\ a\ (x\#y\#zs);\\
]\!] \Longrightarrow P\ a\ xs
\end{array}
$$

TERMINATION

Isabelle tries to prove termination automatically

➜ For most functions and termination relations this works.

TERMINATION

Isabelle tries to prove termination automatically

- ➜ For most functions and termination relations this works.
- ➜ Sometimes not

TERMINATION

Isabelle tries to prove termination automatically

➜ For most functions and termination relations this works.

➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal

TERMINATION

Isabelle tries to prove termination automatically

- ➜ For most functions and termination relations this works.
- ➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal
- ➜ You can give **hints** (additional lemmas) to the recdef package:

recdef quicksort ”measure length”

quicksort [] = []

quicksort $(x\#xs)$ = quicksort $[y \in xs.y \leq x]@[x]@$ quicksort $[y \in xs.x < y]$

(hints recdef_simp: less_Suc_eq_le)

Isabelle tries to prove termination automatically

- ➜ For most functions and termination relations this works.
- ➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal
- ➜ You can give **hints** (additional lemmas) to the recdef package:

recdef quicksort ”measure length”

quicksort [] = []

quicksort $(x\#xs)$ = quicksort $[y \in xs.y \leq x]@[x]@$ quicksort $[y \in xs.x < y]$

(hints recdef_simp: less_Suc_eq_le)

For exploration:

- ➜ allow failing termination proof

Termination

Isabelle tries to prove termination automatically

- ➜ For most functions and termination relations this works.
- ➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal
- ➜ You can give **hints** (additional lemmas) to the recdef package:

recdef quicksort ”measure length”

quicksort [] = []

quicksort $(x\#xs)$ = quicksort $[y \in xs. y \leq x]@[x]@$ quicksort $[y \in xs. x < y]$

(hints recdef_simp: less_Suc_eq_le)

For exploration:

- ➜ allow failing termination proof
- ➜ **recdef (permissive)** quicksort ”measure length”

Isabelle tries to prove termination automatically

- ➜ For most functions and termination relations this works.
- ➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal
- ➜ You can give **hints** (additional lemmas) to the recdef package:

recdef quicksort "measure length"

quicksort [] = []

quicksort $(x\#xs)$ = quicksort $[y \in xs. y \leq x]@[x]@$ quicksort $[y \in xs. x < y]$

(hints recdef_simp: less_Suc_eq_le)

For exploration:

- ➜ allow failing termination proof
- ➜ **recdef (permissive)** quicksort "measure length"
- ➜ termination conditions as assumption in simp and induct rules

DEMO

HOW DOES RECDEF WORK?

We need: general recursion operator

HOW DOES RECDEF WORK?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

How does recdef work?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

(F stands for the recursion equations)

Example:

How does recdef work?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

(F stands for the recursion equations)

Example:

➜ recursion equations: $f = 0 \qquad f\ (\mathsf{Suc}\ n) = f n$

How does recdef work?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

(F stands for the recursion equations)

Example:

- recursion equations: $f = 0 \qquad f\ (\mathsf{Suc}\ n) = f n$
- as one λ-term: $f = \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow f\ n$

HOW DOES RECDEF WORK?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

(F stands for the recursion equations)

Example:

- recursion equations: $f = 0 \qquad f\ (\mathsf{Suc}\ n) = f n$
- as one λ-term: $f = \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow f\ n$
- functor: $F = \lambda f.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow f\ n$

How does recdef work?

We need: general recursion operator

something like: $rec\ F = F\ (rec\ F)$

(F stands for the recursion equations)

Example:

- recursion equations: $f = 0 \qquad f\ (\mathsf{Suc}\ n) = f n$
- as one λ-term: $f = \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow f\ n$
- functor: $F = \lambda f.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow f\ n$

- $rec :: ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta)$ like above cannot exist in HOL (only total functions)
- But 'guarded' form possible:
  $\mathsf{wfrec} :: (\alpha \times \alpha)\ \mathsf{set} \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta)$
- $(\alpha \times \alpha)\ \mathsf{set}$ a well founded order, decreasing with execution

How does recdef work?

Why $rec\ F = F\ (rec\ F)$?

How does recdef work?

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$
\begin{array}{lcl}
F & \equiv & \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n \\
f & \equiv & rec\ F
\end{array}
$$

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$F \equiv \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n$$
$$f \equiv rec\ F$$

$$f\ 0 = rec\ F\ 0$$

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$\begin{array}{rcl} F & \equiv & \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n \\ f & \equiv & rec\ F \\ \\ f\ 0 & = & rec\ F\ 0 \\ \dots & = & F\ (rec\ F)\ 0 \end{array}$$

How does recdef work?

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$\begin{array}{rcl} F & \equiv & \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n \\ f & \equiv & rec\ F \end{array}$$

$$\begin{array}{rcl} f\ 0 & = & rec\ F\ 0 \\ \ldots & = & F\ (rec\ F)\ 0 \\ \ldots & = & (\lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n)\ (rec\ F)\ 0 \end{array}$$

How does recdef work?

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$\begin{array}{rcl} F & \equiv & \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n \\ f & \equiv & rec\ F \end{array}$$

$$\begin{array}{rcl} f\ 0 & = & rec\ F\ 0 \\ \ldots & = & F\ (rec\ F)\ 0 \\ \ldots & = & (\lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n)\ (rec\ F)\ 0 \\ \ldots & = & (\mathsf{case}\ 0\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow rec\ F\ n) \end{array}$$

How does recdef work?

Why $rec\ F = F\ (rec\ F)$?

Because we want the recursion equations to hold.

Example:

$$
\begin{array}{rcl}
F & \equiv & \lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n \\
f & \equiv & rec\ F \\
\\
f\ 0 & = & rec\ F\ 0 \\
\ldots & = & F\ (rec\ F)\ 0 \\
\ldots & = & (\lambda g.\ \lambda n'.\ \mathsf{case}\ n'\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow g\ n)\ (rec\ F)\ 0 \\
\ldots & = & (\mathsf{case}\ 0\ \mathsf{of}\ 0 \Rightarrow 0 \mid \mathsf{Suc}\ n \Rightarrow rec\ F\ n) \\
\ldots & = & 0
\end{array}
$$

Well Founded Orders

Definition

$<_r$ is well founded if well founded induction holds

wf $r \equiv \forall P. (\forall x. (\forall y <_r x. P\ y) \longrightarrow P\ x) \longrightarrow (\forall x. P\ x)$

WELL FOUNDED ORDERS

Definition

$<_r$ is well founded if well founded induction holds

wf $r \equiv \forall P.\ (\forall x.\ (\forall y <_r x. P\ y) \longrightarrow P\ x) \longrightarrow (\forall x.\ P\ x)$

Well founded induction rule:

$$\frac{\text{wf } r \quad \bigwedge x.\ (\forall y <_r x. Py) \Longrightarrow Px}{Pa}$$

WELL FOUNDED ORDERS

Definition

$<_r$ is well founded if well founded induction holds

wf $r \equiv \forall P.\ (\forall x.\ (\forall y <_r x. P\ y) \longrightarrow P\ x) \longrightarrow (\forall x.\ P\ x)$

Well founded induction rule:

$$\frac{\text{wf } r \quad \bigwedge x.\ (\forall y <_r x. Py) \Longrightarrow Px}{Pa}$$

Alternative definition (equivalent):

there are no infi nite descending chains, or (equivalent):

every nonempty set has a minimal element wrt $<_r$

$$\begin{array}{lcl} \text{min } r\ Q\ x & \equiv & \forall y \in Q.\ y \not<_r x \\ \text{wf } r & = & (\forall Q \neq \{\}.\ \exists m \in Q.\ \text{min } r\ Q\ m) \end{array}$$

Well Founded Orders: Examples

➜ $<$ on $\mathbb{N}$ is well founded
well founded induction = complete induction

WELL FOUNDED ORDERS: EXAMPLES

➜ $<$ on $\mathbb{N}$ is well founded
well founded induction = complete induction

➜ $>$ and $\leq$ on $\mathbb{N}$ are **not** well founded

WELL FOUNDED ORDERS: EXAMPLES

- ➜ $<$ on $\mathbb{N}$ is well founded
  well founded induction = complete induction
- ➜ $>$ and $\leq$ on $\mathbb{N}$ are **not** well founded
- ➜ $x <_r y = x$ dvd $y \wedge x \neq 1$ on $\mathbb{N}$ is well founded
  the minimal elements are the prime numbers

WELL FOUNDED ORDERS: EXAMPLES

- $<$ on $\mathbb{N}$ is well founded
  well founded induction = complete induction
- $>$ and $\leq$ on $\mathbb{N}$ are **not** well founded
- $x <_r y = x$ dvd $y \wedge x \neq 1$ on $\mathbb{N}$ is well founded
  the minimal elements are the prime numbers
- $(a, b) <_r (x, y) = a <_1 x \vee a = x \wedge b <_1 y$ is well founded
  if $<_1$ and $<_2$ are

Well Founded Orders: Examples

- $<$ on $\mathbb{N}$ is well founded
  well founded induction = complete induction
- $>$ and $\leq$ on $\mathbb{N}$ are **not** well founded
- $x <_r y = x \text{ dvd } y \wedge x \neq 1$ on $\mathbb{N}$ is well founded
  the minimal elements are the prime numbers
- $(a, b) <_r (x, y) = a <_1 x \vee a = x \wedge b <_1 y$ is well founded
  if $<_1$ and $<_2$ are
- $A <_r B = A \subset B \wedge \text{finite } B$ is well founded

Well Founded Orders: Examples

- $<$ on $\mathbb{N}$ is well founded
  well founded induction = complete induction
- $>$ and $\leq$ on $\mathbb{N}$ are **not** well founded
- $x <_r y = x \text{ dvd } y \wedge x \neq 1$ on $\mathbb{N}$ is well founded
  the minimal elements are the prime numbers
- $(a, b) <_r (x, y) = a <_1 x \vee a = x \wedge b <_1 y$ is well founded
  if $<_1$ and $<_2$ are
- $A <_r B = A \subset B \wedge \text{finite } B$ is well founded
- $\subseteq$ and $\subset$ in general are **not** well founded

More about well founded relations: *Term Rewriting and All That*

The Recursion Operator

Back to recursion: $rec\ F = F\ (rec\ F)$ not possible

Idea:

THE RECURSION OPERATOR

Back to recursion: $rec\ F = F\ (rec\ F)$ not possible

Idea: have wfrec $R\ F$ where R is well founded

The Recursion Operator

Back to recursion: $rec\ F = F\ (rec\ F)$ not possible

Idea: have wfrec $R\ F$ where R is well founded

Cut:

- ➜ only do recursion if parameter decreases wrt R
- ➜ otherwise: abort

The Recursion Operator

Back to recursion: $rec\ F = F\ (rec\ F)$ not possible

Idea: have wfrec $R\ F$ where R is well founded

Cut:

- ➜ only do recursion if parameter decreases wrt R
- ➜ otherwise: abort

- ➜ arbitrary :: α
  cut :: $(\alpha \Rightarrow \beta) \Rightarrow (\alpha \times \alpha)$ set $\Rightarrow \alpha \Rightarrow (\alpha \Rightarrow \beta)$
  cut $G\ R\ x \equiv \lambda y.$ if $(y, x) \in R$ then $G\ y$ else arbitrary

THE RECURSION OPERATOR

Back to recursion: $rec\ F = F\ (rec\ F)$ not possible

Idea: have wfrec $R\ F$ where R is well founded

Cut:

- ➜ only do recursion if parameter decreases wrt R
- ➜ otherwise: abort

- ➜ arbitrary :: α
  cut :: $(\alpha \Rightarrow \beta) \Rightarrow (\alpha \times \alpha)$ set $\Rightarrow \alpha \Rightarrow (\alpha \Rightarrow \beta)$
  cut $G\ R\ x \equiv \lambda y.$ if $(y, x) \in R$ then $G\ y$ else arbitrary

$$\text{wf } R \Longrightarrow \text{wfrec } R\ F\ x = F\ (\text{cut } (\text{wfrec } R\ F)\ R\ x)\ x$$

The Recursion Operator

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

The Recursion Operator

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

$$\mathsf{adm\_wf}\ R\ F \equiv \forall f\ g\ x.\ (\forall z.\ (z,x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x$$

The Recursion Operator

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

$$\mathsf{adm\_wf}\ R\ F \equiv \forall f\ g\ x.\ (\forall z.\ (z, x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x$$

Definition of wf_rec: again fi rst by induction, then by epsilon

$$\frac{}{(x, \qquad) \in \mathsf{wfrec\_rel}\ R\ F}$$

The Recursion Operator

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

$$\mathsf{adm\_wf}\ R\ F \equiv \forall f\ g\ x.\ (\forall z.\ (z,x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x$$

Definition of wf_rec: again fi rst by induction, then by epsilon

$$\frac{}{(x, F\ g\ x) \in \mathsf{wfrec\_rel}\ R\ F}$$

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

$$\mathsf{adm\_wf}\ R\ F \equiv \forall f\ g\ x.\ (\forall z.\ (z,x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x$$

Definition of wf_rec: again fi rst by induction, then by epsilon

$$\frac{\forall z.\ (z,x) \in R \longrightarrow (z, g\ z) \in \mathsf{wfrec\_rel}\ R\ F}{(x, F\ g\ x) \in \mathsf{wfrec\_rel}\ R\ F}$$

THE RECURSION OPERATOR

Admissible recursion

- ➜ recursive call for x only depends on parameters $y <_R x$
- ➜ describes exactly one function if R is well founded

$$\mathsf{adm\_wf}\ R\ F \equiv \forall f\ g\ x.\ (\forall z.\ (z,x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x$$

Definition of wf_rec: again fi rst by induction, then by epsilon

$$\frac{\forall z.\ (z,x) \in R \longrightarrow (z, g\ z) \in \mathsf{wfrec\_rel}\ R\ F}{(x, F\ g\ x) \in \mathsf{wfrec\_rel}\ R\ F}$$

$$\mathsf{wfrec}\ R\ F\ x \equiv \mathsf{THE}\ y.\ (x,y) \in \mathsf{wfrec\_rel}\ R\ (\lambda f\ x.\ F\ (\mathsf{cut}\ f\ R\ x)\ x)$$

More: John Harrison, *Inductive definitions: automation and application*

DEMO

CALCULATIONAL REASONING

The Goal

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

THE GOAL

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

Can we do this in Isabelle?

THE GOAL

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

Can we do this in Isabelle?

➜ Simplifier: too eager

The Goal

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

Can we do this in Isabelle?

➜ Simplifier: too eager

➜ Manual: difficult in apply stile

The Goal

$$
\begin{aligned}
x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\
\ldots &= 1 \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\
\ldots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\
\ldots &= (x^{-1})^{-1} \cdot x^{-1} \\
\ldots &= 1
\end{aligned}
$$

Can we do this in Isabelle?

➜ Simplifier: too eager

➜ Manual: difficult in apply stile

➜ Isar: with the methods we know, too verbose

The Problem

$$\begin{aligned} a &= b \\ \dots &= c \\ \dots &= d \end{aligned}$$

shows $a = d$ by transitivity of $=$

The Problem

$$\begin{array}{lcl} a & = & b \\ \ldots & = & c \\ \ldots & = & d \end{array}$$

shows $a = d$ by transitivity of $=$

Each step usually nontrivial (requires own subproof)

The Problem

$$\begin{array}{lcl} a & = & b \\ \ldots & = & c \\ \ldots & = & d \end{array}$$

shows $a = d$ by transitivity of $=$

Each step usually nontrivial (requires own subproof)

Solution in Isar:

➜ Keywords **also** and **finally** to delimit steps

Chains of equations

The Problem

$$\begin{array}{lcl} a & = & b \\ \ldots & = & c \\ \ldots & = & d \end{array}$$

shows $a = d$ by transitivity of $=$

Each step usually nontrivial (requires own subproof)

Solution in Isar:

➜ Keywords **also** and **finally** to delimit steps

➜ **...**: predefined schematic term variable, refers to right hand side of last expression

Chains of equations

The Problem

$$\begin{array}{lcl} a & = & b \\ \ldots & = & c \\ \ldots & = & d \end{array}$$

shows $a = d$ by transitivity of $=$

Each step usually nontrivial (requires own subproof)

Solution in Isar:

- ➜ Keywords **also** and **finally** to delimit steps
- ➜ **...**: predefined schematic term variable,
  refers to right hand side of last expression
- ➜ Automatic use of transitivity rules to connect steps

ALSO/FINALLY

have "$t_0 = t_1$" [proof]

also

ALSO/FINALLY

have "$t_0 = t_1$" [proof]

also

calculation register

"$t_0 = t_1$"

ALSO/FINALLY

have "$t_0 = t_1$" [proof]

also

have "$\ldots = t_2$" [proof]

calculation register

"$t_0 = t_1$"

ALSO/FINALLY

have "$t_0 = t_1$" [proof]	calculation register
also	"$t_0 = t_1$"
have "$\ldots = t_2$" [proof]	
also	"$t_0 = t_2$"

ALSO/FINALLY

have "$t_0 = t_1$" [proof]	calculation register
also	"$t_0 = t_1$"
have "$\ldots = t_2$" [proof]	
also	"$t_0 = t_2$"
$\vdots$	$\vdots$
also	"$t_0 = t_{n-1}$"

ALSO/FINALLY

have "$t_0 = t_1$" [proof]	calculation register
also	"$t_0 = t_1$"
have "$\ldots = t_2$" [proof]	
also	"$t_0 = t_2$"
$\vdots$	$\vdots$
also	"$t_0 = t_{n-1}$"
have "$\cdots = t_n$" [proof]	

ALSO/FINALLY

have "$t_0 = t_1$" [proof]	calculation register
also	"$t_0 = t_1$"
have "$\ldots = t_2$" [proof]	
also	"$t_0 = t_2$"
$\vdots$	$\vdots$
also	"$t_0 = t_{n-1}$"
have "$\cdots = t_n$" [proof]	
finally	$t_0 = t_n$

ALSO/FINALLY

have "$t_0 = t_1$" [proof]	calculation register
also	"$t_0 = t_1$"
have "$\ldots = t_2$" [proof]	
also	"$t_0 = t_2$"
$\vdots$	$\vdots$
also	"$t_0 = t_{n-1}$"
have "$\cdots = t_n$" [proof]	
finally	$t_0 = t_n$
show P	

—'fi nally' pipes fact "$t_0 = t_n$" into the proof

MORE ABOUT ALSO

➜ Works for all combinations of $=$, $\leq$ and $<$.

MORE ABOUT ALSO

- ➜ Works for all combinations of $=$, $\leq$ and $<$.
- ➜ Uses all rules declared as `[trans]`.

More about also

- Works for all combinations of $=$, $\leq$ and $<$.
- Uses all rules declared as `[trans]`.
- To view all combinations in Proof General:

  Isabelle/Isar $\rightarrow$ Show me $\rightarrow$ Transitivity rules

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"

have "$\ldots \odot r_2$" [proof]

also $\Longleftarrow$

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"
have "$\ldots \odot r_2$" [proof]
also $\Longleftarrow$

Anatomy of a [trans] rule:

➜ Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$

calculation = "$l_1 \odot r_1$"

have "$\ldots \odot r_2$" [proof]

also $\Longleftarrow$

Anatomy of a [trans] rule:

- Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"
have "$\ldots \odot r_2$" [proof]
also $\Longleftarrow$

Anatomy of a [trans] rule:

- Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- pure transitivity: $[\![a = b; b = c]\!] \Longrightarrow a = c$

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"

have "$\ldots \odot r_2$" [proof]

also $\Longleftarrow$

Anatomy of a [trans] rule:

- Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- pure transitivity: $[\![a = b; b = c]\!] \Longrightarrow a = c$
- mixed: $[\![a \leq b; b < c]\!] \Longrightarrow a < c$

Desiging [trans] Rules

calculation = "$l_1 \odot r_1$"

have "$\ldots \odot r_2$" [proof]

also $\Longleftarrow$

Anatomy of a [trans] rule:

- Usual form: plain transitivity $\llbracket l_1 \odot r_1; r_1 \odot r_2 \rrbracket \Longrightarrow l_1 \odot r_2$
- More general form: $\llbracket P\ l_1\ r_1; Q\ r_1\ r_2; A \rrbracket \Longrightarrow C\ l_1\ r_2$

Examples:

- pure transitivity: $\llbracket a = b; b = c \rrbracket \Longrightarrow a = c$
- mixed: $\llbracket a \le b; b < c \rrbracket \Longrightarrow a < c$
- substitution: $\llbracket P\ a; a = b \rrbracket \Longrightarrow P\ b$

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"

have "$\ldots \odot r_2$" [proof]

also $\Longleftarrow$

Anatomy of a [trans] rule:

- Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- pure transitivity: $[\![a = b; b = c]\!] \Longrightarrow a = c$
- mixed: $[\![a \leq b; b < c]\!] \Longrightarrow a < c$
- substitution: $[\![P\ a; a = b]\!] \Longrightarrow P\ b$
- antisymmetry: $[\![a < b; b < a]\!] \Longrightarrow P$

DESIGING [TRANS] RULES

calculation = "$l_1 \odot r_1$"
have "$\ldots \odot r_2$" [proof]
also $\Longleftarrow$

Anatomy of a [trans] rule:

- ➜ Usual form: plain transitivity $[\![l_1 \odot r_1; r_1 \odot r_2]\!] \Longrightarrow l_1 \odot r_2$
- ➜ More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- ➜ pure transitivity: $[\![a = b; b = c]\!] \Longrightarrow a = c$
- ➜ mixed: $[\![a \leq b; b < c]\!] \Longrightarrow a < c$
- ➜ substitution: $[\![P\ a; a = b]\!] \Longrightarrow P\ b$
- ➜ antisymmetry: $[\![a < b; b < a]\!] \Longrightarrow P$
- ➜ monotonicity: $[\![a = f\ b; b < c; \bigwedge x\ y.\ x < y \Longrightarrow f\ x < f\ y]\!] \Longrightarrow a < f\ c$

DEMO

WE HAVE SEEN TODAY ...

- ➜ Recdef
- ➜ More induction
- ➜ Well founded orders
- ➜ Well founded recursion
- ➜ Calculations: also/finally
- ➜ [trans]-rules

EXERCISES

➜ Define a predicate **sorted** over lists

➜ Show that **sorted (quicksort** xs**)** holds

➜ Look at `http://isabelle.in.tum.de/library/HOL/Wellfounded_Recursion.html`

➜ Show that in groups, the left-one is also a right-one: $x \cdot 1 = x$
(you can use the right_inv lemma from the demo)

➜ Take an algebra textbook and formalize a simple theorem over groups in Isabelle.