Slide 1

NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications

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

Slide 2

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

Slide 3

Last Week

- ➜ Constructive Logic & Curry-Howard-Isomorphism
- ➜ The Coq System
- ➜ The HOL4 system
- ➜ Before that: datatypes, recursion, induction

Slide 4

General Recursion

The Choice

- ➜ Limited expressiveness, automatic termination
  - primrec
- ➜ High expressiveness, prove termination manually
  - recdef

RECDEF — EXAMPLES

Slide 5

consts sep :: "'a × 'a list ⇒ '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 × nat ⇒ 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

Slide 6

- ➜ 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

Slide 7

- ➜ 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**:

$$\llbracket \bigwedge a.\ P\ a\ [];$$
$$\bigwedge a\ w.\ P\ a\ [w]$$
$$\bigwedge a\ x\ y\ zs.\ P\ a\ (y\#zs) \Longrightarrow P\ a\ (x\#y\#zs);$$
$$\rrbracket \Longrightarrow P\ a\ xs$$

TERMINATION

Slide 8

Isabelle tries to prove termination automatically

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

Slide 9

DEMO

Slide 10

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:
  wfrec $:: (\alpha \times \alpha)$ set $\Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta)$
- ➜ $(\alpha \times \alpha)$ set a well founded order, decreasing with execution

Slide 11

HOW DOES RECDEF WORK?

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

Because we want the recursion equations to hold.

Example:

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

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

Slide 12

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:

$$\frac{\mathsf{wf}\ r \quad \bigwedge x.\ (\forall y <_r x. P y) \Longrightarrow P x}{P a}$$

Alternative definition (equivalent):

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

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

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

Slide 13

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
- $A <_r B = A \subset B \wedge$ finite B is well founded
- $\subseteq$ and $\subset$ in general are **not** well founded

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

Slide 14

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

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

Slide 15

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

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

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

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

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

Slide 16

Demo

Slide 17

CALCULATIONAL REASONING

Slide 18

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

Slide 19

CHAINS OF EQUATIONS

The Problem

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

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

Slide 20

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

Slide 21

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

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

DESIGING [TRANS] RULES

Slide 22

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: $[\![a = f\ b; b < c; \bigwedge x\ y.\ x < y \Longrightarrow f\ x < f\ y]\!] \Longrightarrow a < f\ c$

Slide 23

DEMO

WE HAVE SEEN TODAY ...

Slide 24

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

EXERCISES

Slide 25

- 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.