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
  - \texttt{primrec}
- High expressiveness, prove termination manually
  - \texttt{recdef}
The definition:
- one parameter
- free pattern matching, order of rules important
- termination relation
  (measure sufficient for most cases)
- must decrease for each recursive call
- must be well founded
- Generates own induction principle

Example sep.induct:

\[
\begin{align*}
V \ a : P a &; \\
V \ aw : P a [ w ] &; \\
V \ ax y zs : P a ( y \# zs ) &\implies P a ( x \# y \# zs ); \\
&\implies P a x s
\end{align*}
\]

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:

\[
\text{recdef quicksort "measure length"}
\]

For exploration:
- allow failing termination proof
- recdef (permissive) quicksort "measure length"
- termination conditions as assumption in simp and induct rules
How does recdef work?

We need: general recursion operator

something like: $\text{rec } F = F (\text{rec } F)$

($F$ stands for the recursion equations)

Example:

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

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

How does recdef work?

Why $\text{rec } F = F (\text{rec } F)$?

Because we want the recursion equations to hold.

Example:

\[
\begin{align*}
F & \equiv \lambda g, \lambda n' . \text{case } n' \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow g n \\
f & \equiv \text{rec } F \\
f 0 & = \text{rec } F 0 \\
\ldots & = F (\text{rec } F) 0 \\
\ldots & = (\lambda g, \lambda n' . \text{case } n' \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow g n) (\text{rec } F) 0 \\
\ldots & = (\text{case } 0 \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow \text{rec } F n) \\
\ldots & = 0
\end{align*}
\]

Well founded orders

Definition

$\prec$, is well founded if well founded induction holds

$\text{wf } r \equiv \forall P. (\forall x . (\forall y < r x . P y) \rightarrow P x) \rightarrow (\exists x . P x)$

Well founded induction rule:

\[
\frac{\text{wf } r \quad \forall x . (\forall y < r x . P y) \rightarrow P x}{P u}
\]

Well founded orders: Examples

Alternative definition (equivalent):

there are no infinite descending chains, or (equivalent):

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

\[
\begin{align*}
\text{min } r Q x & \equiv \forall y \in Q . y \not< r x \\
\text{wf } r & \equiv (\forall Q \neq \emptyset . \exists m \in Q . \text{min } r Q m)
\end{align*}
\]
Well Founded Orders: Examples

- $<$ on $\mathbb{N}$ is well founded
  - well founded induction = complete induction
- $>$ and $\leq$ on $\mathbb{N}$ are not well founded
- $x <_r y = x \text{ dvd } y \land x \neq 1$ on $\mathbb{N}$ is well founded
  - the minimal elements are the prime numbers
- $(a, b) <_r (x, y) = a <_1 x \lor a = x \land b <_1 y$ is well founded
  - if $<_1$ and $<_2$ are
- $A <_r B = A \subseteq B \land \text{ finite } B$ 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: $\text{rec } F = F (\text{rec } F)$ not possible

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

Cut:
- only do recursion if parameter decreases wrt $R$
- otherwise: abort
- arbitrary $\vdash a$
  - cut $\vdash (\alpha \Rightarrow \beta) \Rightarrow (\alpha \times \alpha) \Rightarrow \alpha \Rightarrow (\alpha \Rightarrow \beta)$
  - cut $G R x \equiv \lambda y. \text{ if } (y, x) \in R \text{ then } G y \text{ else arbitrary}$

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

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

$$\text{adm}_{\text{wf}} R F \equiv \forall f g x. (\forall z. (z, x) \in R \rightarrow f z = g z) \rightarrow F f x = F g x$$

Definition of $\text{wfrec}$: again first by induction, then by epsilon

$$\forall z. (z, x) \in R \rightarrow (z, g z) \in \text{wfrec}_R R F$$

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

More: John Harrison, *Inductive definitions: automation and application*
**Slide 17**

**Calculational Reasoning**

**The Goal**

\[ x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1}) \]
\[ \ldots = 1 \cdot x \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \]
\[ \ldots = (x^{-1})^{-1} \cdot x^{-1} \]
\[ \ldots = 1 \]

Can we do this in Isabelle?

- Simplifier: too eager
- Manual: difficult in apply style
- Isar: with the methods we know, too verbose

---

**Slide 19**

**Chains of Equations**

**The Problem**

\[ a = b \]
\[ \ldots = c \]
\[ \ldots = d \]

Shows \( a = d \) by transitivity of =

Each step usually nontrivial (requires own subproof)

**Solution in Isar:**

- Keywords also and finally to delimit steps
- \ldots: 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 \( "t_0 = t_1" \)
- Also
- Have "\( \ldots = t_2 \)" [proof]
- Also \( "t_0 = t_2" \)
- Finally
- Show \( P \)
- "Finally" pipes fact "\( t_0 = t_n \)" into the proof
MORE ABOUT ALSO

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

- Uses all rules declared as $\text{[trans]}$.

Slide 21

- To view all combinations in Proof General:  
  - Isabelle/Isar --- Show me --- Transitivity rules

Slide 22

DESIGNING [TRANS] RULES

| calculation = "$l_1 \odot r_1$" |
| have "... $\odot r_2$ [proof] |
| also $\Leftarrow$ |

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

Slide 23

DEMO

WE HAVE SEEN TODAY ...

- Recdef
- More induction
- Well founded orders
- Well founded recursion
- Calculations: also/finally
- $\text{[trans]}$-rules

Slide 24

EXERCISES
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](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.