COMP 4161

NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods

wf_rec

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

The Choice

➜ Limited expressiveness, automatic termination

- primrec

➜ High expressiveness, termination proof may fail

- fun

➜ High expressiveness, tweakable, termination proof manual

- function

NICTA

fun sep :: "'a ⇒ 'a list ⇒ 'a list"
where

"sep a (x # y # zs) = x # a # sep a (y # zs)" |
"sep a xs = xs"

fun ack :: "nat ⇒ nat ⇒ nat"
where

"ack 0 n = Suc n" |
"ack (Suc m) 0 = ack m 1" |
"ack (Suc m) (Suc n) = ack m (ack (Suc m) n)"

FUN

➜ The definiton:

- pattern matching in all parameters
- arbitrary, linear constructor patterns
- reads equations sequentially like in Haskell (top to bottom)
- proves termination automatically in many cases
  (tries lexicographic order)

➜ Generates own induction principle

➜ May have fail to prove automation:

- use **function (sequential)** instead
- allows to prove termination manually

NICTA

- Each **fun** 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**:

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

NICTA

Isabelle tries to prove termination automatically

- ➜ For most functions this works with a lexicographic termination relation.
- ➜ Sometimes not $\Rightarrow$ error message with unsolved subgoal
- ➜ You can prove automation separately.

function (sequential) quicksort **where**

quicksort [] = [] $|$

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

by pat_completeness auto

termination

by (relation “measure length”) (auto simp: less_Suc_eq_le)

function is the fully tweakable, manual version of **fun**

DEMO

How does fun/function work?

We need: general recursion operator

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

(F stands for the recursion equations)

Example:

- ➜ recursion equations: $f\ 0 = 0 \qquad f\ (\mathsf{Suc}\ n) = f\ n$
- ➜ 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 :: $(\alpha \times \alpha)\ \mathsf{set} \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta)$
- ➜ $(\alpha \times \alpha)$ set a well founded order, decreasing with execution

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

Because we want the recursion equations to hold.

Example:

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

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

NICTA

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:

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

Alternative definition (equivalent):

there are no infinite 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
- $>$ 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
- $A <_r B = A \subset B \wedge \mathsf{finite}\ B$ is well founded
- $\subseteq$ and $\subset$ in general are **not** well founded

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

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

Idea: have $\mathsf{wfrec}\ R\ F$ 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

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

Admissible recursion

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

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

Definition of wf_rec: again first by induction, then by epsilon

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

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

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

Demo