COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods

wf_rec
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
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)"
The definition:

- 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
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**:

\[
\begin{align*}
\& \wedge a. P a []; \\
\& \wedge a \, w. P a [w] \\
\& \wedge a \, x \, y \, zs. P a (y \# zs) \implies P a (x \# y \# zs) ; \\
\implies P a x s
\end{align*}
\]
Isabelle tries to prove termination automatically

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

```plaintext
function (sequential) quicksort where
quicksort [] = [] |
quicksort (x#xs) = quicksort [y ← xs.y ≤ x]@[x]@ quicksort [y ← 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: \[ rec \ F = F \ (rec \ F') \]

\[(F \text{ stands for the recursion equations)}\]

**Example:**

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

\[ rec :: ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta) \]

like above cannot exist in HOL (only total functions)

But 'guarded' form possible:

- \( \text{wfrec :: } (\alpha \times \alpha) \ \text{set} \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)) \Rightarrow (\alpha \Rightarrow \beta) \)
- \( (\alpha \times \alpha) \ \text{set a well founded order, decreasing with execution} \)
Why \( rec \, F = F \,(rec \, F) \) ?

Because we want the recursion equations to hold.

Example:

\[
F \equiv \lambda g. \lambda n'. \text{case } n' \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow g \, n
\]

\[
f \equiv \text{rec } F
\]

\[
f \, 0 = \text{rec } F \, 0
\]

\[
\ldots = F \,(\text{rec } F) \, 0
\]

\[
\ldots = (\lambda g. \lambda n'. \text{case } n' \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow g \, n) \,(\text{rec } F) \, 0
\]

\[
\ldots = (\text{case } 0 \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow \text{rec } F \, n)
\]

\[
\ldots = 0
\]
**Definition**

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

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

**Well founded induction rule:**

$$\begin{align*}
\text{wf } r \land \forall x. (\forall y <_r x. P y) \rightarrow P x \\
P a
\end{align*}$$

**Alternative definition** (equivalent):

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

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

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

- $<$ on $\mathbb{N}$ is well founded
  well founded induction = complete induction
- $>$ and $\leq$ on $\mathbb{N}$ are not well founded
- $x <_r y = x \dvd y \land x \neq 1$ 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
- $A <_r B = A \subset B \land \text{finite } B$ is well founded
- $\subseteq$ and $\subset$ in general are not well founded

More about well founded relations: Term Rewriting and All That
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 :: $\alpha$

\[
\text{cut} :: (\alpha \Rightarrow \beta) \Rightarrow (\alpha \times \alpha) \Rightarrow \alpha \Rightarrow (\alpha \Rightarrow \beta)
\]

$\text{cut } G R x \equiv \lambda y. \text{if } (y, x) \in R \text{ then } G y \text{ else arbitrary}$

\[
\text{wf } R \quad \Rightarrow \quad \text{wfrec } R F x = F (\text{cut } (\text{wfrec } R F) R x) x
\]
Admissible recursion

→ recursive call for \( x \) only depends on parameters \( y <_R x \)
→ describes exactly one function if \( R \) is well founded

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

Definition of \( \text{wf}_\text{rec} \): again first by induction, then by epsilon

\[
\forall z. \ (z, x) \in R \implies (z, g \ z) \in \text{wfrec}_\text{rel} \ R \ F \implies (x, F \ g \ x) \in \text{wfrec}_\text{rel} \ R \ F
\]

\[
\text{wfrec} \ R \ F \ x \equiv \text{THE} \ y. \ (x, y) \in \text{wfrec}_\text{rel} \ R \ (\lambda f \ x. \ F \ (\text{cut} \ f \ R \ x) \ x)
\]

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