fun

General Recursion

The Choice

- Limited expressiveness, automatic termination
  - primrec
- High expressiveness, termination proof may fail
  - fun
- High expressiveness, tweakable, termination proof manual
  - function

fun — examples

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)"

Content

Rough timeline

1

fun

Slide 1

Slide 3

Slide 2

Slide 4
fun

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 fail to prove termination:
- use function (sequential) instead
- allows you to prove termination manually

Generates own induction principle

Each fun definition induces an induction principle

For each equation:
show P holds for lhs, provided P holds for each recursive call on rhs

Example sep.induct:
- \( \forall a. P\ a\ a\ \llbracket \\rrbracket \]
- \( \forall a. P\ a\ \llbracket \\rrbracket \]
- \( \forall a \ x \ y \ z\ x\ y\ z a. P\ a\ (y\ #\ z\ a) \Rightarrow P\ a\ (x\ #\ y\ #\ z a); \]
- \( \Rightarrow P\ a\ z\ x\ a\)

Termination

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 \( \llbracket \\rrbracket = \llbracket \rrbracket \]
quicksort \( \llbracket x\ y\ z a \rrbracket = \llbracket y \llbracket y \llbracket y\ z\ x\ a \rrbracket \rrbracket \]
quicksort \( \llbracket y \llbracket y \llbracket y\ z\ x\ a \rrbracket \rrbracket \]
by patcompleteness auto
termination
by (relation "measure length") (auto simp: lessSucle)

function is the fully tweakable, manual version of fun

 termination by (relation "measure length") (auto simp: lessSucle)
Recall `primrec`:
- defined one recursion operator per datatype
- inductive definition of its graph \((x, f x) \in G\)
- prove totality: \(\forall x. \exists y. (x, y) \in G\)
- prove uniqueness: \((x, y) \in G \Rightarrow (x, z) \in G \Rightarrow y = z\)
- recursion operator: \(rec x = \text{THE } y. (x, y) \in rec\)

Can separate and defer termination proof:
- skip proof of totality
- instead derive equations of the form: \(x \in f_{\text{dom}} \Rightarrow f x = \ldots\)
- similarly, conditional induction principle
- \(f_{\text{dom}} = \text{acc } f_{\text{rel}}\)
- \(\text{acc } = \text{accessible part of } f_{\text{rel}}\)
- the part that can be reached in finitely many steps
- termination: \(\forall x. x \in f_{\text{dom}}\)
- still have conditional equations for partial functions

Three main proof methods:
- lexicographic order (default tried by `fun`)
- `size_change` (different automated technique)
- relation \(R\) (manual proof via well-founded relation)
Well Founded Orders

**Definition**

$<_r$ is well founded if well founded induction holds
\[ \text{wf} \ r \equiv \forall P. (\forall x. (\forall y <_r x. P y) \implies P x) \implies (\forall x. P x) \]

**Well founded induction rule:**
\[ \text{wf} \ r \land \land x. (\forall y <_r x. P y) \implies P x \]

**Alternative definition (equivalent):**

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

- every nonempty set has a minimal element wrt $<_r$
- $\min r Q = \forall y \in Q. y \not<_r x$

\[ \text{wf} \ r = (\forall Q \neq \emptyset. \exists m \in Q. \min r Q m) \]

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
- $(a, b) <_r (x, y) = a < x \land a = x \land b <_r y$ is well founded
  - if $<_r$ and $<_r$ are
- $A <_r B = A \subseteq B \land \text{ finite } B$ is well founded
- $\subseteq$ and $<$ in general are not well founded

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

Extracting the Recursion Scheme

**So far for termination. What about the recursion scheme?**

Not fixed anymore as in primrec.

**Examples:**

- `fun fib where`
  - fib 0 = 1 |
  - fib (Suc 0) = 1 |
  - fib (Suc (Suc n)) = fib n + fib (Suc n)

Recursion: `Suc (Suc n) ~ n, Suc (Suc n) ~ Suc n`

- `fun f where f x = (if x = 0 then 0 else f (x - 1) * 2)`

Recursion: `x \neq 0 \implies x \sim x \cdot 1`

Extracting the Recursion Scheme

**Higher Oder:**

- `datatype 'a tree = Leaf 'a | Branch 'a tree list`

  `fun treemap :: ('a => 'a) => 'a tree => 'a tree where`

  `treemap fn (Leaf n) = Leaf (fn n) |`

  `treemap fn (Branch l) = Branch (map (treemap fn) l)`

Recursion: `x \in \text{set} l \implies (fn, Branch l) \sim (fn, x)`

How to extract the context information for the call?
Extracting the Recursion Scheme

Extracting context for equations

⇒

Congruence Rules!

Recall rule if cong:

\[
\begin{align*}
& b = c; c \Rightarrow x = u; \neg c \Rightarrow y = v \\
(\text{if } b \text{ then } x \text{ else } y) = (\text{if } c \text{ then } u \text{ else } v)
\end{align*}
\]

Read: for transforming \( x \), use \( b \) as context information, for \( y \) use \( \neg b \).

In \texttt{fundef}: for recursion in \( x \), use \( b \) as context, for \( y \) use \( \neg b \).

Congruence Rules for \texttt{fundef}

The same works for function definitions.

\begin{verbatim}
declare my rule[funcong]
\end{verbatim}

\( \text{if cong already added by default} \)

Another example (higher-order):

\[
\begin{align*}
& xs = ys; \forall x. x \in \text{set} \ ys \Rightarrow f x = g x \\
& \Rightarrow \text{map } f \ xs = \text{map } g \ ys
\end{align*}
\]

Read: for recursive calls in \( f \), \( f \) is called with elements of \( xs \)

Further Reading

Alexander Krauss,

\textit{Automating Recursive Definitions and Termination Proofs in Higher-Order Logic}.


http://www4.in.tum.de/~krauss/diss/krauss_phd.pdf
We have seen today...

- General recursion with fun function
- Induction over recursive functions
- How fun works
- Termination, partial functions, congruence rules