Case study: greatest common divisor

Function definition

fun gcd :: "nat ⇒ nat ⇒ nat" where
"gcd m 0 = m" |
"gcd m n = gcd n (m mod n)"

Property of function

theorem gcd_greatest: 
"(k dvd m ∧ k dvd n ∧ (0 < m ∨ 0 < n)) → (k ≤ gcd m n)"

Proofs:
» Gcd.thy

Mathematical specification of gcd

Specification
The function gcd should have the following property:
For $m, n$ with $m \geq 0$, $n \geq 0$, $m, n$ not both zero, it holds:

$$gcd\ m\ n = \max\ \{\ k | k\ divides\ m\ and\ n\}$$

Lemma:
For $m \geq 0$, $n > 0$ we have:
$k$ divides $m$ and $n$ $\iff$ $k$ divides $n$ and $k$ divides $(m \mod n)$

Proof by structural induction:
We show:

a) gcd is correct for $n = 0$ and arbitrary $m$.

b) Induction hypothesis:
   gcd is correct for all pairs $(m, k)$ for arbitrary $k \leq n$ and $m$;
   Show:
   gcd is correct for all pairs $(m, n + 1)$ for arbitrary $m$. 
Mathematical proof of \texttt{gcd} (2)

(a) Induction base:
\[
gcd\ m\ 0
=\ m
=\ \max\ \{\ k \mid k\ \text{divides}\ m\} =\ \max\ \{\ k \mid k\ \text{divides}\ m\ and\ \emptyset\}
\]

(b) Induction step:
Assumptions: \( n \) is given.
For all pairs \((m, k)\) with \( k \leq n \) it holds: \texttt{gcd} is correct for \((m, k)\).

Show: For all \( m \) it holds: \texttt{gcd} is correct for \((m, n+1)\).
\[
gcd\ m\ (n+1)
=\ (*\ Declaration\ of\ gcd\ *)
gcd\ (n+1)\ (m\ \text{mod}\ (n+1))
=\ (*\ m\ \text{mod}\ (n+1)\ \leq\ n\ and\ induction\ hypothesis\ *)
\max\ \{\ k \mid k\ \text{divides}\ (n+1)\ and\ (m\ \text{mod}\ (n+1))\} =\ (*\ \text{Lemma}\ *)
\max\ \{\ k \mid k\ \text{divides}\ m\ and\ (n+1)\}\]
QED.

Outline

Well-definedness proofs:
- Show that there exists a well-founded relation \( wf \) on the arguments
- Show that arguments in recursive calls are smaller w.r.t. \( wf \)

What we need:
- Well-founded relations and induction
- Relations: Relations are sets in Isabelle/HOL
- Sets
  » Sections 6.1, 6.2, 6.4 of Isabelle/HOL Tutorial
5. Verifying Functions 5.3 Well-definedness of total recursive functions

Sets in HOL

Introduction
Sets in HOL differ from sets in set theory:

- All elements of a set have the same type, say \( \alpha \).
- Sets are typed: \( \alpha \text{ set} \)
- Only some values are sets in HOL.

Subsets, extensionality, equality

Subsets:

\[
\begin{align*}
& (\forall \, x. x \in A \rightarrow x \in B) \rightarrow A \subseteq B \quad \text{(subsetI)} \\
& \llbracket A \subseteq B; c \in A \rrbracket \rightarrow c \in B \quad \text{(subsetD)} \\
& (A \cup B \subseteq C) = (A \subseteq C \land B \subseteq C) \quad \text{(Un_subset_iff)}
\end{align*}
\]

Extensionality and equality of sets:

\[
\begin{align*}
& (\forall \, x. (x \in A) = (x \in B)) \rightarrow A = B \quad \text{(set_ext)} \\
& \llbracket A \subseteq B; B \subseteq A \rrbracket \rightarrow A = B \quad \text{(equalityI)} \\
& \llbracket A = B; \llbracket A \subseteq B; B \subseteq A \rrbracket \rrbracket \rightarrow P \rrbracket \rightarrow P \quad \text{(equalityE)}
\end{align*}
\]

Intersection, complement, difference

Sample deduction rules for intersection:

\[
\begin{align*}
& \llbracket c \in A; c \in B \rrbracket \rightarrow c \in A \cap B \quad \text{(IntI)} \\
& c \in A \cap B \rightarrow c \in A \quad \text{(IntD1)} \\
& c \in A \cap B \rightarrow c \in B \quad \text{(IntD2)}
\end{align*}
\]

Set complement and difference:

\[
\begin{align*}
& (c \in \neg A) = (c \notin A) \quad \text{(Compl_iff)} \\
& \neg (A \cup B) = \neg A \cap \neg B \quad \text{(Compl_Un)} \\
& A \cap (B \cap A) = \{\} \quad \text{(Diff_disjoint)} \\
& A \cup \neg A = \text{UNIV} \quad \text{(Compl_partition)}
\end{align*}
\]

Set comprehension

Subsets:

\[
(a \in \{ x. \, P \, x \}) = P \, a \quad \text{(mem_Collect_eq)}
\]

\[
\{ x. \, x \in A \} = A \quad \text{(Collect_mem_eq)}
\]

Some simple facts:

\[
\text{lemma } "\{ x. \, P \, x \cup x \in A \} = \{ x. \, P \, x \} \cup A"
\]

\[
\text{lemma } "\{ x. \, P \, x \rightarrow Q \, x \} = \neg\{ x. \, P \, x \} \cup \{ x. \, Q \, x \}"
\]

More convenient syntax, example:

\[
\{ p \ast q \mid p, q \, p \in \text{prime} \land q \in \text{prime} \}
= \{ z. \, \exists \, p, q \, z = p \ast q \land p \in \text{prime} \land q \in \text{prime} \}
\]
Binding operators

Universal and existential quantification:

\[
\begin{align*}
(\forall x. x \in A \Rightarrow P x) & \Rightarrow \forall x \in A. P x & \text{(ball)} \\
\forall x \in A. P x; x \in A & \Rightarrow P x & \text{(bspec)} \\
\exists x \in A. P x & \Rightarrow \exists x \in A. P x & \text{(bexI)} \\
\exists x \in A. P x; \land. x. [x \in A; P x] & \Rightarrow Q \Rightarrow Q & \text{(bexE)}
\end{align*}
\]

Unions over parameterized sets, written $\bigcup x \in A. B x$. There is one basic law and two natural deduction rules:

\[
\begin{align*}
(b \in (\bigcup x \in A. B x)) = (\exists x \in A. b \in B x) & \quad \text{(UN_iff)} \\
\forall a \in A; b \in B a & \Rightarrow b \in (\bigcup x \in A. B x) & \text{(UN_I)} \\
\forall b \in (\bigcup x \in A. B x); \land x. [x \in A; b \in B x] & \Rightarrow R \Rightarrow R & \text{(UN_E)}
\end{align*}
\]

Relation basics

Identity and composition of relations:

\[
\begin{align*}
\text{Id} & \equiv \{ p. \exists x. p = (x, x) \} & \text{(Id_def)} \\
R \circ S & \equiv \{(x, z). \exists y. (x, y) \in S \land (y, z) \in R \} & \text{(rel_comp_def)} \\
R \circ \text{Id} & = R & \text{(R_O_Id)} \\
R' \subseteq R; S' \subseteq S & \Rightarrow R' \circ S' \subseteq R \circ S & \text{(rel_comp_mono)}
\end{align*}
\]

The converse or inverse of a relation exchanges the operands:

\[
((a, b) \in r^{-1}) = ((b, a) \in r) & \text{(converse_iff)}
\]

Here is a typical lemma proved about converse and composition:

**lemma converse_rel_comp:** "$(r \circ s)^{-1} = s^{-1} \circ r^{-1}$"
5. Verifying Functions

5.3 Well-definedness of total recursive functions

Inverse image

Let

- $r$ of type $\alpha \times \alpha$ and
- $f$ be a function of type $\beta \Rightarrow \alpha$

The inverse image of $r$ w.r.t. to $f$ is:

$\text{inv}_\text{image} \ r \ f \equiv \{(x, y). (f x, f y) \in r\}$  \hspace{1cm} (inv_image_def)

Remark
Inverse images are helpful for defining new well-founded relation from a known well-founded relation $r$.

Well-founded relations

Intuitively, a relation $<$ is well-founded if every descending chain of elements is finite; i.e., there is no infinite descending chain of elements $a_0, a_1, \ldots$:

$\cdots < a_2 < a_1 < a_0$

Isabelle/HOL provides a predicate $\text{wf}$ that asserts that a relation is well-founded; e.g., for $\text{less\_than} :: (\text{nat} \times \text{nat})$ set:

$((x, y) \in \text{less\_than}) = (x < y)$  \hspace{1cm} (less\_than_iff)

Problem
It can be difficult to prove $\text{wf} \ r$ for a relation $r$.

Proving well-foundedness

Proof method
To proof that a relation $r$ is well-founded, show that it is an inverse image of a well-founded relation w.r.t. to some “measure” function $f$.

$\text{theorem} \ \text{wf\_inv\_image}: \ "\text{wf} \ r \Rightarrow \text{wf} \ (\text{inv\_image} \ r \ f)"

Example
Let

definition shorter :: "('a list \times 'a list) set" where
"shorter = \{(xl,yl). \text{length} \ xl < \text{length} \ yl\}"

Proof:
lemma "shorter = \text{inv\_image} \ \text{less\_than} \ \text{length}"
5. Verifying Functions

5.3 Well-definedness of total recursive functions

Well-founded induction

Induction proofs based on well-founded relations
Well-founded relations $r$ can be used for induction proofs:
A property holds for all elements if we can show that it holds for an element $x$ assuming it holds for all predecessors.

In Isabelle/HOL:

\[
\begin{align*}
\text{⟦} & \text{wf } r, \text{ } \land \text{ } x. \forall y. (y, x) \in r \rightarrow P y \Rightarrow P x \text{ ] } \Rightarrow \text{ P a} \\
\text{(wf_induct)}
\end{align*}
\]

Remark
Note that in well-founded inductions, there is no explicit induction base.

Case study: Quicksort

We analyse a functional version of the quicksort algorithm.

```plaintext
function qsort :: 'a :: linorder list ⇒ 'a list where
  "qsort [] = []"
| "qsort (p#l) = qsort (qsplit (op <) p l) @ p # qsort (qsplit (op ≥) p l)"
```

where `linorder` is a type class supporting "<" and "≥" and

```plaintext
primrec qsplit :: "('a ⇒ 'a ⇒ bool) ⇒ 'a :: linorder ⇒ 'a list ⇒ 'a list" where
  "qsplit cr p [] = []"
| "qsplit cr p (h # t) = (if cr h p then h # qsplit cr p t else qsplit cr p t)"
```

Properties to prove

1. Well-definedness of `qsort`
2. (Well-definedness of `qsplit`)
3. Sortedness of result
4. Result is a permutation of input list
5. Verifying Functions  5.4 Case study: Quicksort

Specifying sortedness

Sortedness:

\[
\text{fun qsorted :: } 'a :: \text{linorder list } \Rightarrow \text{bool} \text{ where}
\]
\[
\begin{align*}
\text{"qsorted } [] & \text{ = True"} \\
\text{"qsorted } [x] & \text{ = True"} \\
\text{"qsorted } (a \# b \# l) & \text{ = (b } \geq \text{ a } \land \text{ qsorted } (b \# l))"
\end{align*}
\]

lemma qsort_sorts: "qsorted (qsort xl)"

Specifying the permutation property

Permutation using a multiset abstraction:

\[
\text{primrec count :: } 'a \text{ list } \Rightarrow 'a \Rightarrow \text{nat} \text{ where}
\]
\[
\begin{align*}
\text{"count } [] & \text{ = (} \lambda x. 0\text{)"} \\
\text{"count } (h \# t) & \text{ = (count } t \text{ ) (} h := \text{ count } t \text{ h + 1)}"
\end{align*}
\]

lemma qsort_preserves: "count (qsort xl) = count xl"

Verification

The proofs for the properties are presented step by step in the lecture.

Resulting theory with proofs:
» Quicksort.thy