Alfa/Agda

Thierry Coquand

Formalization and answers by Thierry Coquand
<coquand@cs.chalmers.se>.

7.1 Statement

$prime\,p \rightarrow noether\,A\,(multiple\,p) \rightarrow isNotSquare\,p$

7.2 Definitions

Definition of noether

$noether\,(A \in Set, R \in rel\,A) \in Type$
$noether\,A\,R \equiv$
$(P \in pred\,A) \rightarrow (x \in A) \rightarrow ((y \in A) \rightarrow (R\,y\,x \rightarrow P\,y) \rightarrow P\,x) \rightarrow (x \in A) \rightarrow P\,x$

Definition of multiple

$multiple\,(p \in A) \in rel\,A$
$multiple\,p \equiv \lambda\,x\,y \rightarrow (p\cdot\,x) == y$

Definition of prime

$prime \in pred\,A$
$prime \equiv \lambda\,p \rightarrow (x, y \in A) \rightarrow (p \mid (x\cdot\,y)) \rightarrow (p \mid x) \vee (p \mid y)$

Definition of isNotSquare

$isNotSquare \in pred\,A$
$isNotSquare \equiv \lambda\,p \rightarrow (x, y \in A) \rightarrow \neg\,((p\cdot\,square\,x) == square\,y)$

7.3 Proof

File `noether.alfa`

some general logical concepts and principles

$rel\,(A \in Set) \in Type$
$rel\,A \equiv A \rightarrow A \rightarrow Set$

$pred\,(A \in Set) \in Type$
$pred\,A \equiv A \rightarrow Set$

$exists \in (A \in Set, B \in A \rightarrow Set) \rightarrow Set$
$exists \equiv \lambda\,A\,B \rightarrow \texttt{data}\,\{\,\texttt{Witness}\,(u \in A, v \in B\,u)\,\}$

$existsElim \in (h1 \in exists\,A\,B, h2 \in (x \in A) \rightarrow B\,x \rightarrow C) \rightarrow C$
$(A \in Set, B \in A \rightarrow Set, C \in Set)$

F. Wiedijk (Ed.): The Seventeen Provers of the World, LNAI 3600, pp. 50–54, 2006.

$existsElim \equiv \lambda\, h1\, h2 \rightarrow \texttt{case}\ \ h1\ \texttt{of}\ \{\ \texttt{Witness}\, u\, v \rightarrow h2\, u\, v\ \}$

$\wedge\ (A, B \in Set) \in Set$
$A \wedge B \equiv \texttt{data}\ \{\ \texttt{Pair}\, (a \in A, b \in B)\ \}$

$andElimLeft \in (A \wedge B) \rightarrow A \qquad (A, B \in Set)$
$andElimLeft \equiv \lambda\, h \rightarrow \texttt{case}\ \ h\ \texttt{of}\ \{\ \texttt{Pair}\, a\, b \rightarrow a\ \}$

$andElimRight \in (A \wedge B) \rightarrow B \qquad (A, B \in Set)$
$andElimRight \equiv \lambda\, h \rightarrow \texttt{case}\ \ h\ \texttt{of}\ \{\ \texttt{Pair}\, a\, b \rightarrow b\ \}$

$\vee\ (A, B \in Set) \in Set$
$A \vee B \equiv \texttt{data}\ \left\{ \begin{array}{l} \texttt{Inl}\, (a \in A) \\ \texttt{Inr}\, (b \in B) \end{array} \right\}$

$orElim \in (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow (A \vee B) \rightarrow C \qquad (A, B, C \in Set)$
$orElim \equiv \lambda\, h1\, h2\, h3 \rightarrow \texttt{case}\ \ h3\ \texttt{of}\ \left\{ \begin{array}{l} \texttt{Inl}\ a \rightarrow h1\, a \\ \texttt{Inr}\ b \rightarrow h2\, b \end{array} \right\}$

$noether\ (A \in Set, R \in rel\, A) \in Type$
$noether\, A\, R \equiv$
$(P \in pred\, A) \rightarrow (x \in A) \rightarrow ((y \in A) \rightarrow (R\, y\, x \rightarrow P\, y) \rightarrow P\, x) \rightarrow (x \in A) \rightarrow P\, x$

$\bot \in Set$
$\bot \equiv \texttt{data}\ \{\}$

$\neg\ (A \in Set) \in Set$
$\neg\, A \equiv A \rightarrow \bot$

The principle of infinite descent, following Fermat

$infiniteDescent\ (A \in Set, R \in rel\, A, P \in pred\, A) \in$
$noether\, A\, R \rightarrow (x \in A) \rightarrow (P\, x \rightarrow exists\, A\, \lambda\, x1 \rightarrow R\, x1\, x \wedge P\, x1) \rightarrow (x \in A) \rightarrow \neg\ (P\, x)$
$infiniteDescent\, A\, R\, P \equiv \lambda\, h1\, h2\, x \rightarrow h1\, \lambda\, y \rightarrow \neg\ (P\, y)\ \lambda\, z\, h3\, h4 \rightarrow$
$existsElim\ (h2\, z\, h4)\ \lambda\, y\, h5 \rightarrow h3\, y\ (andElimLeft\, h5)\ (andElimRight\, h5)\ x$

File `lem.alfa`

```
import noether.alfa
```

Definition of commutative monoid; for the main proof, it will be the monoid $\mathbb{N}^*$ for multiplication

$AbMonoid\ (A \in Set, == \in rel\, A) \in Set$
$AbMonoid\, A\ (==) \equiv$
$$\texttt{sig}\ \left\{ \begin{array}{l} ref \in (x \in A) \rightarrow x == x \\ sym \in (x, y \in A) \rightarrow (x == y) \rightarrow y == x \\ trans \in (x, y, z \in A) \rightarrow (x == y) \rightarrow (y == z) \rightarrow x == z \\ ss \in A \rightarrow A \rightarrow A \\ cong \in (x1, x2, y1, y2 \in A) \rightarrow (x1 == x2) \rightarrow (y1 == y2) \rightarrow (x1 \cdot y1) == (x2 \cdot y2) \\ assoc \in (x, y, z \in A) \rightarrow (x \cdot (y \cdot z)) == ((x \cdot y) \cdot z) \\ comm \in (x, y \in A) \rightarrow (x \cdot y) == (y \cdot x) \end{array} \right\}$$

A beginning of the theory of commutative monoid

$\texttt{package}\ ThAbMonoid\ (A \in Set, == \in rel\, A, m \in AbMonoid\, A\ (==))$
$\texttt{with}$
  $\texttt{open}\ m\ \texttt{use}\ ref, sym, trans, cong, \cdot, assoc, comm$

  $square\ (x \in A) \in A$
  $square\, x \equiv x \cdot x$

$multiple\ (p\ \in\ A) \in rel\, A$
$multiple\, p \equiv \lambda\, x\, y\ \rightarrow\ (p \cdot\ x)\ ==\ y$

$congLeft \in (y == z)\ \rightarrow\ (y \cdot\ x)\ ==\ (z \cdot\ x) \qquad (x, y, z\ \in\ A)$
$congLeft \equiv \lambda\, h\ \rightarrow\ cong\, h\ (ref\, x)$

$congRight \in (y == z)\ \rightarrow\ (x \cdot\ y)\ ==\ (x \cdot\ z) \qquad (x, y, z\ \in\ A)$
$congRight \equiv \lambda\, h\ \rightarrow\ cong\ (ref\, x)\ h$

$lemma0 \in (x == z)\ \rightarrow\ (y == z)\ \rightarrow\ x == y \qquad (x, y, z\ \in\ A)$
$lemma0 \equiv \lambda\, h\, h'\ \rightarrow\ trans\, h\ (sym\, y\, z\, h')$

$lemma2 \in (x == y)\ \rightarrow\ square\, x == square\, y \qquad (x, y\ \in\ A)$
$lemma2 \equiv \lambda\, h\ \rightarrow\ cong\, h\, h$

$lemma3 \in (x \cdot\ (y \cdot\ z))\ ==\ (y \cdot\ (x \cdot\ z)) \qquad (x, y, z\ \in\ A)$
$lemma3 \equiv lemma0\, assoc\ (trans\, assoc\ (congLeft\, comm))$

$lemma4 \in (x \cdot\ (x \cdot\ square\, y))\ ==\ square\ (x \cdot\ y) \qquad (x, y\ \in\ A)$
$lemma4 \equiv trans\ (congRight\, lemma3)\ assoc$

$lemma5 \in ((p \cdot\ y1)\ ==\ y)\ \rightarrow\ (p \cdot\ (p \cdot\ square\, y1))\ ==\ square\, y \qquad (p, y, y1\ \in\ A)$
$lemma5 \equiv \lambda\, h\ \rightarrow\ trans\, lemma4\ (lemma2\, h)$

File `lem1.alfa`

`import` *lem.alfa*

Cancelative abelian monoid; $\mathbb{N}^*$ is cancelative for the multiplication

$isCancel\ (A\ \in\ Set, ==\ \in\ rel\, A, m\ \in\ AbMonoid\, A\ (==)) \in Set$
$isCancel\, A\ (==)\ m \equiv (x, y, z\ \in\ A) \rightarrow (m.ss\, z\, x == m.ss\, z\, y)\ \rightarrow\ x == y$

`package` *square* $(A\ \in\ Set, ==\ \in\ rel\, A, m\ \in\ AbMonoid\, A\ (==), cancel\ \in\ isCancel\, A\ (==)\ m)$ `with`
`open` m `use` *ref*, *sym*, *trans*, *cong*, $\cdot$, *assoc*, *comm*
`open` $ThAbMonoid\, A\ (==)\ m$ `use` *square*, *multiple*

$lemma1 \in ((p \cdot\ u)\ ==\ w)\ \rightarrow\ ((p \cdot\ v)\ ==\ w)\ \rightarrow\ u == v \qquad (p, u, v, w\ \in\ A)$
$lemma1 \equiv \lambda\, h2\, h3\ \rightarrow\ cancel\, u\, v\, p\ ((ThAbMonoid\, A\ (==)\ m).lemma0\ (p \cdot\ u)\ (p \cdot\ v)\ w\, h2\, h3)$

$lemma2 \in ((p \cdot\ square\, x)\ ==\ square\, y)\ \rightarrow\ ((p \cdot\ y1)\ ==\ y)\ \rightarrow\ (p \cdot\ square\, y1)\ ==\ square\, x$
$(p, x, y, y1\ \in\ A)$
$lemma2 \equiv \lambda\, h1\, h2\ \rightarrow\ lemma1\ ((ThAbMonoid\, A\ (==)\ m).lemma5\, p\, y\, y1\, h2)\ h1$

$\exists \in pred\, A\ \rightarrow\ Set$
$\exists \equiv exists\, A$

$| \in rel\, A$
$| \equiv \lambda\, x\, y\ \rightarrow\ \exists\, z\ ((x \cdot\ z)\ ==\ y)$

$prime \in pred\, A$
$prime \equiv \lambda\, p\ \rightarrow\ (x, y\ \in\ A) \rightarrow (p\,|\ (x \cdot\ y))\ \rightarrow\ (p\,|\,x)\ \vee\ (p\,|\,y)$

$lemma3 \in prime\, p\ \rightarrow\ (p\,|\,square\, x)\ \rightarrow\ p\,|\,x \qquad (p, x\ \in\ A)$
$lemma3 \equiv \lambda\, h1\, h2\ \rightarrow\ orElim\, \lambda\, h\ \rightarrow\ h\, \lambda\, h\ \rightarrow\ h\ (h1\, x\, x\, h2)$

$lemma4 \in prime\, p\ \rightarrow\ ((p \cdot\ square\, x)\ ==\ square\, y)\ \rightarrow$
$\exists\, y1\ (((p \cdot\ y1)\ ==\ y)\ \wedge\ ((p \cdot\ square\, y1)\ ==\ square\, x)) \qquad (p, x, y\ \in\ A)$
$lemma4 \equiv \lambda\, h1\, h2\ \rightarrow$ `let` $\left[\begin{array}{l} rem \in p\,|\,y \\ rem \equiv lemma3\, h1\ (\texttt{Witness}\ (square\, x)\ h2) \end{array}\right.$
`in` $existsElim\, rem\, \lambda\, y1\, h3\ \rightarrow$ `Witness` $y1$ (`Pair` $h3\ (lemma2\, h2\, h3)$)

$$
\begin{array}{l}
Square \in rel\, A \\
Square \equiv \lambda\, p\, x \rightarrow \exists\, y\ ((p \cdot\ square\, x) == square\, y)
\end{array}
$$

$$
\begin{array}{l}
lemma5\ (h2 \in prime\, p, x \in A, h3 \in Square\, p\, x) \in \exists\, x1\ (((p \cdot\ x1) == x) \wedge Square\, p\, x1)\ (p \in A) \\
lemma5\, h2\, x\, h3 \equiv existsElim\, h3\, \lambda\, y\, h4 \rightarrow existsElim\ (lemma4\, h2\, h4)\ \lambda\, y1\, h5 \rightarrow \\
\quad \mathtt{let} \left[\begin{array}{l} rem1 \in (p \cdot\ y1) == y \\ rem1 \equiv andElimLeft\, h5 \\ rem2 \in (p \cdot\ square\, y1) == square\, x \\ rem2 \equiv andElimRight\, h5 \end{array} \right. \\
\quad \mathtt{in}\ existsElim\ (lemma4\, h2\, rem2)\ \lambda\, x1\, h6 \rightarrow \mathtt{let} \left[\begin{array}{l} rem3 \in (p \cdot\ x1) == x \\ rem3 \equiv andElimLeft\, h6 \\ rem4 \in (p \cdot\ square\, x1) == square\, y1 \\ rem4 \equiv andElimRight\, h6 \end{array} \right. \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{in}\ \mathtt{Witness}\ x1\ (\mathtt{Pair}\ rem3\ (\mathtt{Witness}\ y1\ rem4))
\end{array}
$$

$$
\begin{array}{l}
isNotSquare \in pred\, A \\
isNotSquare \equiv \lambda\, p \rightarrow (x, y \in A) \rightarrow \neg\ ((p \cdot\ square\, x) == square\, y)
\end{array}
$$

$$
\begin{array}{l}
theorem\ (p \in A) \in prime\, p \rightarrow noether\, A\ (multiple\, p) \rightarrow isNotSquare\, p \\
theorem\, p \equiv \\
\quad \lambda\, h1\, h2 \rightarrow \mathtt{let} \left[\begin{array}{l} rem \in (x \in A) \rightarrow \neg\ (Square\, p\, x) \\ rem \equiv infiniteDescent\, A\ (multiple\, p)\ (Square\, p)\ h2\ (lemma5\, h1) \end{array} \right. \\
\qquad\qquad \mathtt{in}\ \lambda\, x\, y\, h3 \rightarrow rem\, x\ (\mathtt{Witness}\ y\, h3)
\end{array}
$$

This is the main result; it applies to the case where A is $\mathbb{N}^*$ and p is 2, but it shows as well that any prime cannot be a square of a rational

7.4 System

What is the home page of the system?

`<http://www.cs.chalmers.se/~catarina/agda/>`

What are the books about the system? No book directly on the system, but it is inspired by:

> Nordstrom, Petersson, Smith, *Programming in Type Theory*, Oxford University Press.

What is the logic of the system? Described in Per Martin-Löf, 'An intuitionistic theory of types', 1972. It can be found in:

> G. Sambin and J. Smith (eds.), *25 years of constructive type theory*, Oxford University Press.

This is extended by general data type declaration and case notation.

What is the implementation architecture of the system? Described in `<http://www.cs.chalmers.se/~catarina/agda/>`.

What does working with the system look like? It feels like programming in a functional language with dependent types. The construction of the proof/program can be done interactively with place-holders/meta-variables. Actually the syntax is very close to the functional language Cayenne, designed by Lennart Augustsson, <http://www.cs.chalmers.se/~augustss/cayenne/>.

What is special about the system compared to other systems? Identification of proofs and functional programs with dependent types + meta-variables.

What are other versions of the system? A first version (1990) was written by Thierry Coquand (caml) and Lennart Augustsson (C). Then (1992) version Lena Magnusson (SML) and Johan Nordlander (C).

Alfa is the interface of the system. It has been written by Thomas Hallgren. There is also an Emacs interface, following ideas of Dan Synek, written by Catarina Coquand,

Who are the people behind the system? The meta-variable idea comes from Bengt Nordstrom, refined later by Thierry Coquand and Lena Magnusson. The previous version Half was designed by Thierry Coquand (gofer) and Dan Synek (C). The actual version is designed by Catarina Coquand (haskell).

What are the main user communities of the system? Peter Hancock and Anton Setzer.

What large mathematical formalizations have been done in the system? Localic version of Hahn-Banach was done by Jan Cederquist in Half.