Mechanised theorem proving:

exponents 3 and 4 of Fermat's last theorem using Isabelle

Roelof Oosterhuis
Supervisors: Prof.dr. J. Top and prof.dr. W.H. Hesselink

University of Groningen

Department of Mathematics
August 2007

Contents

Chapter 1

Introduction

The simplicity of mathematics is probably the main reason why many people consider it as extremely difficult. Like a chess game, the rules are very clear: in order to prove your theorem you only have a small number of 'inference steps' you can apply. These steps bring your theorem in a new, hopefully better, state. The goal is to end up, after a sequence of steps, in a state only consisting of commonly accepted basic facts: the axioms. When you succeed, you win. When you don't, your claim will be no more than a 'conjecture'; and if someone even finds a counterexample you better stop playing - 'checkmate'.

In November 1997 the chess computer 'Deep Blue' wins his final match against the reigning world champion Gerry Kasparov after 19 moves. Surprisingly, 'Deep Blue' was not just such a good player because of the billions of possible moves it could calculate in advance, but it would never have beaten Kasparov if it would not have been highly educated by its programmers, who learned it loads of standard openings and other knowledge from the best chess books. That means that even when 'it is only about calculating with a limited number of possibilities each time', the human brain is (still) more 'clever' than a supercomputer.

In the struggle for making the computer a 'good mathematician' - the underlying subject of this thesis - comparable results can be observed. Complex proofs with many case distinctions can sometimes be a 'piece of cake' for modern theorem provers. On the other hand, proofs of easy statements like *'for all integers*[1] x *we have* $x^2 \geq x$*'* can only be proven by the computer once is it told to distinguish between the cases $x \leq 0$ and $x \geq 1$. Nothing special for a mathematician, but quite creative from a computer's point of view. Anyhow, a computer appears to be quite useful for a mathematician: it can do the bookkeeping, store sub-results without making mistakes, perform calculations, etc. - leaving the 'intelligent' work to the mathematician. Nevertheless, the mathematics has to be expressed in terms the computer is able to deal with: the 'formalisation' of mathematics.

[1] The numbers $\ldots, -2, -1, 0, 1, 2, 3, \ldots$

1.1 This research

This thesis describes a formalisation of a proof of the cases $n = 3$ and 4 of *Fermat's last theorem* (FLT), using the proof assistant *Isabelle*. The formalisation of the *general* FLT, stating that for *all* natural numbers $n > 2$ and all integers x, y, z we have

$$x^n + y^n = z^n \Rightarrow xyz = 0,$$

is one of the major challenges in the formalisation of mathematics: it even appears as first problem on the list of 'ten challenging research problems for computer science' of prof. dr. Jan Bergstra [20, p. 53] [13].
In 1993 Andrew Wiles claimed to have proven the theorem, but the proof turned out to have an important gap. One year later Wiles was able fix this and although there are only a few people who understand his whole proof it is generally assumed to be correct. Anyhow, a formalisation of this proof would really strengthen this belief and would make the proof more accessible and understandable.
The main reasons to perform this experiments with the cases $n = 3$ and 4 are the following:

- Wiles' proof only concerns the cases where n is a prime ≥ 5. To complete the proof these 'small' results are really necessary.
- It should give better insight in usability of proof assistants in general and Isabelle in particular to formalise mathematical proofs, with a focus on number theory.

For the sake of completeness it should be mentioned that the case $n = 4$ already has been formalised (with the proof assistant *Coq* [5]) [8]. Nevertheless, this research starts with formalising this case (again), particularly to get more experience in the formalisation of mathematics and working with Isabelle, before starting with the more difficult case $n = 3$.

1.2 Outline of the thesis

This thesis consists, more or less naturally, of the following four parts:

- An introduction to the concept of a 'mathematical proof' and to the proofs of FLT and special cases of it in particular;
- secondly, a brief overview of the development of proof assistants in general and Isabelle in particular;
- next, a report of the 'case study' (FLT3&4 in Isabelle), containing several dilemmas and problems one might encounter in the formalisation of mathematics;
- and finally some remarks and conclusions, partly based on the research results and partly based on the opinion and experiences of the author.

The first and last part should be readable by non-mathematicians, as well as the majority of the other parts.

Chapter 2

Mathematical proofs and Fermat's last theorem

This chapter contains a brief overview of the main topics in the discussion about the foundations of mathematics. This is meant as an introduction to give the reader a better idea of what a 'proof' is, and not as an historically nor scientifically complete description.
The same holds for the second part, where the reader can find a short introduction to FLT and a global sketch of the several proof attempts.

2.1 The history of mathematical proofs

Mathematical proofs have a long history. One of the oldest proofs is a proof of the Pythagorean theorem by the Hindu priest Apastamba (ca. 600 BC, according to [26]). However, the ancient Greeks, in particular Thales (ca. 624-547 BC), are usually called the 'inventors' of mathematical proofs. Together with his student Pythagoras (ca. 569-475 BC) Thales demonstrated several geometrical facts. In a fictive dialogue between Socrates and a slave, Plato suggests that a proof should convince 'ordinary' people of the truth of mathematical facts. Less than a century later the famous Aristotle, usually considered as the founder of classical logic, writes about his axiomatic view of mathematics. Mathematics should start with some (generally accepted) axioms which can be used to prove statements about mathematical objects. Those objects can be defined from previously defined objects, starting with some primitive objects. This is also the way Euclid builds up his famous works about geometry and number theory: it starts with a few definitions, 'common notions' and postulates, and from that, all facts (propositions) are proven.

Formalisation, Hilbert's ideal

Euclids work can be seen as a first important step in the formalisation of mathematics. It does not just give a justification of the results, by means of a description of the methods that are applied, it also describes the 'foundation' of the theory.

These are the two main aspects of formalisation: analysing a proof and giving a foundation. Nowadays, *formalisation* can be described as expressing statements and proofs in a usually small and simple formal language with strict rules of grammar and unambiguous semantics [12]. Once mathematics is fully formalised, there would be (almost) no room for vagueness or mistakes.

One of the main advocators of the formalisation of mathematics is the German mathematician David Hilbert. In his point of view the ultimate goal of mathematics is to prove all mathematical truths in a formal, consistent system. In this system 'intuition' should not play a role: it is all about the manipulation of symbols according to a consistent set of rules. Although Hilbert must have been aware of the difficulty of reaching this ideal, there even turned out to be fundamental impossibilities of such a project.

Since this and other problems are important to understand the development of the formalisation of mathematics, they will be discussed in the next paragraphs.

Problem I: Set of axioms

The interest for the axioms – and for the foundation of mathematics in general – grows fast at the end of the 19-th century. Several mathematicians try to build up mathematics in their own way. One important approach is the foundation using set theory, particularly developed by Cantor (1845–1918). In that approach all mathematical objects, for example the natural numbers, are defined in terms of sets and operations on sets. Unfortunately, it appeared to be quite difficult to define which operations and which kinds of sets were allowed. An important illustration of this is the so-called Russell paradox: define X to be the set of all sets that do not contain itself as an element ($X := \{Y \mid Y \notin Y\}$). Although this set seems to be well-defined (for a given Y it well-defined whether it is a member of X or not), it gives rise to a contradiction if one asks if X is a member of itself: if it is, then it should not have been; if it is not, then it should have been: contradiction.

Paradoxes like these give rise to discussions on how mathematics should be formalised. For example, at the end of the 19-th there are many discussions about the so-called 'axiom of choice'. Nowadays the most popular approaches are the Peano axioms for basic arithmetic and for general mathematics the set-based system of Zermelo-Fraenkel, with the axiom of choice (ZFC).

Problem II: Logical inference rules

One of the main opponents of Hilbert was the Dutch mathematician L.E.J. Brouwer. In Brouwer's point of view a mathematical statement is true if there is a mental construction that gives its evidence. Therefore the human mind is the starting point of mathematics and logical and mathematical principles are only true if there exists such a mental construction for them. According to his philosophy, this implies that the law of the 'excluded middle' and the principle of 'proof by

contradiction' should be rejected. As a result, to prove that something exists one should present a construction of such an object, rather than deriving a contradiction from the assumption that it does not exist. This approach has led to the current *constructivism* and although Brouwer's controversial way of thinking did not gain much popularity, this constructivism turned out to be quite useful at the development of computer science.

Brouwer's rejection of the law of the excluded middle was in itself not a problem for Hilbert's program, but (for the first time since Aristotle) it did give rise to a new problem: the choice for the set of logic inference rules. Nowadays constructivistic logic is still used in several parts of mathematics, although the classical logic is in general considered as the standard approach.

Problem III: Provability and consistency

The previous problems still don't make Hilbert's program impossible: in principle the mathematical society could be able to agree on the ultimate set of axioms, the logical rules and even agree on the exact definitions of the mathematical concepts. Or, if they can not, Hilbert's program could still be performed in those different 'kinds of mathematics'. Unfortunately, there appeared to be more fundamental problems of such a program.

Provability

As mentioned before, a mathematical statement is in general only accepted to be true if there is a proof of it. If, on the other hand, nobody has found a counterexample *nor* a proof of it, it is called a conjecture (or: hypothesis). One of the aims of Hilbert was to make sure the axioms of his system were *complete*, which means that all statements that are true in the model also can be proven from its axioms. However, Kurt Gödel (1906-1987) proved in 1931 that this was impossible. To be precise, he proved that any mathematical system containing the basic theory of the natural numbers (the Peano arithmetic) contained statements that were true, but not provable. As a result no 'rich enough', consistent set of axioms could be complete, which was *bad news* for Hilbert's program.

It should be mentioned that Gödel also proved a positive result: the completeness of first-order logic. This means that if a (purely) logical statement is true, then it is provable.

Consistency

Another result of Gödel is that any formal system, containing the basic theory of natural numbers, can never prove its own consistency. As a result, we would never be sure that a system like the standard set of axioms (ZFC) does not lead to a contradiction. Although it is strongly believed that ZFC is consistent, another ideal of Hilbert's program became problematic.

Conclusion

From a historical point of view, a proof is not just a justification of a claim, but it should also make the results more accessible. During the process of the formalisation of mathematics, the 'justification' part becomes more important: proofs become a series of small, technical steps within a detailed formal context. Since any mathematician will admit that understanding a proof is more than being convinced that all steps in the proof are correct, a formal proof is, in general, not enough. Therefore, in this thesis we will deal with formal proofs - as described above - and *informal* proofs: mainly a 'description' of the proof, explaining which concepts are applied and leaving out the details, which should convince the reader that the claim is correct and give insight why.

Hilbert's program, to formally prove all mathematical truths in a consistent system, turned out to be too optimistic. Even when mathematicians will be able to agree on 'the' set of axioms, logical rules and definitions, the consistency can not be proven and there will always be true statements that remain unproven.

However, the formalisation of mathematics is still important: it rules out vagueness and ambiguity and, when performed by a computer, it guarentees a high level of correctness. But it's even better: a computer is not just good at bookkeeping, it also has the opportunity to automatically construct proofs. We will return to this at the next chapter.

2.2 Fermat's last theorem

Pythagorean triples

For several reasons – which will be pointed out below – Fermat's last theorem is closely related to the integral solutions of the equation $x^2 + y^2 = z^2$. This equation is well-known from the Pythagorean theorem, which states that in a right-angled triangle the square on the hypothenuse is equal to the sum of the squares on the other sides. A right-angled triangle with integral sides is called a *Pythagorean triangle* and a triple, consisting of numbers that form such a triangle, – like $(3, 4, 5)$ – is called a *Pythagorean triple*. These triples have a long history, which starts a lot earlier than the birth of the Greek Pythagoras (approx. 530 BC), after whom the theorem is named. Some believe they already appear as ratios in triangles used in the megalithic monuments, such as the 'henges' in England and Scotland (approx. 2500 BC) or even earlier in Egyptian buildings [27]. In any case, the Babylonians must have been familiar with these Pythagorean triples. On a Babylonian clay tablet, dated between 1800 and 1650 BC, an interesting table with logically ordered Pythagorean triples can be found. One of the triples is as large as $(13500, 12709, 18541)$ and there is little doubt the Babylonians must have had a method for generating such triples [23].
The full description of such a method – the parametrisation of all (infinitely many) Pythagorean triples – can be found in the famous *Elements* of Euclid (300 BC)

and in the *Arithmetica* of Diophantus (250 AD).

Fermat

Exactly in a Latin translation by Bachet of this *Arithmetica*, about 1400 years later, the French amateur mathematician Pierre de Fermat (1601-1665) writes down his famous 'last theorem'. In the margin next to the description of the construction of Pythagorean triples he writes down that such equations do not have solutions for higher powers than 2. In modern terms: the equation $x^n + y^n = z^n$ for $n \geq 3$ has no solutions in nonzero integers x, y, z. He also mentions that he had found a remarkable proof of it, but that the margin was too narrow to write it down there [29].
Although it is highly implausible that he had a (correct) proof at all, Fermat did have one for the case $n = 4$ (on which section 4.3 is based) in which he particularly used – to let the circle go round – the construction of the Pythagorean triples.

From Euler to Wiles

In the next centuries a lot of mathematicians would be working on proofs of (parts of) FLT. Euler (1707-1783) was able to prove the case $n = 3$ (on which section 4.6 is based), but the proof differed that much from the proof of the case $n = 4$ that it left him 'no hope of extending it to a general proof for n-th powers' [29, p. 117].
In the years after Euler several cases (like $n = 5$ and $n = 7$) were proven, but the first serious progression was made by Kummer (1810-1893), who proved the theorem for all so-called 'regular primes'. Combined with some earlier results, the theorem was proven for all exponents less than 100.
Another interesting result was Mordell's conjecture (from 1922, proven in 1983 by Faltings) which connects the solutions of an algebraic equation with its associated topology. As a particular case of this result, for every $n \geq 4$ there would be at most finitely many solutions of $x^n + y^n = z^n$ in coprime integers x, y, z.
However, the ultimate proof of the general case used a different approach. An important step was the so-called Shimura-Taniyama conjecture (1955), establishing a connection between elliptic curves and modular forms: two quite different parts of mathematics. It was Ribet in 1985, inspired by Frey, who proved that, if a special case of the ST-conjecture were true, FLT follows as a direct consequence. A few years after that Andrew Wiles tried to prove that special case of the ST-conjecture and in 1993 he claimed to have finished it. Unfortunately there appeared to be an important gap in the proof, but a year later Wiles, with the support of Taylor, was able to fix it. Together with the results of Ribet this finally proved FLT.

Towards a formalisation

We can make the following conclusions for the formalisation of FLT:

- The proof of the general case will be *much* more difficult to formalise than the one for the cases $n = 4$ and $n = 3$. There is a big difference between the cases

$n = 4$ and $n = 3$ already: the first one can quite easily be understood by a first year student in mathematics; the latter one is traditionally proven using numbers of the form $a + b\sqrt{-3}$ (with a, b integers), so we need something like number rings and the complex number i. The next step, Kummer's proof, already needs much more 'mathematics', like: ideals, rings of integers and units in it, cyclotomic integers, class numbers and unique factorisation in ideals, etcetera. However, Wiles' proof is still far beyond this scope. In order to formalise it, we need to formalise huge parts of modern mathematics, like modular forms and elliptic curves and last but not least: a formalisation of Wiles' final proof: consisting of more than 100 pages of mathematics, only suitable for 'experts'.

- A formal proof *really makes sense*. Mathematicians simply make mistakes. Fermat was quite likely wrong about his claim of having a proof; Euler made an important mistake in his proof of the case $n = 3$ (see also section 4.5) and also Wiles' initial proof needed several smaller and bigger repairs.

Chapter 3

Isabelle and other proof assistants

This chapter starts with an introduction to what a 'prover' can be used for, what provers are being developed and their differences. The second part contains a more detailed, but still introductory, description of Isabelle and a short explanation why Isabelle is chosen for this project.

3.1 Development of theorem provers

One of the first examples of automated theorem proving is the Automath project, led by De Bruijn (1980). It involved a computer system that could check the correctness of formal texts. After the project several new projects, like 'Mizar', were started. The Mizar system, still used today, has been used to proof-check a large part of (basic) set theory, algebra, topology etc. Later on, many new, quite different, proof systems were developed like HOL, Coq, Isabelle and NQTHM. Some proof systems were developed for just a single project, others were designed to deal with a great diversity of mathematical subjects. Important results are the following:

- The Four colour theorem, stating that any planar map can be coloured with at most 4 different colours, such that no two adjacent regions have the same colour. This was proven by Appel and Haken in 1976, using a lot of computer effort. However, the computer program was that large that some mathematicians doubted its correctness. In 2004 Georges Gonthier and Benjamin Werner gave a formal proof of it, consisting of 60,103 lines of Coq proof text, of which one third was automatically generated [32].

- The Prime number theorem, stating that the number of prime numbers smaller or equal to a given x is very similar to $x/\log x$: their ratio converges to 1. This was proven in 1896 by Hadamard and de la Vallée Poussin and in 2005 Jeremy Avigad gave a formal proof of it using Isabelle (29,753 lines) [32] [2].

- The 'Flyspeck' project [11] (not finished yet). This project is about the formalisation of a proof of Kepler's conjecture, asserting that the density of a packing of congruent spheres in three dimensions is never greater than $\pi/\sqrt{18}$. This conjecture, finally proven in 1998, is an important part of the 18th problem, from the famous 23 problems Hilbert announced at the beginning of the 20th century. The formalisation is a large project in which several research groups are involved. The HOL-light system is the most important prover in the project, although other proof assistants, like Coq and Isabelle are involved as well.

- Theorem provers also have an important industrial use. For example, the control system of a subway or an aeroplane should not cause accidents because of a human programming mistake. To gain higher reliability, such algorithms are checked mechanically. The same holds for microprocessors: the correctness of their implementation of - for example - the division of floating points is verified by automated theorem proving.

The QED manifesto and the 'perfect prover'

In 1994 a group of anonymous scientists try to attract the attention of their colleagues to the opportunities of the combination of formalised mathematics and computer science. In an article called 'QED manifesto' [6] they describe a future in which all mathematical proofs have been written in a formal language, mechanically verified by a computer system. They argue that in such a situation mathematics would be much more accessible, that mistakes would be ruled out, that such a project would have great possibilities in education and that this would be in favour of the increasingly mondial collaboration of mathematians.

At the moment of this research the QED project clearly has not been completed yet. Nevertheless, several different proving systems have been developed and many parts of mathematics have been formalised, independently on different systems. At this moment is does not seem likely that one of the current systems has the ability of becoming 'the' QED system (see also [33]). Some natural requirements of such a system would include (see also [12], [4]):

- The system should be sound: it should not allow invalid theorems to be proven.

- The system should have a small proof kernel: the checking-program, which is responsible for the correctness of the verified proofs, should itself be relatively easy to check (this is also called the 'De Bruijn'-criterion).

- The system should have good automation: it should, at least, be able to automatically prove steps in a proof that only involve purely logical or purely (basic) calculation operations. This leaves the more 'mathematical' steps in the proof to the user.

- The system should be rich enough to deal with abstract mathematics: for example it should be able to see the set of homomorphisms between two abelian groups as a group itself, or to prove things like 'every field has an algebraic closure'.

- The system should be highly user-friendly: it should have readable input files, a good user interface, clarifying error messages and tools for creating and browsing an efficient and clear library.

As emphasized in [33], all current provers still have important shortcomings according to this list. In general, systems with good automation do not have a rich enough language and vice versa. In none of the systems the automated calculational reasonings come close to the power of computer algebra systems like Mathematica (unfortunately, these are not sound at this moment). An integration of such powers would probably make the systems much more attractive for mathematicians.

Most interesting provers

One of the first choices in the research behind this thesis was the choice for an appropriate proof assistant. The most important criterions for such a decision involved:

- the accessibility of the proof assistant: the number of users is a first indication of it;

- its suitability for proofs in elementary number theory: this could, at least to some extend, be deduced from the amount of number theoretical proofs that already have been formalised in the prover;

- its user-friendliness: it should provide a pretty environment and generate readable output files.

Based on a comparison of 17 provers [32], the following provers were chosen for closer investigation: HOL, Coq, Mizar and Isabelle. All these provers have a large mathematical standard library, in particular containing the proofs of:

- the fundamental theorems of arithmetic and algebra;

- the gcd-algorithm;

- Fermat's little theorem and Euler's generalisation.

Besides that, these provers have a considerable number of users, have a good reliability and at least some form of automation. The following paragraphs are largely based on [32], including the example proof texts. For a better description of the underlying concepts see [3].

Coq

The Coq system [5] has been developed at INRIA (France) and has a lot of users in France as well as in Holland (Nijmegen). It is based on intuistionistic type theory. The most famous result in Coq, since its start in 1984, is the earlier mentioned proof of the Four colour theorem. Furthermore, the case $n = 4$ of FLT has already been formalised in Coq, which is a good indication for its suitability for our purpose. On the other hand, the Coq proof text is quite technical and therefore not well readable by unexperienced users. Example proof text:

```
generalize H0; rewrite H1; case p; auto; intros; discriminate;
```

Mizar

The Mizar project [17] started around 1973 and was mainly developed at Bialystok (Poland). At this moment the system has many users all over the world. The Mizar system is based on set theory. Compared with other proof assistants Mizar has the largest mathematical library. Some relevant results are the construction of Pythagorean triples and the theories of quotient rings and ideals and cyclic groups. Compared with the other three systems Mizar does not have much automation, which makes, in particular, the calculational reasonings quite laborious. On the other hand, the Mizar language is very good readable and is quite similar to informal proof texts. Example proof text:

```
then 2 divides m*m by NAT_1:def 3;
then 2 divides m by INT_2:44,NEWTON:98;
```

HOL

The HOL system [10] is, according to its name, based on higher order logic. It was developed around 1988 at Cambridge. A simplified version, called HOL-light, was made to prove many mathematical results, mainly from the 'top 100 list', see [34]. HOL-light is also involved in the earlier mentioned Flyspeck project. Relevant results in HOL-light concern the law of Quadratic Reciprocity and the construction of Pythagorean triples. The HOL proof texts are quite comparable with Coq. Example proof text:

```
MATCH_MP_TAC num_WF THEN REWRITE_TAC[RIGHT_IMP_FORALL_THM]
```

Isabelle

The Isabelle system [18] was developed around 1986 at Cambridge and München. The system is not based on a specific logic: it allows the user to choose between several logics, like HOL and ZFC. One of the largest projects was the proof of the Prime number theorem. Relevant results are a formalisation of the law of Quadratic Reciprocity and of quotient ideals. The Isabelle system has been attributed with an 'Isar' mode, which allows the user to write proofs in the more technical way (like Coq and HOL), as well as in a more mathematical way (like Mizar). Example proof text (Isar mode):

```
from eq have p dvd m^2 ..
with p_prime show p dvd m by (rule prime_dvd_power_two)
```

Conclusion

For our project the Isabelle prover seems most suitable. In constrast with HOL and Coq it has a very readable proof text, which is easier to learn and easier to read by others. A choice between Isabelle and Mizar is more complicated, see for example [31]. For our purpose, the large library of Mizar seems more attractive, but Isabelle has a much stronger automation. Besides that, the Isabelle environment is a little more advanced and the proof texts, in particular the LaTeX-output, are a little easier to read. Therefore Isabelle was chosen for this project.

3.2 The Isabelle system

This section contains a very short characterisation of the Isar language and gives a short impression what working with Isabelle is like.

The Isabelle/Isar language in one example

The next proof text is an example of a proof of $\gcd(a, b) = 1 \Rightarrow \gcd(a^n, b) = 1$ for natural numbers a, b, n.

```
lemma gcd(a,b)=1 ==> gcd(a^n,b)=1
  proof -
  assume ab: gcd(a,b)=1
  thus gcd(a^n,b)=1
  proof (induct n)
    case 0
    show gcd(a^0,b)=1 by auto
  next
    case (Suc n)
    hence gcd(a^n,b)=1 by simp
    with ab have gcd(a*a^n,b)=1 by (simp only: gcd-mult-cancel)
    thus gcd(a^Suc n,b)=1 by simp
  qed
qed
```

In the Isar style, theorems, lemmas, but also the basic steps in a proof can be proven in (roughly) three ways:

- Using the command `by`, followed by one or more rules or automation commands. In the example above most steps can be proven by just using `by simp`. The most 'mathematical' step is proven by the rule `gcd-mult-cancel`, which can be found in the GCD library, stating
  ```
  gcd(k,n) = 1 ==> gcd(k*m,n) = gcd(m,n)
  ```
  In our case the line starts with `with ab`: this means that the result labeled

`ab` (stating $\gcd(a, b) = 1$) and the above result ($\gcd(a^n, b) = 1$) should be used in the step. Note that the conclusion ($\gcd(a * a^n, b) = 1$) not follows from only applying `gcd-mult-cancel`, it also uses the transitivity of the equality. This rule is automatically invoked by using `simp only: gcd-..` instead of just using `rule gcd-...`
Finally, note that the statement $\gcd(a^0, b) = 1$ requires the `auto`-method: first it has to rewrite $a^0 = 1$ (note that 0^0 is defined as 1) and then it has to apply the rule $\gcd(1, m) = 1$, which can be found as `gcd-1` in the GCD library. However, the rule is declared to be a simplification rule and hence it is automatically used by `auto`.

- Using the command `proof`, optionally followed by some proof method like `rule ccontr` (proof by contradiction) or, like above, `induct n`. In the latter case, the user has to prove the statement for $n = 0$ and, by assuming the statement holds for some n, for `Suc n` ($= n + 1$). Any use of `proof` should be finished with a `show-` or a `thus`-statement, followed by the statement that had to be proven and the `qed`-keyword. Note that `thus` is just an abbreviation of `then show` – just like `hence` abbreviates `then have` –, where the command `then` expresses that the previous result should be used.

- Using a sequence of the commands `apply`, each followed by one or more rules, tactics, or automation commands, finished by the `done`-command. This is actually the traditional Isabelle style. In general it results in a much shorter proof script. For example, we could also have proven the above lemma by the commands
```
    apply(induct n)
    apply(auto simp add: gcd-mult-distrib)
  done
```
Note that this can even be abbreviated to the single line proof
```
  by (induct n, auto simp add: gcd-mult-distrib)
```
However, for more complicated proofs this latter abbreviation is not possible anymore. Moreover, the `apply`-style gives some benefits like a good interaction with the proof state. This will be pointed out below.

A good and short introduction to the Isar language can be found in [19]. The Isabelle webpage [18] contains some further documentation about the Isabelle language.

The Isabelle/ProofGeneral environment

The most convenient way to work with Isabelle is to use the Proof General system [21]. Proof General is a generic interface for proof assistants, based on the text editor Emacs. It provides an environment to process Isabelle or Isar proof texts line by line and displays the proof state at any requested moment. The proof state consists of the already proven results – which can be invoked – and the proof-goal at the specific place in the proof. For example, in the above proof the proof state reads, after the method `apply(induct n, auto)`, as follows (a little modified):

```
ALL n. [| gcd(a^n, b)=1; gcd(a,b)=1 |] ==> gcd(a*a^n,b)=1
```

which immediately suggests that a rule like `gcd-mult-distrib` should be used. In particular in the '`apply`'-style this proof state is essential, but it can also be clarifying in Isar-mode proofs.
Besides this, the Proof General environment provides tools for searching theorems, for displaying characters in a more mathematical mode and for displaying free and fixed variable, keywords, etc. in different colours.

The Isabelle system has several tools for generating readable output files. For example, it automatically generates LaTeX proof documents like the appendices of this thesis – including the theory dependencies tree – and a number of html-files which makes the theory files completely suitable for internet browsers. Both tools are also used for publishing the *Archive of Formal Proofs* [14]. This is an online journal, containing the most important results achieved in Isabelle. The entries in this archive are generated completely automatically and, of course, verified automatically.

3.3 Conclusion

To conclude, people that are working in the field of computer assisted theorem proving are aware of the high potential of such systems. Moreover, in the last years some serious projects have been undertaken and several aspects of the QED manifesto have been shown to be realisable. Nevertheless, all current proof assistants have their (essential) shortcomings and therefore it seems reasonable that 'the system' still has to be developed. However, for our purpose still several proof assistants are more or less suitable. We choose for Isabelle, because of the very readable input and output files, its powerful automation and pretty working environment.
For the proof of FLT3&4 the most basic mathematics is available in Isabelle. Some exceptions on this:

- calculating with equivalences up to units;
- the proof method 'infinite descent';
- results on ideals (in fact they are present, but they are developed in an old-fashioned way and hence not applicable);
- the definition of a (general) unique factorisation domain.

These results or definitions should therefore be avoided or should still be developed.

Chapter 4

Case study: Fermat's last theorem in Isabelle

The main goal of this chapter is to describe how the formalisation of mathematics could work in practice. What choices need to be made during the process? How getting familiar with a proof assistant like Isabelle?
In constrast with the previous sections, this chapter has a more report-like character: the 'core results' of the research can be found in the appendix.

We start with some remarks on the general approach of this case study. After that, the cases $n = 4$ and $n = 3$ are treated separately. In both cases we first discuss the several possibilities to prove the theorem, give an informal description of the (ultimate) proof and make some observations on their formalised version.

4.1 General approach

My general approach of this case study was as follows:

- Just try to get started with Isabelle: try to learn it by working with it and by studying example files.
- Immediately start with parts of the proof of the easy case $n = 4$. After that, probably with a lot of 'trial-and-error', I should be able to handle the more difficult case $n = 3$. Afterwards, it might make sense to improve the proof the case $n = 4$ with my new experiences.

It should be emphasized that there is never *one proof*: in particular in our case, there exist many quite different ways to prove the theorem. Such might result in a completely different way of reasoning. But there are also many possibilities on a more detailed level. For example, one can just argue by linear forward-reasoning, but in more complicated proofs it might make sense to work more top-down: splitting the problem into more sub-problems. Although such might be a matter of 'taste', the following objectives are quite natural:

- The proof should be as general as possible. Note that in a formal proof one can not prove something by just saying 'by analogy', even when just all 2's need to be replaced by 3's.
- The proof should be as readable as possible. The most straightforward way to achieve this is to stay as close to the original proof as possible.
- The proof should be as short as possible. Note that a very detailed proof can sometimes be (extremely) shortened by making some technical changes in the proof.

It goes without saying that this three objectives can have a negative influence on each other. In such a case one should choose the most 'elegant' way, which could be a matter of taste.
Note that this immediately implies that the formalisation of a proof is an interactive process. Of course one should start with some informal proof, but during the formalisation it might turn out that another approach would be far much better. Therefore, the informal proofs in the next sections are more like an end-product of a formalisation process, rather than an starting-point of it. However, in the sections 4.4 and 4.7 the most important issues in the formalisation processes will be discussed.

4.2 Proving the case $n = 4$

All proofs of FLT4 assume that x, y, z is a counterexample (a nontrivial solution of something like $x^4 + y^4 = z^4$) and construct a smaller counterexample, using solutions of Pythagorean triples.

Pythagorean triples

A triple of integers (a, b, c) is a primitive Pythagorean triple if it is a solution of $a^2 + b^2 = c^2$ and $\gcd(a, b) = 1$. At least Euclid already knew how to obtain (all) such solutions: they are all of the form

$$(k^2 - l^2)^2 + (2kl)^2 = (k^2 + l^2)^2.$$

There are several ways to prove this. The most straightforward method is, nowadays, to see $a^2+b^2 = (a+bi)(a-bi)$ as a factorisation of c^2 in the ring of Gaussian integers: $\mathbb{Z}[i]$. Using the fact that the two factors are coprime and the fact that $\mathbb{Z}[i]$ is a UFD, this means that both factors are squares, up to a possible unit. In particular $a + bi = u(k + li)^2$ with $u \in \{\pm 1, \pm i\}$, which immediately yields the solution.
The attractiveness of this method is, apart from its elegance, its analogy with the proofs of higher cases of FLT, for example in Kummer's proof, which works over the number ring $\mathbb{Z}[\zeta_n]$ (note that $\zeta_4 = e^{2\pi i/4} = i$). NB: such rings are in general not UFD's, but there is some kind of unique factorisation in ideals, which is being utilised.

However, I did not use this method, but I used a more elementary approach (see section 4.3). The main reason to choose for this 'elementary approach' was the fact that working over the ring $\mathbb{Z}$, about which many results have already been formalised in Isabelle, was for me, as an amateur in Isabelle, a more comfortable idea than starting with more abstract subjects like "every PID is a UFD", etcetera.

Infinite descent

So, having constructed a smaller counterexample, what is the contradiction? To quote Fermat: "if [there would have been a solution], then there would also be a smaller one with the same property, and so on, which is impossible"[1]. However, from this quote it is not yet clear how the 'contradiction' works. Fermat himself adds to the previous quote: "to explain why would make this discourse too long, as the whole mystery of this method lies there". In another correspondence he points out to be quite proud of this method of 'infinite descent', as he calls it himself, and in an example concerning prime numbers of the form $4n + 1$ he explains: "... and so on until you reach 5". Indeed, this gives a contradiction as there are no prime numbers of that form smaller than 5.

However, in order to formalise this proof method, there are still two, technically different, interpretations possible (NB: we work over the natural numbers):

1. "For all n with $\neg P(n)$ there exists a $m < n$ and $\neg P(m)$." Indeed, from this fact one can conclude $P(n)$ for all n, by just applying 'strong induction':

$$\Big[\forall n:\ [\forall m < n:\ P(m)] \Rightarrow P(n)\Big] \Rightarrow \Big[\forall n:\ P(n)\Big].$$

   However, in the case $n = 0$ our statement has a technical disadvantage: "assume $\neg P(0)$, show there exists a $m < 0$ etc.", which is of course not possible. Fortunately, this is logically the same as showing that $P(0)$ holds (because a contradiction 'implies anything').

2. Therefore another, logically equivalent but less counterintuitive, interpretation is to say: "$P(0)$ holds (or some other lower bound) and for all $n > 0$ with $\neg P(n)$ there exists a $m < n$ with $\neg P(m)$".

Although Fermat seems to intend the first one, in my formalisation I have chosen for the second, more intuitive, method.

Working over $\mathbb{N}$ or over $\mathbb{Z}$?

The last important - technical - decision I had to make before starting with the formalisation was the choice to work over the set of integers or over the set of natural numbers. In the daily life of a mathematician both approaches are usually mixed together, but in Isabelle they are quite different: theories about, for

[1] See [29, p. 76]. NB: In this fragment he writes about the fact that the area of a Pythagorean triangle can not be a square – which is almost the same as FLT4.

example, addition, multiplication, parity, gcd's, prime numbers and congruences in the case of natural numbers usually have a counterpart for the case of integers, developed in distinct 'theory' files. Unfortunately, in some aspects there are more formalised facts for integers and in some aspects more for natural numbers. Moreover, the approaches are sometimes quite different. For example, an *integer* x is defined to be odd (notation $x \in zOdd$) if there exists an integer k such that $x = 2k + 1$. On the other hand, a *natural number* n is defined to be odd (notation $odd\ n$) if $\neg(even\ n)$ holds.
Although the mappings between $\mathbb{N}$ and $\mathbb{Z}$ do not seem too complicated ($int(n)$ is the natural embedding in $\mathbb{Z}$ and $nat(x)$ is for $x \geq 0$ the natural embedding in $\mathbb{N}$ and $nat(x) = 0 \in \mathbb{N}$ for $x < 0$), it is quite important to decide whether to develop a theory in $\mathbb{N}$ and then 'lift' the results to $\mathbb{Z}$ or to start in $\mathbb{Z}$ immediately.

For the proof of the case $n = 4$ I have chosen for the first approach: starting in $\mathbb{N}$ and then translating the result to $\mathbb{Z}$, for the following reasons:

- The library seemed to contain in general more results on natural numbers.
- Concepts like 'induction' or facts like $a + b \geq a$ are easier in this case.
- Unique factorisation was developed in $\mathbb{N}$, not in $\mathbb{Z}$.
- The lemmas 4 and 5 (see next section) are a little easier in the case of $\mathbb{N}$.

4.3 Informal proof

Infinite descent

Theorem 1. *Let $P(\cdot)$ be a statement about natural numbers. Assume $P(0)$ and assume that for each $n > 0$ with $\neg P(n)$ there exists a $m < n$ such that $\neg P(m)$. Then $P(n)$ for all n.*

Proof. Apply 'strong' induction on n, i.e. assume $n > 0$ and assume $P(k)$ for all $k < n$; to show: $P(n)$. Therefore assume $\neg P(n)$, but then we can find a $m < n$ such that $\neg P(m)$. This contradicts our hypothesis. □

In most cases, like in the proof of FLT4, we work over the integers instead of the natural numbers. More generally, we work over a domain and have a mapping to the natural numbers that induces the 'descent'.

Corollary 2. *Let $P(\cdot)$ be a statement about elements of some set D and let $V : D \to \mathbb{N}$. Assume $V(x) = 0 \Rightarrow P(x)$ and assume that for each $x \in D$ with $V(x) > 0$ and $\neg P(x)$ there exists a $y \in D$ with $V(y) < V(x)$ and $\neg P(y)$.
Then $P(x)$ for all $x \in D$.*

Proof. Just apply 'infinite descent' on the statement

$$Q(n) := \Big[\forall x \in D :\ V(x) = n \Rightarrow P(x)\Big]. \qquad \square$$

The main theorem

Theorem 3. *Let $x^4 + y^4 = z^4$ for some integers x, y, z. Then $xyz = 0$.*

PROOF. Suppose we have a counterexample x, y, z. With some substitutions and dividing out the common divisor of x and y we can obtain integers a, b, c such that the statement $Q(a, b, c)$ holds, where

$$Q(a,b,c) := \Big[a^4 + b^4 = c^2, \quad abc \neq 0, \quad \gcd(a,b) = 1, \quad a \text{ odd}\Big].$$

Our goal is to obtain integers $\hat{a}, \hat{b}, \hat{c}$ with $Q(\hat{a}, \hat{b}, \hat{c})$, but with $\hat{c}^2 < c^2$. Applying 'infinite descent' (corollary 2) we derive a contradiction. To be precise, we take:

$$D := \mathbb{Z}, \quad P(c) := \Big[\forall a, b : \neg Q(a,b,c)\Big], \quad V(c) := c^2.$$

Because (a^2, b^2, c) is a (primitive) Pythagorean triple, we can use a result of Euclid (lemma 5) to obtain $u, v \in \mathbb{Z}$ with $\gcd(u, v) = 1$ and

$$a^2 = u^2 - v^2, \quad b^2 = 2uv, \quad |c| = u^2 + v^2.$$

This also implies $u, v \neq 0$ and $\gcd(a, v) = 1$. Now we can repeat Euclid's method for $a^2 + v^2 = u^2$. This gives $k, l \in \mathbb{Z}$ with $\gcd(k, l) = 1$ and

$$a = k^2 - l^2, \quad v = 2kl, \quad |u| = k^2 + l^2.$$

But using the fact that b must be even we have

$$(b/2)^2 = \frac{1}{2}|uv| = |k| \cdot |l| \cdot (k^2 + l^2),$$

so the latter must be a square too. Because of lemma 4 we know there exist $\hat{a}, \hat{b}, \hat{c} \in \mathbb{Z}$ such that

$$k = \pm\hat{a}^2, \quad l = \pm\hat{b}^2, \quad k^2 + l^2 = \pm\hat{c}^2,$$

where the last '$\pm$' must obviously be '$+$'. Hence we have

$$\hat{c}^2 = \hat{a}^4 + \hat{b}^4.$$

Furthermore, one easily verifies $\gcd(\hat{a}, \hat{b}) = 1$, $\hat{a}, \hat{b}, \hat{c} \neq 0$ and that either $\hat{a}$ or $\hat{b}$ must be odd. Finally we have

$$\hat{c}^2 = k^2 + l^2 = |u| \leq u^2 < u^2 + v^2 = |c| \leq c^2. \qquad \square$$

Relatively prime power divisors

Lemma 4. *Let K be a unique factorisation (semi)domain and $ab = c^n$ for some $n \in \mathbb{N}_{>1}$ and $a, b, c \in K$ with a and b relatively prime.*
Then there exist $\alpha \in K$ and $\varepsilon \in K^*$ such that $a = \varepsilon\alpha^n$.

PROOF. Let c have the prime factorisation $p_1 p_2 \cdots p_m$. We prove the lemma by induction on m.

- Assume $m = 0$: then c is a unit and hence a is a unit which clearly can be written (up to a unit) as n-th power.
- Assume $m \geq 0$ and assume the claim holds for c (i.e. for any coprime a, b we have $ab = c^n \Rightarrow \exists \alpha \in K,\ \varepsilon \in K^* : a = \varepsilon \alpha^n$): to prove the claim also holds for $\hat{c} = pc$, where p is a prime. Therefore assume we have $\hat{a}\hat{b} = \hat{c}^n = p^n c^n$ for coprime $\hat{a}$ and $\hat{b}$ and make the following case distinction:
  - Assume $p \mid \hat{a}$ and hence $p \nmid \hat{b}$, so $p^n \mid \hat{a}$ and hence let $\hat{a} = p^n a$. But now we can apply our hypothesis on $a\hat{b} = c^n$ (clearly a and $\hat{b}$ are coprime), to obtain α and ε such that $a = \varepsilon \alpha^n$. But then $\hat{a} = \varepsilon(p\alpha)^n$, which proves our claim.
  - Assume $p \nmid \hat{a}$ and hence $p^n \mid \hat{b}$, so let $\hat{b} = p^n b$. Therefore we can apply the hypothesis on $\hat{a}b = c^n$, which immediately proves the claim. □

The formal proof starts with the case $K = \mathbb{N}$ and lifts this result to $K = \mathbb{Z}$.

Euclid on pythagorean triples

Lemma 5. *Let $a^2 + b^2 = c^2$ with $a, b, c \in \mathbb{Z},\ \gcd(a, b) = 1$ and a odd.*
Then there exist $p, q \in \mathbb{Z}$ with $\gcd(p, q) = 1$ such that

$$a = p^2 - q^2, \quad b = 2pq, \quad |c| = p^2 + q^2.$$

PROOF. First assume $a, b, c \in \mathbb{N}$. Then we have

$$a^2 = (c - b)(c + b),$$

where both factors are odd, relatively prime and hence (using lemma 4 with $K = \mathbb{N}$) also squares. Let

$$(c - b) = r^2, \quad (c + b) = s^2.$$

Now, because r and s are odd we can set

$$p := \frac{s + r}{2}, \quad q := \frac{s - r}{2}$$

and it is straightforward to check that this is a solution. Finally, if a and/or b and/or c would have been negative then the argument can easily be modified, by interchanging (if necessary) p and q and/or swapping one of their signs. □

4.4 The formalised proof: conclusions

In this section I will discuss the results and the chosen approach. The results of the formalisation can be found in appendix A, chapter 1–3.

- Getting started with Isabelle is not too complicated, even for someone without any experience with proof assistants. Its library contains many good examples and there is a good documentation online. On the other hand, the library is not very well-organised and sometimes easy steps in a proof require a lot of work. For example, the proof of

$$[a \geq 1;\ b > 0] \Rightarrow ab \geq b$$

for integers a, b, requires something like:

```
assume a1: "a >= 1" and b0: "b > 0"
show "a*b >= b"
proof (cases)
  assume "a=1"
  thus ?thesis by auto
next
  assume "~ a=1"
  with a1 have "a > 1" by simp
  with b0 have "b*a > b*1" by (simp only: zmult_zless_mono2)
  thus ?thesis by (auto simp add: mult_ac)
qed
```

- There is an interesting difference between the formalised versions of the proofs of lemma 4 and 5, when compared with their 'informal' proofs in section 4.3. For the proof of lemma 4, using induction on the list of prime factors of c, I needed 1.5 pages of formalised proof script, plus approx. 1 page of helping lemmas and 0.5 pages of 'conversion to the integers'. For the proof of lemma 5 I needed not less than 3.5 pages plus approx. 1.5 pages of helping lemmas. Moreover, its conversion to the integers required another 1.5 pages. On the other hand, the *informal* proof of the latter looks much easier and shorter than the earlier. Calculating the ratios between the informal and the formal proof (without helping lemmas), we find a 'space factor' of 4 versus 13. For the proof of the main theorem this factor equals approx. 7. Note that this comparison is only 'locally' possible: in some textbooks a proof like

  PROOF (VERY INFORMAL). Just construct a smaller solution of $a^4 + b^4 = c^2$ by applying the standard formula for pythagorean triples twice. □

  might already be sufficient.

- However, the difference between the formalised proof of the lemmas 4 and 5 requires some explanation.

  – In Isabelle, calculating in $\mathbb{N}$ turns out to be, in general, far more complicated than calculating in $\mathbb{Z}$. Although one might expect that things like inequalities should be easier when working with positive numbers,

some important disadvantages occur when performing additions and substractions. For example, if $m < n$ for natural numbers m, n, then $m - n$ is just defined as 0. Therefore, facts like $(a-b)^2 = a^2 + b^2 - 2ab$ are far from trivial, even when $a \geq b$.

- Lifting a result from $\mathbb{N}$ to $\mathbb{Z}$, in particular when dealing with many variables, can be a lot of work.
- Inductive proofs are in general quite effective, in contrast with calculations.

- Therefore the main conclusion is that working over $\mathbb{N}$ should be minimized. Apart from the reasons above, there also turned out to be more results on $\mathbb{Z}$ present than expected (distributed over many different library files).

4.5 Proving the case $n = 3$

Kummer's proof

Like the previous case, there are many variations in the existing proofs of FLT3. The most natural proof would be the proof of Kummer, because it would be a first step to prove FLT for all regular primes. Traditionally the proof splits into two cases:

Case I: $3 \nmid xyz$. Unfortunately, Kummer's proof of this case does not work for $n = 3$. Instead, a result of Sophie Germain could be used, which works for all odd primes n with the property that $2n + 1$ is also prime. However, in our case – already being an exception of Kummer's proof – we could use a simple congruency argument: if $3 \nmid v$ then $v^3 \equiv \pm 1 \bmod 9$, and hence the equation $x^3 + y^3 \equiv z^3 \bmod 9$ does not have a solution in this case.

Case II: $3 \mid xyz$. Kummer's proof works over the ring $\mathbb{Z}[\zeta_n]$, but in the case $n = 3$ this is a UFD, so the proof can be simplified. A sketch of the proof looks like this:

Define $\pi := 1 - \zeta_3$ (with $\zeta_3 = e^{2\pi i/3}$) and notice that π^2 and 3 are equivalent up to a unit in the ring $\mathbb{Z}[\zeta_3]$. Now, case II of FLT3 is true if the following holds:

Theorem 6. *Let $\varepsilon \in \mathbb{Z}[\zeta_3]^*$. Then $x^3 + y^3 = \varepsilon z^3$ does not have a solution with $x, y, z \in \mathbb{Z}[\zeta_3]$ and $xyz \neq 0$ and $\pi \mid z$.*

Proof (sketch). Suppose there exists such a solution. Then there also exists a solution with coprime $x, y, z \in \mathbb{Z}[\zeta_3]$ and such that the multiplicity of π in z (notation: $\pi^n \,||\, z$) is minimal. Given such a solution we will show there exists one with a smaller n.

1. From $\varepsilon z^3 = (x + y)(x + \zeta_3 y)(x + \zeta_3^2 y)$ we have $\pi \mid (x + \zeta_3^j y)$ for some j.
2. Moreover, $(x + \zeta_3^j y) - (x + \zeta_3^{j+1} y) = \zeta_3^j \pi y$ implies $\pi \mid (x + \zeta_3^j y)$ for all j.

3. Therefore $\pi^2 \mid (x + \zeta_3^k y)$ for some $k \in \{0, 1, 2\}$.

   PROOF. Working modulo π^2 (which is the same as working modulo 3) we can write $x \equiv a_0 + a_1\pi$ and $y \equiv b_0 + b_1\pi$ for some $a_0, a_1, b_0, b_1 \in \mathbb{Z}$ (needs a small lemma). But then

   $$x + \zeta_3^j y \equiv (a_0 + b_0) + (a_1 + b_1 - b_0 j)\pi.$$

   Hence $a_0 + b_0 \equiv 0$ and using the fact that $b_0 \not\equiv 0$ (otherwise π would divide both y and x) we know $b_0^2 \equiv 1 \bmod 3$. So, for $j = -(a_1 + b_1)b_0 \in \mathbb{Z}$ we have $\pi^2 \mid (x + \zeta_3^j y)$. □

4. We may assume $k = 0$, because with the substitution $\hat{y} := \zeta_3^k y$ the above facts for y still hold for $\hat{y}$. Hence: $\pi^2 \mid x + y$.

5. With the same way of reasoning as in step 3 we conclude that π^2 can not divide $x + \zeta_3 y$ or $x + \zeta_3^2 y$.

6. Hence $n \geq 2$ and the multiplicity of π in $(x + y)$ equals $3n - 2$.

7. The factors $(x + \zeta_3^j y)/\pi$ are relatively prime.

   PROOF. Assume $g\pi$ divides both $(x + \zeta_3^i y)$ and $(x + \zeta_3^j y)$ for some prime $g \in \mathbb{Z}[\zeta_3]$ and $0 \leq i < j \leq 2$. Hence $g\pi$ divides both $x(\zeta_3^j - \zeta_3^i)$ and $y(\zeta_3^j - \zeta_3^i)$. But π is the only prime divisor of $(\zeta_3^j - \zeta_3^i)$ and with step 5 we conclude that g can not divide $(\zeta_3^j - \zeta_3^i)$. Hence g divides both x and y, which leads to a contradiction, because x and y are coprime. □

8. Moreover, by lemma 4 and the fact that $\mathbb{Z}[\zeta_3]$ is a UFD (still needs a proof), each factor must be a cube, up to a unit. Say: $(x + \zeta_3^j y)/\pi = \varepsilon_j \alpha_j^3$ for $\alpha_j \in \mathbb{Z}[\zeta_3]$ and units $\varepsilon_j \in \mathbb{Z}[\zeta_3]^*$ for $j = 0, 1, 2$.

9. In particular the α_j's are relatively prime and the multiplicity of π in α_0 is $n - 1 \geq 1$.

10. Combining the three equations $x + \zeta_3^j y = \varepsilon_j \alpha_j^3 \pi$ and eliminating x and y we get $E_0\alpha_0^3 = \alpha_1^3 + E_2\alpha_2^3$, where $E_0 = \varepsilon_0\varepsilon_1^{-1}(1 + \zeta_3^2)$ and $E_2 = \varepsilon_2\varepsilon_1^{-1}\zeta_3$ are both units.

11. Hence modulo 3 we have $E_2 \equiv (-\alpha_1\alpha_2^{-1})^3$ and hence $E_2 = u^3$ for some unit u (needs an elementary proof).

12. Using this we obtain $\alpha_1^3 + (u\alpha_2)^3 = E_0\alpha_0^3$ with $\pi^n \nmid \alpha_0$.

Conclusion: there exists a solution with a smaller $n \geq 1$ and hence, by the method of infinite descent we have a proof of our theorem. □

Towards a formalisation?

In order to formalise such a proof we need at least the following:

- Calculations in the number ring $\mathbb{Z}[\zeta_3]$ and properties of ζ_3.
- Calculating with congruences in that ring.
- The concept of a UFD, of a PID (principal ideal domain) and an ED (euclidean domain), and hence the general concept of unique factorisation, of a principal ideal and of a euclidean function.

Note that the general proof of Kummer also uses the concepts like class numbers, 'unique factorisation in ideals' and other operations on ideals, which can be avoided in our case. However, to formalise the above concepts in Isabelle would already require a lot of theory building: of course it would be possible to develop some *ad hoc* facts that would be enough for this case – but such would not be preferable. Moreover, to be able to do the required calculations, many (basic) facts about divisibilities would have to be lifted from $\mathbb{Z}$ and, for convenience, we would need something like 'equivalence up to units'.
At the moment of this research both the concepts of UFD and 'equivalence up to a unit factor' were still 'under construction' by the theorem proving group at München. In order to avoid duplicate work, I decided to investigate other possible approaches.

Euler's proof

Euler proved FLT3 in two different ways. Both proofs assume there exists a nontrivial solution of $x^3 + y^3 = z^3$ with x and y coprime and odd. The proofs continue by setting $p := (x + y)/2$ and $q := (x - y)/2$, which implies z^3 factors as $2p(p^2 + 3q^2)$. Moreover, assuming $3 \nmid z$, these factors are coprime and hence $p^2 + 3q^2$ must be a cube. The next step is to obtain a and b, using the following 'key lemma':

$$\left.\begin{array}{rcl} p^2 + 3q^2 & = & w^3 \\ \gcd(p, q) & = & 1 \end{array}\right\} \Rightarrow \exists\, a, b : \left\{\begin{array}{rcl} p & = & a^3 - 9ab^2, \\ q & = & 3a^2b - 3b^3. \end{array}\right. \tag{4.1}$$

Once this is established, it is quite easy to construct a new, smaller solution of the initial equation, which leads to the requested contradiction. The case $3 \mid z$ is handled in a similar way.
The difference between the two proofs is the way the key lemma is proven. In his first attempt (1760) Euler uses properties of the quadratic form $x^2 + 3y^2$. In fact he is able to prove that if $p^2 + 3q^2$ is a cube then there exist a and b such that

$$p^2 + 3q^2 = (a^3 - 9ab^2)^2 + 3(3ab^2 - 3b^3)^2.$$

However, this does not imply (yet) that 4.1 holds: the representation of an integer by such a quadratic form is in general not unique (e.g. $0^2 + 3 \cdot 2^2 = 3^2 + 3 \cdot 1^2$) and

in a later correspondence Euler admits this mistake.[2] Nevertheless, the problem can be repaired with a deeper study of this particular quadratic form (see [9, Ch. 2]).
Surprisingly, Euler's second proof (1770) contains a similar error. Or at least he does not give a full proof. This time the proof uses the ring $\mathbb{Z}[\sqrt{-3}]$, in which p^2+3q^2 factors as $(p+q\sqrt{-3})(p-q\sqrt{-3})$. Using the fact that these factors must be coprime, one might try to apply lemma 4 to conclude that both must be cubes, in particular there should exists a, b such that $p+q\sqrt{-3} = (a+b\sqrt{-3})^3$, which immediately implies 4.1. Unfortunately, the lemma can only be applied if $\mathbb{Z}[\sqrt{-3}]$ is a UFD, which is not the case (e.g. $2 \cdot 2 = (1+\sqrt{-3})(1-\sqrt{-3})$ although 2 and $1 \pm \sqrt{-3}$ are not equivalent up to a unit). However, the error can be fixed with techniques that Euler developed for another problem.

Towards a formalisation

In order to formalise FLT3 I have chosen for the first approach of Euler. The advantage is that it only uses facts about $\mathbb{Z}$, instead of a less trivial number ring with calculations concerning square roots and imaginary numbers. Both approaches are quite different from Kummer's proof (although the underlying ideas are in some sense very similar) and can not be generalised in an easy way. In fact both are quite *ad hoc*: the core is a deep study of either the form x^2+3y^2 or numbers of the particular form $x+y\sqrt{-3}$.
The proof in the next section is developed along the lines of the book of Edwards [9]. Nevertheless, in some crucial places there were opportunities to shorten the proof. On the other hand, in order to avoid calculations with $\sqrt{-3}$ a new approach had to be developed, which results in a (still) quite long proof.
The proofs of theorem 7 and the lemmas 8, 9 (first part), 11 and 16 - 21 are based on [9, Ch. 2]. Lemma 13 is taken from [24, p. 166].

4.6 Informal proof

The main theorem

Theorem 7. *Let $x^3+y^3=z^3$ for some integers x, y, z. Then $xyz=0$.*

Proof. 1. Assume $x^3+y^3=z^3$ and $xyz \neq 0$. After some renaming, taking negative values and dividing out the common factor we obtain nonzero integers x, y, z with $x^3+y^3=z^3$, $\gcd(x,y)=1$ and z even (and hence x, y odd). Our aim is finding α, β, γ having the same properties but with $|\gamma^3| < |z^3|$. With infinite descent we have then proven our claim.

[2] There is an interesting website about FLT, containing a detailed proof (of the case $n=3$): http://fermatslasttheorem.blogspot.com/2005/05/fermats-last-theorem-n-3-key-lemma.html.
The author also mentions Euler's mistake. Unfortunately, he *exactly* makes the same mistake in his own proof: "(7)... Which combined with step (1) gives us: $p^2+3q^2 = [a^3-9ab^2]^2+3(3a^2b-3b^3)^2$.
(8) Which means that we could define a, b such that: $p=a^3-9ab^2$ and $q=3a^2b-3b^3$..." (quoted at July 12th, 2007).

2. Since x and y are odd and coprime we obtain coprime integers p, q such that $x + y = 2p$ and $x - y = 2q$. We calculate

$$x^3 + y^3 = (x + y)(x^2 - xy + y^2) = 2p(p^2 + 3q^2) = z^3$$

and $\gcd(2p, p^2 + 3q^2)$ is equal to 1 or 3, which exactly corresponds to the cases $3 \nmid p$ and, respectively, $3 \mid p$. Our goal will be to show that such p, q don't exist in either case.

3. First assume $\gcd(2p, p^2 + 3q^2) = 1$. Then, by lemma 4, both $2p$ and $p^2 + 3q^2$ are cubes. Moreover $p^2 + 3q^2$ is odd and hence, using the key lemma (21) below, we know there exist integers a, b such that

$$p = a(a - 3b)(a + 3b), \quad q = 3b(a - b)(a + b).$$

It follows that $2p = (2a)(a-3b)(a+3b)$ must be a cube and since $\gcd(p, q) = 1$ these factors are relatively prime. Hence we obtain integers α, β, γ satisfying $2a = \gamma^3$, $a - 3b = \alpha^3$, $a + 3b = \beta^3$ and now it is easy to verify that (α, β, γ) satisfies the above requirements. (Note that $|\gamma^3| = |z^3|$ if and only if $x = y = 1$, but $z^3 = 2$ has no integer solutions.)

4. Now consider the case $\gcd(2p, p^2 + 3q^2) = 3$ and hence $p = 3r$ for some r. It follows that

$$x^3 + y^3 = (18r)(q^2 + 3r^2) = z^3,$$

where $\gcd(18r, q^2 + 3r^2) = 1$. Therefore $q^2 + 3r^2$ is a cube and as above we obtain coprime a, b such that

$$q = a(a - 3b)(a + 3b), \quad r = 3b(a - b)(a + b).$$

But also $18r = 3^3(2b)(a-b)(a+b)$ is a cube and hence all factors are cubes, say $2b = \gamma^3$, $a - b = (-\alpha)^3$, $a + b = \beta^3$ and hence (α, β, γ) gives a smaller solution. □

The binary quadratic form $x^2 + Ny^2$

Let N be an integer and define

$$S_N := \{a^2 + Nb^2 \mid a, b \in \mathbb{Z}\}.$$

To prove the key lemma, we will first prove some statements about this set S_N.

Multiplication and division

The following identities will be crucial:

$$(a^2 + Nb^2)(c^2 + Nd^2) = (ac \pm Nbd)^2 + N(ad \mp bc)^2. \tag{4.2}$$

Note that, if we take $\alpha = a + b\sqrt{-N}$ and $\beta = c + d\sqrt{-N}$, then the above equations are immediately inspired by

$$\alpha\bar{\alpha} \cdot \beta\bar{\beta} = \alpha\beta \cdot \overline{\alpha\beta} = \bar{\alpha}\beta \cdot \overline{\bar{\alpha}\beta}$$

in the ring $\mathbb{Z}[\sqrt{-N}]$. In fact, most of the next lemmas are just the counterpart of the (more obvious) facts in this ring.
Using these identities we have:

Lemma 8. *Let $A \in S_N$ and $B \in S_N$. Then $AB \in S_N$ and $A^n \in S_N$ for all $n \geq 1$.*

But we can also perform divisions:

Lemma 9. *Let $A \in S_N$ and let $P \in S_N$ be a prime divisor of A. Then $\frac{A}{P} \in S_N$. More explicitly: if, in addition, $A = a^2 + Nb^2$ and $P = p^2 + Nq^2$ then there exist integers u, v such that $A = (u^2 + Nv^2)P$ and $(a, b) = (pu \pm Nqv,\ pv \mp qu)$.*

Proof. We have

$$(bp + aq)(bp - aq) = b^2P - q^2A,$$

which is hence divisible by P. Therefore P divides $bp \pm aq$ for some choice of the sign. Now consider

$$AP = (ap \mp Nbq)^2 + N(bp \pm aq)^2$$

and hence P divides $ap \mp Nbq$. So, we have

$$\begin{aligned} vP &= bp \pm aq \\ uP &= ap \mp Nbq \end{aligned}$$

for some integers u, v. But that implies

$$\begin{aligned} Nbpq &= \pm(ap^2 - puP) &&= NqvP \mp Naq^2 \\ apq &= \pm(pvP - bp^2) &&= quP \pm Nbq^2 \end{aligned}$$

and hence

$$\left.\begin{aligned} a &= \tfrac{puP \pm NqvP}{p^2+Nq^2} &&= pu \pm Nqv \\ b &= \tfrac{pvP \mp uqP}{p^2+Nq^2} &&= pv \mp qu \\ A &= P(u^2 + Nv^2). \end{aligned}\right\}$$ □

Lemma 10. *Let P be an odd prime divisor of $A = a^2 + Nb^2$ and let $n \geq 1$, $P^n = p^2 + Nq^2$ and $\gcd(a, b) = \gcd(p, Nq) = 1$. Then $\frac{A}{P^n} \in S_N$.*
More explicitly: then there exist coprime integers u, v such that $A = (u^2+Nv^2)P^n$ and $(a, b) = (pu \pm Nqv,\ pv \mp qu)$.

Proof. Redo the proof of lemma 9 and use P^n instead of P. The only difficult step is

$$P^n \,|\, (bp + aq)(bp - aq) \Rightarrow P^n \,|\, (bp + aq) \ \vee\ P^n \,|\, (bp - aq),$$

which needs the fact that P can not divide both factors. This is exactly excluded by the additional (quite technical) conditions $\gcd(a, b) = \gcd(p, Nq) = 1$. □

Lemma 11. *Let $P \notin S_N$ be a prime and $PQ \in S_N$ for some $Q > 0$.*
Then there exists a prime divisor R of Q with $R \notin S_N$.

Proof. Suppose $R_1 \cdots R_n = Q$ for primes R_i which are all members of S_N. Then we can divide PQ consecutively by these prime factors and still stay in S_N by lemma 9. By induction (on n) we obtain a contradiction. □

Finally, we can strengthen lemma 8 in the case of the cube of a prime:

Lemma 12. *Let $N \geq 1$ and let $P = p^2 + Nq^2$ be an odd prime, $p \neq 0$.*
Then there exist a, b such that $P^3 = a^2 + Nb^2$ and $\gcd(a, Nb) = 1$.

Proof. Let $P = p^2 + Nq^2$. To get $P^3 = a^2 + Nb^2$ we take

$$a = p^3 - 3Npq^2, \quad b = 3p^2q - Nq^3.$$

Assume $\gcd(a, b) > 1$, then there exists an odd prime g dividing both a and b. Hence

$$g \mid p(p^2 - 3Nq^2) \ \wedge \ g \mid q(3p^2 - Nq^2).$$

Because g is prime we have four possibilities, and all lead to the conclusion $g \mid P$ and hence $g = P$. But that implies $P \mid p^2$ and hence $P \mid p$, which implies $p^2 \geq P^2 > P$, which is a contradiction. Similarly, if an odd prime g divides both a and N then it divides p^3 and hence also p and P: contradiction. □

Uniqueness

If $N > 1$ then the representation of a prime by the form $x^2 + Ny^2$ is unique (up to sign). Under some conditions, the same holds for the square and the cube of a prime.

Lemma 13. *Let prime $P = a^2 + Nb^2 = c^2 + Nd^2$ and $N > 1$.*
Then $(|a|, |b|) = (|c|, |d|)$.

Proof. We have

$$(ad + bc)(ad - bc) = a^2d^2 - b^2c^2 = (P - Nb^2)d^2 - b^2(P - Nd^2) = P(d^2 - b^2).$$

Now assume $d \neq \pm b$ (otherwise $|a| = |c|$ and we are done), and hence P divides either $ad + bc$ or $ad - bc$.
Now apply the two possibilities of formula (4.2) to obtain

$$P^2 = (ac + Nbd)^2 + N(ad - bc)^2 = (ac - Nbd)^2 + N(ad + bc)^2.$$

But either $ad + bc$ or $ad - bc$ is a nonzero multiple of P, so $P^2 \geq NP^2$, which is impossible. □

Lemma 14. *Let prime $P = p^2 + Nq^2$, $P^2 = r^2 + Ns^2$, $\gcd(r, s) = 1$ and $N > 1$.*
Then $(|r|, |s|) = (|p^2 - Nq^2|, |2pq|)$.

Proof. From lemma 9 we know there exist integers u, v such that

$$(r, s) = (pu \pm Nqv, pv \mp qu), \quad P = u^2 + Nv^2.$$

The latter implies, using lemma 13, that $(|u|, |v|) = (|p|, |q|)$. This gives four possibilities for (u, v). For (r, s) this yields, using $\gcd(r, s) = 1$, the given (four) possibilities. □

Lemma 15. *Let prime* $P = p^2 + Nq^2$, $P^3 = a^2 + Nb^2$, $\gcd(a, b) = 1$ *and* $N > 1$. *Then* $(|a|, |b|) = (|p^3 - 3Npq^2|, |3p^2q - Nq^3|)$.

PROOF. From lemma 9 we know there exist integers u, v (which must be coprime) such that

$$(a, b) = (pu \pm Nqv, pv \mp qu), \quad P^2 = u^2 + Nv^2.$$

The latter implies, using lemma 14, that $(|u|, |v|) = (|p^2 - Nq^2|, |2pq|)$. Combining this (four) possibilities with the (two) possibilities of (a, b) yields the given (four) possibilities. □

The case $N = 3$

Lemma 16. *Let* $A \in S_3$ *be even. Then* 4 *divides* A *and* $A/4 \in S_3$.

PROOF. If $A = a^2 + 3b^2$ then a and b must have the same parity. If a and b are even then we just divide them both by 2. So assume a and b are odd. Then $a \pm b$ is a multiple of 4 for an appropriate choice of the sign. But then $4A = (a \mp 3b)^2 + 3(a \pm b)^2$ is divisible by 4^2 and hence clearly $A/4 \in S_3$. □

Lemma 17. *Let* $P \notin S_3$ *be an* ***odd*** *prime and let* $PQ \in S_3$ *for some* $Q > 0$. *Then there exists an odd prime divisor* R *of* Q *with* $R \notin S_3$.

PROOF. If Q is odd then we are done by applying lemma 11. So assume Q is even and assume the statement holds for all smaller values of Q (for fixed P). So Q is a positive even integer. But now, according to lemma 16, PQ is divisible by 4 and $PQ/4 \in S_3$. So we can apply the statement on $Q/4$ to obtain an odd prime R dividing $Q/4$ with $R \notin S_3$. Clearly we can use this R. □

Lemma 18. *Let* P *be an odd prime divisor of* $A = a^2 + 3b^2$ *and* $\gcd(a, b) = 1$. *Then* $P \in S_3$.

PROOF. Suppose the statement is true for all smaller odd primes P (where A, a, b are not fixed). Now choose integers c and d with $c \equiv a \bmod P$ and $d \equiv b \bmod P$ such that $|c| < P/2$ and $|d| < P/2$. Hence $P \mid c^2 + 3d^2$ and by the choice of c, d we have $c^2 + 3d^2 < P^2$. If necessary we can divide c, d by $\gcd(c, d)$ to make them relatively prime and still have $P \mid c^2 + 3d^2$. Therefore let $Q := (c^2 + 3d^2)/P$ and hence $Q < P$. Moreover, by lemma 17, if $P \notin S_3$ then there exists an odd prime R dividing Q with $R \notin S_3$. On the other hand, $R < P$ and we can apply the statement on R (as divisor of Q) to conclude $R \in S_3$. This contradiction proves the lemma. □

Key lemma: cubes

Fix $N = 3$ and define

$$Q(a, b) := \left[\begin{array}{c} \text{there exist integers } p,\ q \text{ such that} \\ a = p^3 - 9pq^2, \quad b = 3p^2q - 3q^3 \end{array} \right].$$

Lemma 19. *Let* $\gcd(a, b) = 1$, P *an odd prime and* $a^2 + 3b^2 = P^3$. *Then* $Q(a, b)$.

PROOF. Because of lemma 18 we obtain p, q such that $P = p^2 + 3q^2$. Then, by lemma 15 we have

$$(|a|, |b|) = (|p^3 - 9pq^2|, |3p^2q - 3q^3|),$$

which proves $Q(a, b)$. □

Lemma 20. *Let $Q(a_1, b_1)$ and $Q(a_2, b_2)$. Then $Q(a_1a_2 \pm 3b_1b_2, a_1b_2 \mp a_2b_1)$.*

PROOF. Let

$$a_i = p_i^3 - 9p_iq_i^2, \quad b_i = 3p_i^2q_i - 3q_i^3 \text{ for } i = 1, 2.$$

Then just take

$$p_3 = p_1p_2 \pm 3q_1q_2, \quad q_3 = p_1q_2 \mp p_2q_1.$$

Finally, with some calculation we conclude

$$a_1a_2 \pm 3b_1b_2 = p_3^3 - 9p_3q_3^2, \quad a_1b_2 \mp a_2b_1 = 3p_3^2q_3 - 3q_3^3. \qquad \square$$

Lemma 21. *Let $\gcd(a, b) = 1$, w an odd integer and $a^2 + 3b^2 = w^3$. Then $Q(a, b)$.*

PROOF. We prove the lemma by induction on the number of prime factors of w. Let $C(n)$ be the statement

$$\left[\left.\begin{array}{ll} w = & \prod_{i=1}^{n} P_i, \quad \text{for odd primes } P_1, \ldots, P_n, \text{ and} \\ w^3 = & a^2 + 3b^2, \quad \text{for coprime numbers } a \text{ and } b \end{array} \right\} \Rightarrow Q(a, b) \right].$$

Our aim is to prove $C(n)$ for all $n \geq 0$.

- Assume $n = 0$: hence $w = 1$ and therefore $(a, b) = (\pm 1, 0)$ and hence $(p, q) = (a, 0)$ works.
- Assume $C(n)$ is true for some $n \geq 0$: to prove: $C(n + 1)$. Therefore assume we have

  $$w = \prod_{i=1}^{n+1} P_i = \hat{w}P_{n+1}, \quad w^3 = a^2 + 3b^2$$

  for some coprime integers a, b. To prove: $Q(a, b)$.
  We have $P_{n+1} | w$ and hence, by lemma 18, we obtain α, β such that

  $$P_{n+1} = \alpha^2 + 3\beta^2, \quad \alpha \neq 0$$

  (note that $\alpha = 0$ implies $3 \,|\, w$ and hence 3 divides both a and b). Therefore, using lemma 12 there exist p, q such that

  $$P_{n+1}^3 = p^2 + 3q^2, \quad \gcd(p, 3q) = 1.$$

  From lemma 10 we now obtain coprime $\hat{a}, \hat{b}$ such that

  $$\frac{w^3}{P_{n+1}^3} = \hat{w}^3 = \hat{a}^2 + 3\hat{b}^2, \quad (a, b) = (p\hat{a} \pm 3q\hat{b}, p\hat{b} \mp q\hat{a}).$$

  Now apply $C(n)$ on $\hat{w}$ to conclude $Q(\hat{a}, \hat{b})$. Furthermore apply lemma 19 on P_{n+1} to conclude $Q(p, q)$. Finally apply lemma 20 to conclude $Q(a, b)$. □

4.7 The formalised proof: conclusions

The formalised proof can be found in the chapters 4.1–4.5 and 5 of the appendix.

Space factor

The overall **space factor** is quite comparable with the previous case: without the comments the formal proof is about 10 times as long as the informal proof. However, between the several lemmas there is an interesting difference. The proof of the theorem itself (7) has a space factor of 15, which is quite large. The main reasons for this are the 'expensive' operations like choosing the parities right and performing calculations with cubes and squares. There is another significant difference between the lemmas 9 and 10. In their informal proofs the latter is only one line 'longer', whereas the formalised version is two pages longer. Apparently, working with gcd's, divisibilities and several case distinctions requires a lot of formal proof text. This also the reason why lemmas 12 and 18 have a space factor of 13 and 15. On the other hand, despite the fact that the informal proofs of lemma 14 and 15 are largely on a kind of meta-level (like 'combining these four possibilies with those two possibilies yields the given four possibilities'), their space factors are just approx. 5. To achieve this, some tricks had to be applied. For example, by making the units explicit (like $s = e \cdot 2pq,\ |e| = 1$ instead of $s = \pm 2pq$) the several cases could be simplified with some smart calculations.

The formalisation process

In general, during the formalisation my informal proof was adjusted frequently (the proof in the previous section is the result of a long process). There are several reasons for this phenomenon (which is also mentioned in [12]):

- In most cases there are many ways to formalise a specific part of a mathematical proof. Like mentioned above, such can have important consequences for the length of the formalisation. Making a good choice between the possible approaches requires a high level of experience with the proof assistant (which was increasing in my case).

- Such can also result in a quite different approach of the total proof (like working over $\mathbb{N}$ versus $\mathbb{Z}$). Sometimes a better inspection of the library files results in new 'available facts' and sometimes crucial results are not available or not applicable.

- The process of formalisation can bring errors or unexpected gaps in the proofs to light. Although it depends on how detailed one's formal proof is, in general there will arise small mistakes in the proof or some steps will be underestimated. A typical example is the omission of the fact that 2 is not the cube of an integer, in the proof of theorem 7, step 3. Otherwise the case $p = 1$ and $q = 0$ is a counterexample: it does not give a 'descent'. Other,

less innocent examples are the initially omitted conditions (concerning coprimalities) of lemma 10 in addition to lemma 9. During the formalisation process this perfectly led to the conditions that could fix the error.

4.8 Addendum: related existence proofs

After a succesful completion of the initial case study – exponents 3 and 4 of FLT – I decided to formalise some additional number theoretic results, namely:

1. all prime numbers $\equiv 1 \bmod 3$ can be written as $x^2 + 3y^2$;
2. all prime numbers $\equiv 1 \bmod 4$ can be written as $x^2 + y^2$;
3. all natural numbers can be written as $x^2 + y^2 + z^2 + t^2$.

The reasons to perform this additional formalisations are as follows:

- The first result is a good completion of the study of the quadratic form $x^2 + 3y^3$. Moreover, its proof is relatively easy when using lemma 18.
- The second result can be proven with exactly the same techniques and would therefore not involve too much work. Besides that, it can be found as number 20 in the 'top 100' list of mathematical theorems [34], which makes the formalisation more interesting.
- The last one, also known as Lagrange's four-square theorem, is a quite elegant result. It also plays an important role in the proof of Matiyasevich's theorem that there exists no algorithm that can verify whether a given (Diophantine) equation has a solution in integers or not, in a finite number of steps (also known as Hilbert's tenth problem) [16]. Like the proof of the solutions of sums of two squares, the proof of the four-square theorem also mainly involves techniques that already have been applied in earlier proofs. The problem can also be found as number 19 on Wiedijk's list [34].
- It could be interesting to investigate how many time someone, with some first experience, needs for the different stages in the formalisation of such proofs. Several authors also made observations of this type, which could be compared with the own observations.
- The proofs require a little new 'mathematics' I did not use before. For example, working with the Legendre symbol, (explicit) calculations with congruences, sets, sizes of sets and bijections between sets. This might help to get a better-founded opinion about Isabelle.

Numbers of the form $x^2 + 3y^2$ and $x^2 + y^2$

In order to prove which prime numbers can be written as $x^2 + 3y^2$ or $x^2 + y^2$ we can largely make use of our previous results. The only additional concept we need

is the Legendre symbol $(\frac{a}{p})$, which expresses whether an integer a is a quadratic residue modulo p. To be precise, if a is an integer and p is an odd prime then

$$\left(\frac{a}{p}\right) := \begin{cases} 0 & \text{if } p \mid a \\ 1 & \text{if } p \not\mid a \text{ and } \exists x: \ x^2 \equiv a \bmod p \\ -1 & \text{otherwise.} \end{cases}$$

Note that this definition immediately implies that if $a \equiv b \bmod p$ then $(\frac{a}{p}) = (\frac{b}{p})$. The Legendre symbol is already formalised in Isabelle, as well as the following facts that we need:

Euler's criterion: if p is an odd prime then

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p.$$

Quadratic reciprocity: if $p \neq q$ are odd primes then

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

The next fact is not formalised yet and is therefore part of our formalisation:

Lemma 22. *Let p be an odd prime. Then $(\frac{a}{p})(\frac{b}{p}) = (\frac{ab}{p})$.*

Proof. Using Euler's criterion we have

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \equiv a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv (ab)^{\frac{p-1}{2}} \equiv \left(\frac{ab}{p}\right) \bmod p.$$

Since $p > 2$ the equality follows immediately. □

Now we can prove our theorems (note that for a prime number $p \equiv 1 \bmod 3 \Leftrightarrow p \equiv 1 \bmod 6$):

Theorem 23. *Let $p \equiv 1 \bmod 6$ be a prime number. Then $\exists x, y \in \mathbb{Z}: \ p = x^2 + 3y^2$.*

Proof. Write $p = 6m + 1$. We have $(\frac{p}{3}) = (\frac{1}{3}) = +1$. Hence, using lemma 22, the law of quadratic reciprocity and Euler's criterion it follows that

$$\left(\frac{-3}{p}\right) = \left(\frac{-1}{p}\right)\left(\frac{3}{p}\right) = \left(\frac{-1}{p}\right)\left(\frac{3}{p}\right)\left(\frac{p}{3}\right) = (-1)^{\frac{p-1}{2}}(-1)^{\frac{3-1}{2}\frac{p-1}{2}} = (-1)^{6m} = +1.$$

By definition, we can now obtain an integer s such that $s^2 \equiv -3 \bmod p$ and hence $p \mid s^2 + 3 \cdot 1^2 \in S_3$. Using lemma 18 we immediately conclude $p \in S_3$. □

Theorem 24. *Let $p \equiv 1 \bmod 4$ be a prime number. Then $\exists x, y \in \mathbb{Z}: \ p = x^2 + y^2$.*

Proof. In this case $(\frac{-1}{p}) = (-1)^{\frac{p-1}{2}} = +1$ and hence there exists an integer s such that p divides $s^2 + 1 \in S_1$. From here the proof of lemma 18 can be repeated. Note that in the induction hypothesis the condition that the prime is odd can be omitted, because the conditions $|c| \leq p/2$ and $|d| \leq p/2$ are enough to guarentee $c^2 + d^2 < p^2$. Therefore we can easily apply lemma 11 to conclude $p \in S_1$. □

The above proofs are taken from [22] and [24]. The proof of theorem 25 is largely based on [29].

Lagrange's four-square theorem

Theorem 25. *Any natural number m can be written as the sum of four squares.*

PROOF. 1. We first prove that all odd prime numbers can be written as the sum of four squares. Using induction on the number of prime factors of m and the formula

$$(a_1^2 + a_2^2 + a_3^2 + a_4^2)(b_1^2 + b_2^2 + b_3^2 + b_4^2) = c_1^2 + c_2^2 + c_3^2 + c_4^2, \tag{4.3}$$

where

$$\begin{aligned} c_1 &= a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4, \quad c_2 = a_1b_2 - a_2b_1 - a_3b_4 + a_4b_3, \\ c_3 &= a_1b_3 + a_2b_4 - a_3b_1 - a_4b_2, \quad c_4 = a_1b_4 - a_2b_3 + a_3b_2 - a_4b_1, \end{aligned}$$

we have then proven the general case, because the cases $0, 1$ and 2 are trivial.

2. Let $p = 2n + 1$ be a prime. Then the set of residue classes

$$\{x^2 \bmod p \mid 0 \leq x \leq n\}$$

contains $n + 1$ distinct elements, because if $0 \leq x, y \leq n$ then

$$x^2 \equiv y^2 \bmod p \;\Rightarrow\; p \,|\, (x + y)(x - y) \;\Rightarrow\; x = y.$$

Obviously the same holds for the set

$$\{-1 - y^2 \bmod p \mid 0 \leq y \leq n\},$$

which implies they have at least one element in common. That implies there exist $x, y \in [0, n]$ such that $p \,|\, x^2 + y^2 + 1$.

3. Therefore there exists a $t \in [1, n]$ such that pt can be written as the sum of four squares, say $pt = a_1^2 + a_2^2 + a_3^2 + a_4^2$. Now we continue by arguing that if $t > 1$ then there exists an $s \in [1, t - 1]$ such that ps can also be written as the sum of four squares. Using infinite descent we will ultimately conclude that p itself can be written as the sum of four squares.

4. First assume t is even. Then the a_i's can be arranged such that both $a_1 + a_2$ and $a_3 + a_4$ are even. Hence

$$\left(\frac{a_1 + a_2}{2}\right)^2 + \left(\frac{a_1 - a_2}{2}\right)^2 + \left(\frac{a_3 + a_4}{2}\right)^2 + \left(\frac{a_3 - a_4}{2}\right)^2$$

represents $p \cdot \frac{t}{2}$ as sum of four squares, so we are done.

5. Now assume t is odd. Then we choose $b_i \equiv a_i \bmod t$ such that $|b_i| < t/2$ for $i = 1 \ldots 4$. Therefore, analogous to the proof of lemma 18, there exists an $s < t$ such that $st = b_1^2 + b_2^2 + b_3^2 + b_4^2$. Note that s can not be zero, because then all b_i's would be zero and hence t would divide p, which is impossible. Now we calculate

$$pt \cdot st = c_1^2 + c_2^2 + c_3^2 + c_4^2,$$

where the c_i's are obtained from formula (4.3). Moreover, using the fact that $t \mid a_i - b_i$ for all $i = 1 \ldots 4$, it can easily be shown that t divides all c_i's and therefore

$$ps = \left(\frac{c_1}{t}\right)^2 + \left(\frac{c_2}{t}\right)^2 + \left(\frac{c_3}{t}\right)^2 + \left(\frac{c_4}{t}\right)^2$$

as requested. This completes our proof. □

Formalisation

The formalisation of the above proofs can be found in appendix A, section 4.6 and in appendix B.

Time factor

For the first two proofs I needed roughly 11 hours in total:

- 1 hour for reading the proofs in the text books;
- 2 hours for studying the Isabelle libraries to find and understand the necessary results (in particular the results about quadratic reciprocity);
- 2 hours for making a detailed version of the proofs;
- 5 hours for writing a raw version of the formal proofs;
- 1 hour for improving the formal proofs and adding comments.

Note that these tasks were not performed one after another. As mentioned before, in general there is an important interaction between the design and the 'implementation' of the proof.
The above results can give a first impression on how many time one needs for writing a formal proof in comparison with writing an informal proof. However, we have already seen that there are many kinds of 'informal' proofs, so there is not just one answer to this question. Therefore I consider two cases:

- Writing an informal proof like the one in section 4.8. In that case I also would have to read the text book proofs first. After that I would need about 2 hours to write the proof in LaTeX, without any mistakes. In total this would take 3 hours.
- Writing course notes for first year students. In that case I would need about 1 hour for checking what the students are supposed to know already. Let's assume the students know exactly as much as was present in the Isabelle library. Then I still would have to explain the concepts a little (like the Legendre symbol) and, more important, I would have to divide the proofs into much more little steps. I even would have to give some examples here and there. All in all such would take about 4 pages of LaTeX-output (about a half of the formal proof text) and would take me, in total, about 8 hours, including textual and lay-out refinements.

To conclude, writing a formal proof takes more time than writing an informal proof. However, if such an informal proof would be on the level of first-year students, the differences are quite small. Informal proofs like the ones in this thesis can be written 3 to 4 times faster than the formal proof.

For the third proof I needed about 12 hours (including the study of several text book proofs, etc.). The major part of this consisted of finding a way to formally prove that the two mentioned (residue) sets could not be disjoint. A first attempt using results on residue sets gave some complications in the end. A closer inspection of the results on (ordinary) sets turned out that such a more elementary approach would be more straightforward.

Space factor

The average space factor in the above proofs is approximately 8–9. This is a little smaller than the earlier results. There are two natural explanations for this:

1. No new auxiliary lemmas had to be proven, in particular because of earlier work.
2. Due to better experience with the `simp`-methods the formal proof text is more efficient than in the proofs of FLT3&4.

Quantitative results

We could conclude that the overall **space factor** in our case study is about 8–10 and is decreasing a little during the project. This confirms De Bruijn's claim [4, Ch. 4] that the ratio stays approximately the same when one proceeds in formalising a mathematical theory. However, a ratio of 9 is quite much larger than the 4 that Barendregt and Wiedijk mention in [4]. Once again it should be emphasized that such comparisons are quite complicated and strongly depend on the degree of detailedness of the informal proof. However, if we would not have used the Isar-mode, and instead the less readable but much shorter `apply`-style, the ratio might have been close to 4.

Barendregt and Wiedijk also make an estimation of the average **time factor** in the formalisation of a mathematical theory. They estimate a ratio of about 10. In my research this time factor has surely decreased strongly. Without any experience in Isabelle, one line of Isar proof text could sometimes take more than 10 minutes. However, in the proofs of section 4.8 the time factor, when only comparing the *writing* of the formal and informal proof, was less than 5 – which is, apparently, quite efficient.

Chapter 5

Discussion

This final chapter contains a discussion of the main results of the case study. The text will also pay attention to possible futural research and the further development of proof assistants.

5.1 Remarks on Isabelle

At an early stage in this research, Isabelle was chosen as the proof assistant to work with. In this paragraph there will be made some comments on Isabelle, largely on the 'user level'.

The main (positive) conclusion is of course, considering the results of this research, that Isabelle is a 'suitable' prover for projects like this: to prove some elementary results in number theory by a user without any specific knowledge about the formalisation of mathematics or proof assistants. This might be related with the following:

- The Isar language (the 'mathematical' style) makes Isabelle very accessible: not only for the unexperienced reader, but also for someone getting started in formalising mathematics. Moreover, it provides many different ways to present a proof, which enables the user to stay very close to the 'informal' proof language.
- There is a sufficient amount of documentation online. Although it might be recommendable to develop some standard tutorial for Isar, the document [19] is a good start and, together with some online course information containing many example files, this appeared to be enough for this research.
- There were quite a lot of basic facts 'available' in the Isabelle library about integers and natural numbers: even vs. odd, powers, gcd's, congruencies, etc.

Nevertheless, there should be given some comments on Isabelle:

- The library is not always very well-organised. It would be recommendable to make some 'guide' to all results in Isabelle. Moreover, some results should

be re-organised, like the results on integers distributed over the theory files IntDiv, IntDef, IntArith, Int2, IntFact, IntPrimes, etc.

- Typically, when a user wants to apply a method (he has finally found) to prove another small step in the proof, the proof assistant gives an error. Such can have several reasons, like: one of the necessary hypotheses is missing or the types do not match (for example when trying to apply a lemma about integers on natural numbers). In particular for an unexperienced user it can be very difficult to find the reason of the error. If it would be possible to provide more information to the user why some proof method has failed, such would be highly recommendable.

- Errors can be even more complicated. Isabelle provides the user much freedom in choosing new variables. For example, the user can introduce new variables without any specification of their type or, even worse, with the same name as an already existing variable. This can be a terrible source of incomprehensible mistakes. To give a (quite pathological but real-life) example:

```
assume n_gr1:"n>1"
have "2*n > n"
proof (induct n)
  (..)
  from n_gr1 have "n>0" by (..)
```

  Note that whatever rule or method is used, Isabelle will not be able to prove $n > 0$ (and without explaining why), simply because the assumption `n_gr1` refers to some other n than the n that is automatically introduced within the scope of the proof.
  Although the user might benefit from the high flexibility, errors like these can be considered as a disadvantage.

- Regardingless of the performances of the other proof assistants, one might be little disappointed about the strength of the *arith*-method: a proof method that should be able to perform some basic calculational reasoning. For example, to prove $a \geq 1 \wedge b > 0 \Rightarrow ab \geq b$ requires a lot of Isar lines (see section 4.4) and, together with basic steps like the distributiveness of multiplication, one might expect such to be done automatically.

FLT4: Coq versus Isabelle

Since FLT4 has been formalised in Coq already [8], a comparison with our formalisation in Isabelle might make sense. The formalisation of Delahaye and Mayero is comparable in the following sense:

- The amount of formalised number theory in Coq before they started does not seem to be very different from our situation.

- Their approach also works over the integers and utilises the parametrisation of Pythagorean triples.

However, it differs in the following sense:

- Their proof of the results on Pythagorean triples is quite different. They consider the rational points on the unit square, which involves calculations with rational numbers and solving more difficult equations.
- In fact they prove a stronger result (*Diophantus' 20'th problem*). This involves a little more work, although it uses the same techniques.

This would suggest that their formalisation would involve a greater effort than ours – even when both would be performed with the same proof assistant.
The actual number of lines they needed was about 2000. Compared with our formalisation of FLT4, which required 1000 lines (which could have been much a few hundred less if we would not have started with $\mathbb{N}$), this is quite a lot of proof text. Moreover, in average their lines contain about 2 times as much commands as in our case.
They also bring up the amount of time they needed for the project, which was equivalent with about 2 months of development. In our case we also needed about 2 months for completing the proof of FLT4, but that was completely *from scratch*: without any experience with Isabelle or other proof assistants. Note that the results on sums of squares (section 4.8) also involved 1000 lines of Isar-code, which was finished within the equivalent of one week.
The above results give, at least, no evidence that Coq would have been more suitable for this project. In fact it seems likely that comparable number theoretic proofs require more lines and more time. However, it would be recommendable to do some more research before drawing this conclusion.

5.2 Higher cases of FLT

The natural generalisation of this research would be the formal proof of higher cases of FLT. As mentioned before, the proof of all cases where n is a prime ≥ 5 by Wiles would be a major project. The author of this thesis is not familiar with all details of that proof and therefore the reader should not expect conclusions about the amount of such a project or the usability of Isabelle for it. Nevertheless, some things could be said about other special cases of FLT:

- There exist quite elementary proofs for the cases 5 and 7, see for example [9] and [22]. Although it is not unreasonable that such proofs can also be transformed into proofs only dealing with integers, it would be highly recommendable to start with some general theory about extended number rings and calculations with complex numbers and units.
- A formalisation of Kummer's proof, concerning all cases where n is a regular prime, involves more mathematics. As already mentioned in section 4.5, this also requires operations on ideals and results on the class number of number

rings. Since 5 and 7 are both regular, it would be recommendable to just start with Kummer's proof, omitting the quite accidental proofs of those special cases.

In the author's opinion, there should be no invincible obstacles in formalising Kummer's proof in Isabelle. Nevertheless, it should be mentioned that once Wiles' proof has been formalised Kummer's proof becomes, strictly speaking, redundant, in contrast to the proofs of exponents 3 and 4.

5.3 Development of proof assistants

This section discusses more general aspects of the subject. Therefore, its content should not be considered as a scientifically justified result of the research, in particular because Isabelle was the only proof assistant involved. Nevertheless, the case study did reveal some aspects that could be put in a broader context.

Why is formalisation important?

First of all, it should be mentioned that the development of proof assistants can have great consequences for mathematics. Not only can mechanically verified proofs guarantee the correctness of a proof, once a significant part of (basic) mathematics is formalised it can also be of great use in the education of mathematics. Provided that formal proofs are attributed with good comments, it is quite easy to develop an online system which makes such proofs perfectly accessible, at any level of detail (see also [15, Ch. 8.2]). Another application is the use of proof assistants in the class room: once a student is familiar with some proof assistant, that proof assistant can function as a fully automated and immediate auditor of class room and home work exercises (involving finding a proof).

Besides this it goes without saying the automation of proof finding has great potential. Taking into consideration that computer algebra systems are able to solve difficult sets of equations, integration problems and to perform qualitative deductions (like checking (in-)equalities), it would be quite reasonable that proof assistants could also be equipped with such automated methods. To give an idea: more than 30% of the formal proof text of our experiment could have been omitted if Isabelle would have the same calculational strength as, for example, Mathematica. For an average mathematician such would surely make the system much more attractive.

Why is theorem proving so difficult?

We started this thesis with a comparison between proving a theorem and playing a chess game. Such a comparison makes sense, because in both situations the rules are clear and the number of possible 'moves' at a given stage is limited. However, the computer can beat the best chess player in the world, but there are still many proof-problems a first-year student can solve better than the best theorem provers at this moment. There might be three explanations for this:

1. The number of 'possible moves' at each moment is a lot bigger. For example, there could be lots of possible rewritings, substitutions, applications of possible helpful theorems or applications of proof methods like proofs by induction, contradiction or infinite descent.

2. The number of total 'necessary moves' can be much larger. A usual chess game takes less than 100 moves. On the other hand, there can be very easy stated problems with having a huge proof: Fermat's Last Theorem is probably the best example of it. But the same holds for easier proofs. For example, the proof of the parametrisation of the Pythagorean Triples could be invented by an average student mathematics. However, it involves hundreds of Isabelle proof tactics.

3. Many proofs a student is supposed to come up with involve just a limited number of standard 'tricks'. Most problems in exams are designed such that they are solvable when one is familiar with, say, the 25 most important frame works to tackle a problem. Such frame works are in general very top-down, which is (still) very difficult for theorem provers.

A future for theorem provers

Note that this last explanation is very analogous with one remark in the introduction: the chess computer 'Deep Blue' never would have beaten Kasparov if its programmers would not have attributed it with loads of standard opening diagram, tricks in end games, etcetera. For the field of automated theorem proving this would mean a large effort in pattern recognition and programming top-down proof methods, beside creating more power on bottom-up tactics.

The author is convinced that it should be possible to create a theorem proving system that could tackle most (proving) problems a student is supposed to solve. Because of the quite restricted number of frame works and proving tricks such would involve, this might be possible in a decade. However, it is quite unreasonable – unless artificial intelligence is able to beat the mathematical power of the human brain – that the computer would get better in theorem finding than professional mathematicians. To give an example: in some proofs like the proof of the Hartman-Grobman theorem the existence of some special smooth functions plays an important role, e.g. a $f \in C^\infty$ such that $f(x) = 0$ for all $|x| > 1$ and $f(0) = 1$. The easiest solution of this is the function $f(x) = e^{-1/(1-x^2)}$ for $|x| < 1$ and $f(x) = 0$ for $|x| \geq 1$. However, 'inventing' such a function is not just a matter of 'applying some tricks' and therefore it does not seem likely that computers will ever be able to independently come up with such a solution.

5.4 Concluding summary

The first goal of this research – to give a formal proof of the exponents 3 and 4 of Fermat's Last Theorem – has been achieved succesfully. This means that another

small step in the formalisation of the general FLT has been made. Besides that, the research has given an interesting first impression of the field of 'formalising mathematics': a project like this is 'doable' in a few months. To be precise, without any specific experience in this field we could at least give 'some' proof of FLT3&4, using the proof assistant Isabelle. During this project some observations were made:

- Formalising mathematics is an interactive process in which the design of the proof and its implementation strongly influence each other. In particular, formalising a proof frequently results in finding a 'better' proof. Depending on the user's own accuracy, a proof assistant might even help a mathematician in finding smaller or bigger mistakes in his proof.

- Writing a formal proof takes more time than writing a text book proof. In the last formalisations of this research this ratio was about 5. This is smaller than the 10 suggested in [4].

- The ratio between the lengths (the space factor) of the formal and informal version of the proofs in this thesis is about 10 (4300 lines of formal proof and 50 lines per page, compared with 8.5 pages of informal proof text). This is larger than the ratio of 4 that was estimated in [4]. There could be several reasons for this. At least Isar is a language which focuses more on readability than on efficiency. Besides that, our proofs contained many calculational deductions, which required quite a lot of Isar lines.

All in all, Isabelle was a suitable proof assistant for this project and the Isar-mode results in very readable proof texts (see the appendices).
We conclude with a few remarks on the development of proof assistants:

- Even with the current strengths, proof assistants can have a good application in undergraduate education. Such might, in the first case, help students with training their proving skills. Moreover, on a longer term, such might also be used to mechanically evaluate some exams, or parts of it.

- The least exciting part of our formalisations involved the calculational deductions, in particular the reasoning with inequalities. It would be recommendable to put effort in investigating which techniques of computer algebra systems (like Mathematica) could be incorporated in the simplification methods of Isabelle (and other provers).

- In general it should be pointed out that the ambitions of the QED manifesto require better world wide collaboration. At least for newcomers in this field it can be quite peculiar to be faced with lots of systems that have all formalised parts of mathematics in a different way. It would be recommendable to collaborate on designing 'the system', or maybe even earlier in designing a formal language which can be used in several current and futural proving environments.

Samenvatting

Inleiding

Een wiskundig bewijs is in principe uiteen te rafelen in stappen die slechts bestaan uit het toepassen van een logische afleidingsregel of een wiskundig axioma. Evenals in het schaakspel is op elk moment het aantal mogelijke 'zetten' relatief klein. Het succes van de computer in het schaakspel (bijna 10 jaar geleden versloeg 'Deep Blue' de regerend wereldkampioen Kasparov in 7 duels), doet vermoeden dat de computer ook in het bewijzen van wiskundige stellingen een significante rol zou kunnen spelen. Dit blijkt ook - ten dele - het geval te zijn.
Om de computer daadwerkelijk wiskunde te laten 'bedrijven', dient de wiskunde eerst in een taal uitgedrukt te worden die de computer kan interpreteren. Daarnaast moet overeenstemming bereikt worden over welke logische en wiskundige grondwaarheden de computer mag gebruiken om mee te redeneren. Dit is het *formaliseren* van de wiskunde. In de afgelopen decennia zijn diverse automatische stellingbewijzers ontwikkeld die hiermee kunnen werken. Toch zijn deze stellingbewijzers in het algemeen niet in staat om een gemiddeld bewijs dat in eerstejaarscolleges wiskunde voorkomt automatisch te bewijzen. Wél zijn er goede resultaten geboekt moet zogenaamde bewijsassistenten: programma's die met behulp van tactieken die door een gebruiker worden ingevoerd steeds kleine stukjes van een bewijs zelfstandig kunnen doen.

De laatste stelling van Fermat

Prof. dr. Jan Bergstra geeft in [20] een opsomming van de, volgens hem, 10 meest gezichtsbepalende problemen voor het informaticaonderzoek. De eerste daarvan luidt als volgt: *Formaliseer en controleer per computer een bewijs van het door Wiles aangetoonde vermoeden van Fermat.* Dit vermoeden staat ook wel bekend als Fermat's Laatste Stelling, waarin hij stelt dat de vergelijking $x^n + y^n = z^n$ voor het geval $n > 2$ geen oplossingen heeft in gehele getallen x, y, z ongelijk aan nul. Hoewel hij beweert een bewijs voor deze stelling te hebben is dit zeer onaannemelijk. In ieder geval bewees hij de stelling zelf voor het geval $n = 4$ en wist Euler enkele decennia later het geval $n = 3$ te bewijzen. Wiskundigen hebben vervolgens bijna 300 jaar tevergeefs gezocht naar een bewijs van het algemene geval. Pas in 1993 weet Andrew Wiles met een bewijs te komen, maar dit blijkt een belangrijk 'gat' te bevatten. Een jaar later weet hij dit op te lossen, waarmee de stelling uiteindelijk bewezen is. Wiles' bewijs bouwt overigens voort op vele moderne resultaten uit de wiskunde die zeker nog niet bij Fermat bekend waren.

Dit onderzoek

Met name vanwege dit laatste is het formaliseren van Wiles' bewijs een zeer omvangrijk project. Op dit moment is, afgezien van enkele specifieke projecten, alleen relatief elementaire wiskunde geformaliseerd. In dit onderzoek beperken we ons daarom tot de gevallen $n = 3$ en $n = 4$ van Fermat's laatste stelling. De redenen om deze gevallen te formaliseren zijn als volgt:

- Wiles' bewijs gaat alleen over de gevallen waar n een priemgetal groter of gelijk aan 5 is. Pas samen met het bewijs voor de gevallen $n = 3$ en $n = 4$ is het bewijs van Fermat's laatste stelling compleet.
- Het formaliseren van deze relatief eenvoudige gevallen kan meer inzicht geven in de bruikbaarheid van stellingbewijzers in het algemeen en Isabelle in het bijzonder om wiskundige bewijzen mee te formaliseren, met een focus op de getaltheorie.

Er is gekozen voor de bewijsassistent 'Isabelle' omdat deze een relatief krachtig mechanisme heeft om kleine stappen in een bewijs zelfstandig te bewijzen, maar vooral vanwege de zeer leesbare in- en uitvoer van wiskundige bewijsteksten.

Resultaten

De resultaten van dit onderzoek zijn positief. Allereerst bleek Isabelle een zeer toegankelijk bewijsprogramma, waarin al snel de eerste kleine stappen van het bewijs van het geval $n = 4$ bewezen konden worden. Niettemin bleek het formaliseren een tijdrovend proces te zijn waarbij relatief eenvoudige stappen soms nog veel werk kostten. Om die reden is er ook voor gekozen om het bewijs van het geval $n = 3$ op een zo elementair mogelijke manier te formaliseren. Na een succesvolle afronding hiervan zijn nog een aantal gerelateerde getaltheoretische bewijzen geformaliseerd, namelijk die van onder andere de volgende stellingen:

- elk priemgetal dat van de vorm $4m + 1$ is, kan geschreven worden als de som van twee kwadraten;
- elk natuurlijk getal kan geschreven worden als de som van vier kwadraten.

Samen met de resultaten over Pythagoreïsche drietallen, die nodig waren voor het geval $n = 4$ van Fermat's laatste stelling, zijn hiermee drie resultaten uit de 'top 100 van wiskundige stellingen' geformaliseerd, zie [34]. Het geheel heeft bovendien geleid tot twee publicaties in de *Archive of Formal Proofs* [14], een online tijdschrift waarin de belangrijke resultaten met Isabelle gepubliceerd worden.
De formele, automatisch geverifieerde bewijzen zijn te vinden in de appendix van deze scriptie. In totaal beslaan de formele bewijsteksten zo'n 4300 regels over 85 pagina's. Dit is ruim 10 keer zoveel als dezelfde bewijzen zoals ze 'informeel' zijn opgeschreven in de voorgaande hoofdstukken. Voor de formele bewijzen van de stellingen over de sommen van kwadraten was ongeveer 4 á 5 keer zoveel tijd nodig als voor het uitwerken van hun informelere versie in deze scriptie. Vergeleken met resultaten in andere stellingbewijzers betekent dit dat bewijzen in Isabelle/Isar relatief lang zijn, maar dat het produceren ervan relatief snel kan.

Bibliography

[1] W.S. Anglin, *Mathematics: a concise history and philosophy.* New York (etc.): Springer-Verlag, 1994.

[2] Jeremy Avigad, Kevin Donnelly, David Gray and Paul Raff, 'A formally verified proof of the prime number theorem', *ACM Transactions on Computational Logic*, vol. V, no. N, April 2006, p. 1-22.
http://www.acm.org/pubs/tocl/accepted/283avigad.pdf

[3] Henk Barendregt and Herman Geuvers, *Proof-Assistants Using Dependent Type Systems.* In: Alan Robinson and Andrei Voronkov (ed.) *Handbook of Automated Reasoning.* Elsevier Science Publishers B.V., 2001.

[4] Henk Barendregt and Freek Wiedijk, 'The Challenge of Computer Mathematics', *Transactions A of the Royal Society* 363 no. 1835, p. 2351-2375, 2005.
http://www.cs.ru.nl/∼freek/notes/RSpaper.pdf

[5] Yves Bertot and Pierre Castéran, *Interactive theorem proving and program development, Coq'Art: the calculus of inductive constructions.* Berlin (etc.): Springer-Verlag, 2004. The Coq website:
http://coq.inria.fr/

[6] R. Boyer et al, *The QED Manifesto.* In: A. Bundy, editor, *Automated Deduction - CADE 12*, volume 814 of *LNAI*, p. 238-251. Springer-Verlag, 1994.
ftp://ftp.mcs.anl.gov/pub/qed/

[7] Chin-Liang Chan and Richard Char-Tung Lee, *Symbolic Logic and Mechanical Theorem Proving.* London: Academic Press, 1973.

[8] David Delahaye and Micaela Mayero, *Diophantus' 20th Problem and Fermat's Last Theorem for* $n = 4$ (2005).
http://hal.archives-ouvertes.fr/hal-00009425/en/

[9] Harold M. Edwards, *Fermat's Last Theorem. A Genetic Introduction to Algebraic Number Theory.* New York (etc.): Springer-Verlag, 1977.

[10] M.J.C. Gordon and T.F. Melham, *Introduction to HOL. A theorem proving environment for higher order logic.* Cambridge (etc.): Cambridge University Press, 1993. The HOL-light website:
http://www.cl.cam.ac.uk/users/jrh/hol-light/

[11] Tom Hales, *The flyspeck project fact sheet.*
http://www.math.pitt.edu/∼thales/flyspeck/

[12] John Harrison, 'Formalized Mathematics', *Mathesis Universalis* 1(2), Spring 1996.
http://www.calculemus.org/MathUniversalis/

[13] Wim H. Hesselink, *Computer verification of Wiles' proof of Fermat's Last Theorem.*
http://www.cs.rug.nl/∼wim/fermat/wilesEnglish.html

[14] Gerwin Klein, Tobias Nipkow and Larry Paulson (ed.), *Archive of Formal Proofs.* http://afp.sf.net/

[15] Iris Loeb, *Natural deduction, sharing by presentation* (PhD. thesis). Radboud Universiteit Nijmegen, 2007.

[16] Yuri Matiyasevich, *Hilbert's Tenth Problem.* Cambridge, London: MIT Press, 1993.

[17] The Mizar website: http://www.mizar.org/

[18] Tobias Nipkow, Larry Paulson and Markus Wenzel, *Isabelle/HOL: A proof assistant for higher-order logic.* Lecture notes in Computer Science, vol. 2283, Berlin: Springer-Verlag, 2002. The Isabelle website: http://isabelle.in.tum.de/

[19] Tobias Nipkow, *Structured proofs in Isar/HOL.* http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf

[20] *Nationale Onderzoeksagenda Informatie en Communicatietechnologie (NOAG-ict) 2005–2010*, Albani drukkers, Den Haag, 2005. http://ict.stw.nl/noagict/noagict.pdf

[21] The Proof General website: http://proofgeneral.inf.ed.ac.uk/

[22] Paulo Ribenboim, *Fermat's Last theorem for amateurs.* New York (etc.): Springer-Verlag, 1999.

[23] Eleanor Robson, Neither Sherlock Holmes nor Babylon: A Reassessment of Plimpton 322. *Historia Mathematica*, 28, p. 167-206, 2001.

[24] Daniel Shanks, *Solved and Unsolved Problems in Number Theory.* New York: Chelsea, 1962.

[25] D.J. Struik, *A concise history of mathematics.* New York: Dover, 1948.

[26] G. Thibaut, *The Pandit.* Old Series 9 (1874), 10 (1875) and New Series 1 (1876-77).

[27] B.L. van der Waerden, *Geometry and Algebra in Ancient Civilizations.* Berlin: Springer-Verlag, 1983.

[28] ——, *Ontwakende wetenschap. Egyptische, Babylonische en Griekse wiskunde.* Groningen: Noordhof, 1950. Translated as: *Science awakening*, 1954.

[29] André Weil, *Number theory: an approach through history; From Hammurapi to Legendre.* Boston (etc.): Birkhäuser, 1983.

[30] Markus M. Wenzel, *Isabelle/Isar - a versatile environment for human-readable formal proof documents* (PhD. thesis). Technische Universität München, 2001. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html

[31] Markus M. Wenzel and Freek Wiedijk, 'A comparison of the mathematical proof languages Mizar and Isar', *Journal of Automated Reasoning* 29, p. 389-411, 2002. http://www4.in.tum.de/∼wenzelm/papers/romantic.pdf

[32] Freek Wiedijk, editor, *The Seventeen Provers of the World.* Foreword by Dana S. Scott, Springer LNAI 3600, 2006. http://www.cs.ru.nl/∼freek/comparison/comparison.pdf and .../diffs.pdf

[33] ——, *The QED Manifesto revisited.* http://www.cs.ru.nl/∼freek/notes/qed2.pdf

[34] ——, *Formalizing 100 theorems.* http://www.cs.ru.nl/∼freek/100/

[35] Andrew J. Wiles, 'Modular elliptic curves and Fermat's Last Theorem', *Annals of mathematics* 141, p. 443-551, 1995.

Acknowledgements

Finishing this thesis I would like to thank my supervisors Jaap Top and Wim Hesselink for giving me the opportunity to do this research. It was a perfect match with my interest in number theory, computer science and the foundations of mathematics. Besides that I appreciated the high degree of freedom they gave me for choosing a way to deal with this 'pilot study'.
I also want to thank Freek Wiedijk for helping me with my first steps in this field and Clemens Ballarin for his e-mailsupport during a large part of the research. Moreover, I want to thank Tobias Nipkow, Larry Paulson and Jeremy Avigad for their enthusiastic reception of my results. It gave much satisfaction to finish my student time in this way. I am grateful to the above people and all others that contributed to this.

Appendix

The enclosed documents contain the text of the formal proofs of FLT3&4 and of some results on sums of squares. The documents will be published as:

A. Roelof Oosterhuis, *Exponents 3 and 4 of Fermat's last theorem and the construction of Pythagorean triples.* In: G. Klein, T. Nipkow, and L. Paulson (ed), *The Archive of Formal Proofs*, 2007.

B. ——, *Sums of two and four squares.* Idem.

NB: at this moment the files are only available in the development version of the AFP, which can be found at http://afp.sourceforge.net/devel.shtml.

Exponents 3 and 4 of Fermat's Last Theorem and the parametrisation of Pythagorean Triples

Roelof Oosterhuis
University of Groningen

August 24, 2007

Abstract

This document gives a formal proof, verified by the proof assistant 'Isabelle'[1], of the cases $n = 3$ and $n = 4$ (and all their multiples) of Fermat's Last Theorem: if $n > 2$ then for all integers x, y, z:

$$x^n + y^n = z^n \Longrightarrow xyz = 0.$$

Both proofs only use facts about the integers and are developed along the lines of the standard proofs, like in section 1 and 2 of Harold M. Edwards, *Fermat's Last Theorem. A Genetic Introduction to Algebraic Number Theory*, New York (etc.): Springer Verlag, 1977.
First, the framework of 'infinite descent' is being formalised and in both proofs there is a central role for the lemma

$$\gcd(a, b) = 1 \ \wedge \ ab = c^n \Longrightarrow \exists\, k : |a| = k^n.$$

Furthermore, the proof of the case $n = 4$ uses a parametrisation of the Pythagorean triples. The proof of the case $n = 3$ contains a study of the quadratic form $x^2 + 3y^2$. This study is completed with a result on which prime numbers can be written as $x^2 + 3y^2$.
We remind the reader that the case $n = 4$ of FLT, in contrast to the case $n = 3$, has already been formalised (in the proof assistant 'Coq')[2]. Moreover it should be mentioned that the parametrisation of the Pythagorean Triples can be found as number 23 on the list of 'top 100 mathematical theorems'.[3]
This research is part of a M.Sc. thesis under supervision of Jaap Top and Wim H. Hesselink (RU Groningen). The author wants to thank Clemens Ballarin (TU München) and Freek Wiedijk (RU Nijmegen) for their support.
More information: see http://www.roelofoosterhuis.nl/MScthesis.pdf

[1] See http://isabelle.in.tum.de/
[2] See http://hal.archives-ouvertes.fr/hal-00009425/en/
[3] See http://www.cs.ru.nl/~freek/100/

Contents

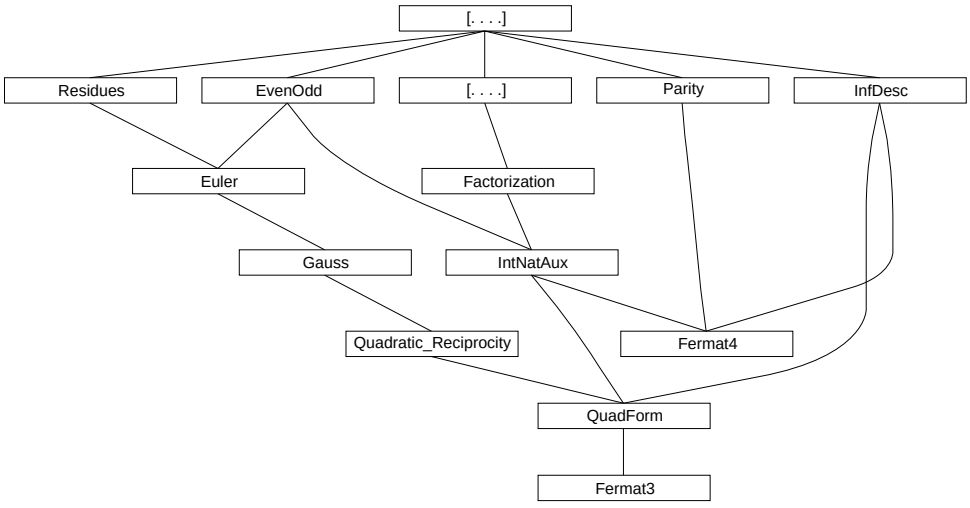

Figure 1: The depence on existing files in the Isabelle library.

1 The proof method ‘infinite descent’

theory *InfDesc*
imports *Main*
begin

The method of infinite descent, frequently used in number theory. Based on *less-induct*. $P(n)$ is true for all $n \in \mathbb{N}$ if

- case “0”: given $n = 0$ prove $P(n)$,
- case “smaller”: given $n > 0$ and $\neg P(n)$ prove there exists a smaller integer m such that $\neg P(m)$.

lemma *nat-infinite-descent*:
$[\![P\ 0; !!n.\ n>0 \Longrightarrow \neg\ P\ n \Longrightarrow (\exists\, m::nat.\ m < n \wedge \neg P\ m)\ ]\!] \Longrightarrow P\ n$
by (*induct n rule*: *less-induct*, *case-tac n>0*, *auto*)

lemmas *infinite-descent*
= *nat-infinite-descent* [*rule-format*, *case-names 0 smaller*]

Infinite descent using a mapping to $\mathbb{N}$: $P(x)$ is true for all $x \in D$ if there exists a $V : D \to \mathbb{N}$ and

- case “0”: given $V(x) = 0$ prove $P(x)$,
- case “smaller”: given $V(x) > 0$ and $\neg P(x)$ prove there exists a $y \in D$ such that $V(y) < V(x)$ and $\neg P(y)$.

NB: the proof also shows how to use the previous lemma.

corollary *nat-val-infinite-descent*:
fixes V:: $'a \Rightarrow nat$
assumes *ass0*: $!!x.\ V\ x = 0 \Longrightarrow P\ x$
and *assn*: $!!x.\ V\ x > 0 \Longrightarrow \neg P\ x \Longrightarrow (\exists\, y.\ V\ y < V\ x \wedge \neg P\ y)$
shows $P\ x$
proof −
obtain n **where** $n = V\ x$ **by** *auto*
moreover have $!!x.\ V\ x = (n::nat) \Longrightarrow P\ x$
proof (*induct n rule*: *infinite-descent*)
case *0* — i.e. $V(x) = 0$
with *ass0* **show** $P\ x$ **by** *auto*
next — now $n > 0$ and $P(x)$ does not hold for some x with $V(x) = n$
case (*smaller n*)
then obtain x **where** *vxn*: $V\ x = n$ **and** $V\ x > 0 \wedge \neg\ P\ x$ **by** *auto*
with *assn* **obtain** y **where** $V\ y < V\ x \wedge \neg\ P\ y$ **by** *auto*
with *vxn* **obtain** m **where** $m = V\ y \wedge m<n \wedge \neg\ P\ y$ **by** *auto*
thus *?case* **by** *auto*
qed
ultimately show $P\ x$ **by** *auto*
qed

lemmas *val-infinite-descent*
= *nat-val-infinite-descent* [*rule-format*, *case-names 0 smaller*]

end

2 Powers, prime numbers and divisibility

theory *IntNatAux*
imports
~~/*src/HOL/NumberTheory/Factorization*
~~/*src/HOL/NumberTheory/EvenOdd*
begin

Contains lemmas about divisibility and coprimality of powers, as well as some results about parities and small powers. Most lemmas are developed for the integers as well as for the natural numbers.

2.1 Auxiliary results

lemma *make-relprime*:
$(a \neq 0 \vee b \neq 0) \Longrightarrow \exists\ c\ d.\ a = gcd(a,b)*c \wedge b = gcd(a,b)*d \wedge gcd(c,d) = 1$
proof −
assume *ab0*: $a\neq 0 \vee b\neq 0$
let *?g* = $gcd(a,b)$
have *?g dvd a* ∧ *?g dvd b* **by** *auto*
then obtain *c d* **where** *abcd*: $a = ?g*c \wedge b = ?g*d$ **by** (*auto simp add*: *dvd-def*)
moreover have $gcd(c,d)=1$
proof −
from *abcd* **have** $?g*gcd(c,d)=?g$ **by** (*auto simp add*: *gcd-mult-distrib2*)
moreover with *ab0* **have** $?g \neq 0$ **by** (*simp add*: *gcd-zero*)
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *factor-exists-general*: $(a::nat) \neq 0 \Longrightarrow (\exists\ ps.\ primel\ ps \wedge prod\ ps = a)$
proof −
assume *a0*: $a\neq 0$
show *?thesis*
proof (*case-tac a=1*)
assume *a=1* **hence** *primel* [] ∧ *prod* [] = *a* **by** (*auto simp add*: *primel-def*)
thus *?thesis* **by** *auto*
next
assume $a\neq 1$ **with** *a0* **have** $a>Suc\ 0$ **by** *auto*
thus *?thesis* **by** (*rule factor-exists*)
qed
qed

lemma *make-zrelprime*: $(a \neq 0 \vee b \neq 0)$
$\Longrightarrow \exists\ c\ d.\ a = zgcd(a,b)*c \wedge b = zgcd(a,b)*d \wedge zgcd(c,d)=1$
proof −

```
  assume ab0: a ≠ 0 ∨ b ≠ 0
  let ?g = zgcd(a,b)
  have ?g dvd a ∧ ?g dvd b by auto
  then obtain c d where abcd: a = ?g*c ∧ b = ?g*d by (auto simp add: dvd-def)
  moreover have zgcd(c,d) = 1
  proof −
    from abcd have ?g * zgcd(c,d) = ?g
      by (auto simp add: zgcd-zmult-distrib2 zgcd-geq-zero)
    moreover with ab0 have ?g ≠ 0 by (auto simp add: zgcd-def gcd-zero)
    ultimately show ?thesis by simp
  qed
  ultimately show ?thesis by auto
qed

lemma int-nat-abs-eq-abs: int(nat|x::int|) = |x|
  by simp

lemma prime-impl-zprime-int: prime (a::nat) ⟹ zprime (int a)
proof −
  assume pra: prime a
  show zprime (int a)
  proof −
    from pra have agr1: 1 < int a by (unfold prime-def, auto)
    moreover have !!m. m ≥ 0 ∧ m dvd int a ∧ m ≠ int a ⟹ m=1
    proof −
      { fix m assume m: m ≥ 0 ∧ m dvd int a ∧ m ≠ int a
        then obtain k::int where k: int a = m*k by (auto simp add: dvd-def)
        from m have int (nat m) = m by auto
        with k have int a = (int (nat m)) * k by simp
        hence nat (int a) = nat ( (int (nat m)) * k ) by simp
        hence a = nat ( (int (nat m)) * k ) by (simp only: nat-int)
        also have ... = (nat m) * (nat k) by (simp add: nat-int nat-mult-distrib)
        finally have nat m dvd a by auto
        with pra have nat m = a ∨ nat m = 1 by (auto simp add: prime-def)
        moreover from m have nat m ≠ a by auto
        ultimately have nat m = 1 by auto
        hence m = 1 by arith }
      thus !!m. m ≥ 0 ∧ m dvd int a ∧ m ≠ int a ⟹ m=1 by auto
    qed
    ultimately show ?thesis by (auto simp add: zprime-def)
  qed
qed

lemma zprime-factor-exists: (a::int) > 1 ⟹ ∃ p. zprime p ∧ p dvd a
proof −
  assume a1: a > 1 hence a: int (nat a) = a by (auto simp add: int-nat-eq)
  with a1 have nat a > 1 by auto
  hence ∃ l. primel l ∧ prod l = nat a by (simp only: factor-exists)
  then obtain l where l: primel l ∧ prod l = nat a by (auto)
  show ?thesis
  proof (cases l)
    case Nil with l have nat a = 1 by auto
```

```
    with a1 show ?thesis by arith
  next
    case (Cons p ps)
    with l have nat a = p*prod ps and p: prime p by (auto simp add: primel-def)
    hence int (nat a) = (int p)*int(prod ps)
      by (auto simp add: int-mult)
    with a p have zprime (int p) ∧ int p dvd a
      by (auto simp add: prime-impl-zprime-int)
    thus ?thesis by blast
  qed
qed

lemma best-division-abs: (x::int) > 0 ⟹ ∃ n. 2 * |y − n*x| ≤ x
proof −
  assume x0: x>0
  then obtain b where b: b ≥ 0 ∧ b < x ∧ [y = b] (mod x)
    by (blast dest: zcong-zless-unique)
  hence x dvd (y−b) by (simp only: zcong-def)
  then obtain m where y−b = x*m by (auto simp add: dvd-def)
  hence m: b = y − m*x by (simp only: mult-ac)
  show ?thesis
  proof (cases)
    assume 2*|b| ≤ x
    with m show ?thesis by auto
  next
    assume ¬ 2*|b| ≤ x
    with b have bx: 2*b > x by auto
    hence bx1: 2*(x−b) < x by auto
    from b have bx2: b−x < 0 by auto
    obtain n where n = m+1 by simp
    hence y − n*x = y − m*x − x by (simp only: zadd-zmult-distrib zmult-1)
    with m have n: y − n*x = b−x by simp
    with bx2 have pos: −y + n*x > 0 by simp
    moreover from n bx1 have 2*(−y + n*x) < x by simp
    ultimately have 2*|y − n*x| < x by simp
    hence 2*|y − n*x| ≤ x by (unfold zabs-def, auto)
    thus ?thesis by auto
  qed
qed

lemma best-odd-division-abs: ⟦ (x::int) > 0; x ∈ zOdd ⟧
  ⟹ ∃ n. 2 * |y − n*x| < x
proof −
  assume x>0 and odd: x ∈ zOdd
  then obtain n where n: 2 * |y − n*x| ≤ x by (auto dest: best-division-abs)
  moreover have x ≠ 2 * |y − n*x|
  proof (rule ccontr, clarsimp)
    assume x = 2*|y − n*x|
    hence x ∈ zEven by (unfold zEven-def, auto)
    with odd show False by (auto simp only: odd-iff-not-even)
  qed
  ultimately have 2*|y − n*x| < x by simp
```

thus *?thesis* **by** *auto*
qed

lemma *zprime-2*: *zprime 2*
apply (*auto simp add*: *zprime-def*)
apply (*frule zdvd-imp-le*, *simp*)
apply (*auto simp add*: *order-le-less dvd-def*)
done

lemma *zgcd1-iff-no-common-primedivisor*:
(*zgcd*(*a*,*b*)=*1*) = (¬(∃ *p*. *zprime p* ∧ *p dvd a* ∧ *p dvd b*))
proof (*rule ccontr*, *auto*)
fix *p* **assume** *ab*: *zgcd*(*a*,*b*)=*1* **and** *p dvd a* **and** *p dvd b* **and** *p*: *zprime p*
hence *p dvd a* ∧ *p dvd b* **by** *simp*
hence *p dvd zgcd*(*a*,*b*) **by** (*simp add*: *zgcd-greatest-iff*)
with *ab p* **show** *False* **by** (*unfold zprime-def*, *auto*)
next
let *?g* = *zgcd*(*a*,*b*)
assume *g1*: *?g* ≠ *1* **and** *ps*: ∀ *p*. *zprime p* ⟶ *p dvd a* ⟶ ¬ *p dvd b*
moreover have *?g* ≠ *0*
proof (*rule ccontr*, *simp*)
assume *?g=0* **hence** *nat*|*a*|=*0* ∧ *nat*|*b*|=*0*
by (*unfold zgcd-def*, *auto simp add*: *gcd-zero*)
hence *a=0* ∧ *b=0* **by** *arith*
hence *2 dvd a* ∧ *2 dvd b* **by** *simp*
with *ps* **show** *False* **by** (*auto simp add*: *zprime-2*)
qed
moreover have *?g* ≥ *0* **by** (*rule zgcd-geq-zero*)
ultimately have *?g* > *1* **by** *auto*
then obtain *p* **where** *zprime p* ∧ *p dvd ?g*
by (*frule-tac a=?g* **in** *zprime-factor-exists*, *auto*)
hence *zprime p* ∧ *p dvd a* ∧ *p dvd b* **by** (*simp add*: *zgcd-greatest-iff*)
with *ps* **show** *False* **by** *auto*
qed

lemma *pos-zmult-pos*: *a* > (*0*::*int*) ⟹ *a*∗*b* > *0* ⟹ *b* > *0*
apply (*case-tac b* = *0*, *auto*)
apply (*rule ccontr*, *subgoal-tac b* < *0*, *auto*)
apply (*subgoal-tac a*∗*b* < *a*∗*0*, *auto dest*: *zmult-zless-mono2*)
done

2.2 Parity of integers

lemma *power-preserves-even*: *n*>*0* ⟹ (*x*^*n* ∈ *zEven*) = (*x* ∈ *zEven*)
apply (*induct n*, *auto simp add*: *even-times-either*)
apply (*case-tac n*≠*0*, *auto dest*: *even-product*)
done

lemma *power-preserves-odd*: *n*>*0* ⟹ (*x*^*n* ∈ *zOdd*) = (*x* ∈ *zOdd*)
apply (*induct n*, *auto*, *rule odd-mult-odd-prop*, *auto*)
apply (*case-tac n*≠*0*, *auto dest*: *odd-times-odd*)
done

lemma *even-plus-odd*: $a \in zEven \Longrightarrow b \in zOdd \Longrightarrow a+b \in zOdd$
apply (*auto simp add*: *zEven-def zOdd-def*)
apply (*rule-tac x=k+ka* **in** *exI*, *auto*)
done

lemma *odd-plus-odd*: $[\![x \in zOdd; y \in zOdd]\!] \Longrightarrow x+y \in zEven$
apply (*auto simp add*: *zOdd-def zEven-def*)
apply (*rule-tac x=1+k+ka* **in** *exI*, *auto*)
done

lemma *odd-plus-odd*: $a \in zOdd \Longrightarrow b \in zOdd \Longrightarrow a+b \in zEven$
apply (*auto simp add*: *zEven-def zOdd-def*)
apply (*rule-tac x=1+k+ka* **in** *exI*, *auto*)
done

lemma *even-plus-odd-prop1*: $a+b \in zOdd \Longrightarrow a \in zOdd \Longrightarrow b \in zEven$
by (*subgoal-tac a+b − a ∈ zEven*, *auto dest*: *odd-minus-odd*)

lemma *even-plus-odd-prop2*: $a+b \in zOdd \Longrightarrow a \in zEven \Longrightarrow b \in zOdd$
by (*subgoal-tac a+b − a ∈ zOdd*, *auto dest*: *odd-minus-even*)

2.3 Powers of natural numbers

lemma *gcd-1-power-left-distrib*: $gcd(a,b)=1 \Longrightarrow gcd(a\hat{}n,b)=1$
by (*induct n*, *auto simp add*: *gcd-mult-cancel*)

NB: the next (identical) lemma is just added to illustrate the difference between Isar and Isabelle scripting.

lemma *alternative-gcd-1-power-left-distrib*: $gcd(a,b)=1 \Longrightarrow gcd(a\hat{}n,b)=1$
proof −
assume *ab*: $gcd(a,b)=1$
thus $gcd(a\hat{}n,b)=1$
proof (*induct n*)
case *0*
show $gcd(a\hat{}0,b)=1$ **by** *auto*
next
case (*Suc n*)
hence $gcd(a\hat{}n,b)=1$ **by** *simp*
with *ab* **have** $gcd(a*a\hat{}n,b)=1$ **by** (*simp only*: *gcd-mult-cancel*)
thus $gcd(a\hat{}Suc\ n,b)=1$ **by** *simp*
qed
qed

lemma *gcd-1-power-distrib*: $gcd(a,b) = 1 \Longrightarrow gcd(a\hat{}n,b\hat{}n)=1$
proof −
assume $gcd(a,b)=1$
hence $gcd(a\hat{}n,b)=1$ **by** (*rule gcd-1-power-left-distrib*)
hence $gcd(b,a\hat{}n)=1$ **by** (*simp only*: *gcd-commute*)
hence $gcd(b\hat{}n,a\hat{}n)=1$ **by** (*rule gcd-1-power-left-distrib*)
thus $gcd(a\hat{}n,b\hat{}n)=1$ **by** (*simp only*: *gcd-commute*)
qed

lemma *gcd-power-distrib*: *gcd(a,b) ^n = gcd(a^n,b^n)*
proof *cases*
assume *a=0 ∧ b=0*
thus *?thesis* **by** (*auto simp add*: *power-0-left*)
next
let *?g = gcd(a,b)*
assume ¬ (*a=0 ∧ b=0*)
hence *a ≠ 0 ∨ b ≠ 0* **by** *simp*
then obtain *c d* **where** *abcd*: *a = ?g*c ∧ b = ?g*d ∧ gcd(c,d)=1*
by (*frule-tac a=a* **in** *make-relprime*, *auto*)
moreover have *(?g*c) ^n = ?g^n * c^n ∧ (?g*d) ^n = ?g^n * d^n*
by (*simp add*: *power-mult-distrib*)
ultimately have *gcd(a^n,b^n) = ?g^n * gcd(c^n,d^n)* **by** (*simp only*: *gcd-mult-distrib2*)
moreover from *abcd* **have** *gcd(c^n,d^n) = 1* **by** (*simp only*: *gcd-1-power-distrib*)
ultimately show *?thesis* **by** *simp*
qed

Useful lemma: if prime $p|a^n$ then $p|a$.

lemma *prime-dvd-power*: ⟦ *prime p*; *p dvd a^n* ⟧ ⟹ *p dvd a*
proof (*induct n*)
case *0* **hence** *prime p ∧ p = 1* **by** *auto*
thus *?thesis* **by** *auto*
next case (*Suc n*) **hence** *IH*: *prime p ∧ p dvd a^n ⟹ p dvd a* **by** *auto*
assume *p*: *prime p* **and** *p dvd a^Suc n*
hence *p dvd a*a^n* **by** *simp*
with *p* **have** *p dvd a ∨ p dvd a^n* **by** (*simp add*: *prime-dvd-mult*)
with *IH* **and** *p* **show** *p dvd a* **by** *auto*
qed

lemma *prime-power-dvd-cancel-right*:
⟦ *prime p*; ¬ *p dvd b*; *p^n dvd a*b* ⟧ ⟹ *p^n dvd a*
proof −
assume *p*: *prime p* **and** *pb*: ¬ *p dvd b*
hence *p1*: *p>1* **by** (*simp add*: *prime-def*)
have !!*a*. *p^n dvd a*b* ⟶ *p^n dvd a*
proof (*induct n*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc n*) **hence** *IH*: !!*a*. *p^n dvd a*b* ⟶ *p^n dvd a* **..**
fix *a* **show** *p^Suc n dvd a*b* ⟶ *p^Suc n dvd a*
proof (*auto*)
assume *ppnab*: *p*p^n dvd a*b*
hence *p dvd a*b* **by** (*rule dvd-mult-left*)
with *p* **have** *p dvd a ∨ p dvd b* **by** (*rule prime-dvd-mult*)
with *pb* **have** *p dvd a* **by** *simp*
then obtain *k* **where** *apk*: *a = p * k* **by** (*auto simp add*: *dvd-def*)
with *ppnab* **have** *p*p^n dvd p*(k*b)* **by** (*auto simp add*: *mult-ac*)
with *p1* **have** *p^n dvd k*b* **by** (*auto dest*: *dvd-mult-cancel*)
with *IH* **have** *p^n dvd k* **..**
with *apk* **show** *p*p^n dvd a* **by** (*simp add*: *mult-dvd-mono*)
qed

```
  qed
  thus p^n dvd a*b ⟹ p^n dvd a ..
qed
```

Helping lemma: if $n > 0$ then $a^n | b^n \iff a | b$.

```
lemma nat-power-dvd-mono: n ≠ 0 ⟹ (a^n dvd b^n) = (a dvd (b::nat))
proof
  assume n ≠ 0
  then obtain m where mn: n = Suc m
    by (frule-tac n=n in not0-implies-Suc, auto)
  assume a^n dvd b^n
  then obtain k where k: b^n = a^n * k by (auto simp add: dvd-def)
 moreover have gcd(a^n, (a^n)*k) = (a^n) * gcd(1,k) by (simp add: gcd-mult-distrib2)
  ultimately have gcd(a^n,b^n) = a^n by (auto simp add: gcd-commute gcd-1)
  hence gcd(a,b)^n = a^n by (simp add: gcd-power-distrib)
  with mn have a = gcd(a,b) by (rule-tac n=m in power-inject-base, auto)
  moreover have gcd(a,b) dvd b by (rule gcd-dvd2)
  ultimately show a dvd b by simp
next
  assume a dvd b
  then obtain k where b = a * k by (auto simp add: dvd-def)
  hence b^n = a^n * k^n by (simp only: power-mult-distrib)
  thus a^n dvd b^n by auto
qed
```

Theorem: if $n > 0$ and $\gcd(a, b) = 1$ and $ab = c^n$ then $\exists k : a = k^n$. Proof uses induction on the number of prime factors of c.

```
theorem nat-relprime-power-divisors:
  assumes npos: n≠0 and abcn: a*b = c^n and relprime: gcd(a,b) = 1
  shows ∃ k. a = k^n
proof −
  from npos obtain m where mn: n = Suc m
    by (frule-tac n=n in not0-implies-Suc, auto)
  show ?thesis
  proof (case-tac c=0)
    assume c=0 with abcn npos mn have a*b = 0 by (auto simp only: power-0-Suc)
    hence a=0 ∨ b=0 by auto
    moreover
    { assume a=0 with mn npos have a = 0^n by (auto simp only: power-0-Suc)
      hence ?thesis by blast }
    moreover
    { assume b=0 with relprime have a = 1^n by (auto simp only: gcd-0 power-one)
      hence ?thesis by blast }
    ultimately show ?thesis by blast
  next
    assume c≠0 then obtain xs where xs: primel xs ∧ prod xs = c
      by (frule-tac a=c in factor-exists-general, auto)
    moreover have
      !!a b. (primel xs ∧ a*b = (prod xs)^n ∧ gcd(a,b)=1) ⟹ ∃k. a = k^n
    proof (induct xs)
      case Nil hence ass: a*b=1^n by simp
```

```
  hence a*b=1 by (simp only: power-one)
  hence b=1 by simp
  with ass show ∃ k. a = k^n by auto
next
  case (Cons p ps)
  hence ass: primel ps ∧ prime p ∧ a*b=p^n*(prod ps) ^n ∧ gcd(a,b)=1
    and IH: !!a b. primel ps ∧ a*b = (prod ps) ^n ∧ gcd(a,b)=1 ⟹ ∃ k. a = k^n
    by (auto simp add: primel-def power-mult-distrib)
  hence pnab: p^n dvd a*b and pn0: p^n ≠ 0 by (auto simp add: prime-def)
  moreover
  { assume pa: p dvd a
    have ¬ p dvd b
    proof (rule ccontr, simp)
      assume p dvd b
      with pa have p dvd gcd(a,b) by (simp add: gcd-greatest-iff)
      with ass show False by (auto simp add: prime-def)
    qed
    with ass pnab have p^n dvd a by (simp add: prime-power-dvd-cancel-right)
    then obtain A where A: a = p^n * A by (auto simp add: dvd-def)
    with ass pn0 have A*b = (prod ps) ^n by auto
    moreover have gcd(A,b)=1
    proof −
      let ?g = gcd(A,b)
      have ?g dvd A ∧ ?g dvd b by (simp add: gcd-greatest)
      with A have ?g dvd a ∧ ?g dvd b by (simp add: dvd-mult)
      hence ?g dvd gcd(a,b) by (simp only: gcd-greatest)
      with ass show ?g = 1 by auto
    qed
    moreover from IH ass have
      A*b = (prod ps) ^n ∧ gcd(A,b)=1 ⟹ ∃ k. A = k^n by auto
    ultimately have ∃ k. A = k^n by auto
    then obtain k where A = k^n by auto
    with A have a = (p*k) ^n by (auto simp add: power-mult-distrib)
    hence ∃ k. a = k^n by blast }
  moreover
  { assume ¬ p dvd a
    moreover from ass pnab have p^n dvd b*a ∧ prime p
      by (auto simp only: mult-ac)
    ultimately have p^n dvd b by (simp add: prime-power-dvd-cancel-right)
    then obtain B where B: b = p^n * B by (auto simp add: dvd-def)
    with ass pn0 have a*B = (prod ps) ^n by auto
    moreover have gcd(a,B)=1
    proof −
      let ?g = gcd(a,B)
      have ?g dvd a ∧ ?g dvd B by (simp add: gcd-greatest)
      with B have ?g dvd a ∧ ?g dvd b by (simp add: dvd-mult)
      hence ?g dvd gcd(a,b) by (simp only: gcd-greatest)
      with ass show ?g = 1 by auto
    qed
    moreover from IH ass have
      a*B = (prod ps) ^n ∧ gcd(a,B)=1 ⟹ ∃ k. a = k^n by auto
    ultimately have ∃ k. a = k^n by auto }
```

```
    ultimately show ∃ k. a = k^n by auto
  qed
  moreover from abcn relprime have gcd(a,b)=1 ∧ a*b=c^n by simp
  ultimately show ?thesis by auto
 qed
qed
```

2.4 Powers of integers

Now turn to the case of integers. This lemma is based on its equivalent for the natural numbers.

```
corollary int-relprime-power-divisors:
 assumes abcn: a*b = c^n and n: n>1 and relprime: zgcd(a,b) = 1
 shows ∃ k. |a| = k^n
proof −
 let ?a1 = nat|a|
 let ?b1 = nat|b|
 let ?c1 = nat|c|
 from relprime have absrelprime: gcd(?a1, ?b1)=1 by (auto simp only: zgcd-def)
 have |a*b| = |a|*|b| by (simp add: abs-mult)
 with abcn have |c|^n = |a|*|b| by (simp add: power-abs)
 hence int(?c1^n) = int(?a1*?b1) by (simp only: int-nat-abs-eq-abs int-mult int-power)
 hence ?a1*?b1 = ?c1^n by (simp only: int-int-eq)
 with absrelprime and n have ∃ k. ?a1 = k^n by (simp only: nat-relprime-power-divisors)
 then obtain k::nat where alpha: ?a1 = k^n by auto
 moreover have int ?a1 = |a| by (simp add: int-nat-eq)
 ultimately have |a| = int(k^n) by simp
 hence |a| = int(k)^n by (simp only: int-power)
 thus ?thesis by auto
qed

corollary int-triple-relprime-power-divisors:
 ⟦ a*b*c = d^n; n > 1; zgcd(a,b)=1; zgcd(b,c)=1; zgcd(c,a)=1 ⟧
 ⟹ ∃ k l m. |a| = k^n ∧ |b| = l^n ∧ |c| = m^n
proof −
 assume abcd: a*b*c = d^n and n1: n > 1
   and ab: zgcd(a,b)=1 and bc: zgcd(b,c)=1 and ca: zgcd(c,a)=1
 hence ba: zgcd(b,a)=1 and cb: zgcd(c,b)=1 and ac: zgcd(a,c)=1
   by (auto simp only: zgcd-commute)
 from ba ca have zgcd(b*c,a)=1 by (simp only: zgcd-zmult-cancel)
 with abcd have a*(b*c) = d^n ∧ zgcd(a,b*c)=1 by (simp add: zgcd-commute)
 with n1 have k: ∃ k. |a| = k^n by (auto dest: int-relprime-power-divisors)
 from ab cb have zgcd(a*c,b)=1 by (simp only: zgcd-zmult-cancel)
 with abcd have b*(a*c) = d^n ∧ zgcd(b,a*c)=1
   by (simp add: zgcd-commute mult-ac)
 with n1 have l: ∃ l. |b| = l^n by (auto dest: int-relprime-power-divisors)
 from ac bc have zgcd(a*b,c)=1 by (simp only: zgcd-zmult-cancel)
 with abcd have c*(a*b) = d^n ∧ zgcd(c,a*b)=1
   by (simp add: zgcd-commute mult-ac)
 with n1 have m: ∃ m. |c| = m^n by (auto dest: int-relprime-power-divisors)
 from k l m show ?thesis by auto
```

qed

lemma *neg-odd-power*: $[\![\ x \in zOdd;\ x \geq 0\ ]\!] \Longrightarrow (-a::int)\hat{}(nat\ x) = -(a\hat{}(nat\ x))$
proof −
assume $x \in zOdd$ **and** $0 \leq x$
hence $-(a\hat{}(nat\ x)) = (-1)\hat{}(nat\ x) * a\hat{}(nat\ x)$ **by** (*simp add*: *neg-one-odd-power*)
also have $\ldots = (-1*a)\hat{}(nat\ x)$ **by** (*simp only*: *power-mult-distrib*)
finally show *?thesis* **by** *simp*
qed

lemma *neg-even-power*: $[\![\ x \in zEven;\ x \geq 0\ ]\!] \Longrightarrow (-a::int)\hat{}(nat\ x) = a\hat{}(nat\ x)$
proof −
assume $x \in zEven$ **and** $x \geq 0$
hence $a\hat{}(nat\ x) = (-1)\hat{}(nat\ x) * a\hat{}(nat\ x)$ **by** (*simp add*: *neg-one-even-power*)
also have $\ldots = (-1*a)\hat{}(nat\ x)$ **by** (*simp only*: *power-mult-distrib*)
finally show *?thesis* **by** *simp*
qed

corollary *int-relprime-odd-power-divisors*:
$[\![\ a*b = c\hat{}(nat\ x);\ nat\ x > 1;\ x \in zOdd;\ zgcd(a,b) = 1\ ]\!] \Longrightarrow \exists\ k.\ a = k\hat{}(nat\ x)$
proof −
assume $a*b = c\hat{}(nat\ x)$ **and** *x1*: $nat\ x > 1$ **and** *odd*: $x \in zOdd$ **and** $zgcd(a,b)=1$
hence $\exists\ k.\ |a| = k\hat{}(nat\ x)$ **by** (*simp only*: *int-relprime-power-divisors*)
then obtain k **where** *k*: $|a| = k\hat{}(nat\ x)$ **by** *blast*
{ assume $a \neq k\hat{}(nat\ x)$
with k **have** $a = -(k\hat{}(nat\ x))$ **by** *arith*
with *x1 odd* **have** $a = (-k)\hat{}(nat\ x)$ **by** (*simp add*: *neg-odd-power*) **}**
thus *?thesis* **by** *blast*
qed

corollary *int-triple-relprime-odd-power-divisors*:
$[\![\ a*b*c = d\hat{}(nat\ x);\ nat\ x>1;\ x\in zOdd;\ zgcd(a,b)=1;\ zgcd(b,c)=1;\ zgcd(c,a)=1\ ]\!]$
$\Longrightarrow \exists\ k\ l\ m.\ a = k\hat{}(nat\ x) \wedge b = l\hat{}(nat\ x) \wedge c = m\hat{}(nat\ x)$
proof −
assume *abcd*: $a*b*c = d\hat{}(nat\ x)$ **and** *x1*: $nat\ x > 1$ **and** *odd*: $x \in zOdd$
and *ab*: $zgcd(a,b)=1$ **and** *bc*: $zgcd(b,c)=1$ **and** *ca*: $zgcd(c,a)=1$
hence *ba*: $zgcd(b,a)=1$ **and** *cb*: $zgcd(c,b)=1$ **and** *ac*: $zgcd(a,c)=1$
by (*auto simp only*: *zgcd-commute*)
{ from *ba ca* **have** $zgcd(b*c,a)=1$ **by** (*simp only*: *zgcd-zmult-cancel*)
with *abcd* **have** $a*(b*c) = d\hat{}(nat\ x) \wedge zgcd(a,b*c)=1$
by (*simp add*: *zgcd-commute*)
with *x1 odd* **have** $\exists\ k.\ a = k\hat{}(nat\ x)$
by (*auto dest*: *int-relprime-odd-power-divisors*) **}**
moreover
{ from *ab cb* **have** $zgcd(a*c,b)=1$ **by** (*simp only*: *zgcd-zmult-cancel*)
with *abcd* **have** $b*(a*c) = d\hat{}(nat\ x) \wedge zgcd(b,a*c)=1$
by (*simp add*: *zgcd-commute mult-ac*)
with *x1 odd* **have** $\exists\ l.\ b = l\hat{}(nat\ x)$
by (*auto dest*: *int-relprime-odd-power-divisors*) **}**
moreover
{ from *ac bc* **have** $zgcd(a*b,c)=1$ **by** (*simp only*: *zgcd-zmult-cancel*)
with *abcd* **have** $c*(a*b) = d\hat{}(nat\ x) \wedge zgcd(c,a*b)=1$

```
    by (simp add: zgcd-commute mult-ac)
  with x1 odd have m: ∃ m. c = m^(nat x)
    by (auto dest: int-relprime-odd-power-divisors) }
 ultimately show ?thesis by auto
qed

lemma zgcd-1-power-left-distrib: zgcd(a,b)=1 ⟹ zgcd(a^n,b)=1
 by (induct n, auto simp add: zgcd-zmult-cancel)

lemma zgcd-1-power-distrib: zgcd(a,b) = 1 ⟹ zgcd(a^n,b^n)=1
proof −
 assume zgcd(a,b)=1
 hence zgcd(a^n,b)=1 by (rule zgcd-1-power-left-distrib)
 hence zgcd(b,a^n)=1 by (simp only: zgcd-commute)
 hence zgcd(b^n,a^n)=1 by (rule zgcd-1-power-left-distrib)
 thus zgcd(a^n,b^n)=1 by (simp only: zgcd-commute)
qed

lemma zgcd-power-distrib: zgcd(a,b) ^n = zgcd(a^n,b^n)
proof cases
 assume a=0 ∧ b=0
 thus ?thesis by (auto simp add: power-0-left)
next
 let ?g = zgcd(a,b)
 assume ¬ (a=0 ∧ b=0)
 hence ab0: a ≠ 0 ∨ b ≠ 0 by simp
 hence non0: zgcd(a,b) ≠ 0 ∧ zgcd(a^n,b^n) ≠ 0
   by (auto simp add: zgcd-def gcd-zero power-eq-0-iff)
 moreover have zgcd(a,b) ≥ 0 ∧ zgcd(a^n,b^n) ≥ 0 by (simp add: zgcd-geq-zero)
 ultimately have zgcd(a,b) ^n > 0 ∧ zgcd(a^n,b^n) > 0
   by (auto simp add: zero-less-power)
 moreover from ab0 obtain c d where abcd: a = ?g∗c ∧ b = ?g∗d ∧ zgcd(c,d)=1
   by (frule-tac a=a in make-zrelprime, auto)
 moreover have (?g∗c) ^n = ?g^n ∗ c^n ∧ (?g∗d) ^n = ?g^n ∗ d^n
   by (simp add: power-mult-distrib)
 ultimately have gabcdn: zgcd(a^n,b^n) = ?g^n ∗ zgcd(c^n,d^n)
   by (auto simp add: zgcd-zmult-distrib2)
 moreover from abcd have zgcd(c^n,d^n) = 1 by (simp only: zgcd-1-power-distrib)
 ultimately show ?thesis by auto
qed

lemma zprime-zdvd-zmult-general: ⟦ zprime p; p dvd m∗n ⟧ ⟹ p dvd m ∨ p dvd n
 apply (case-tac m ≥ 0, simp only: zprime-zdvd-zmult)
 apply (subgoal-tac −m ≥ 0 ∧ p dvd (−m)∗n, subgoal-tac p dvd −m ∨ p dvd n)
 prefer 2
 apply (rule-tac m=−m in zprime-zdvd-zmult, auto)
done

lemma zprime-zdvd-power: ⟦ zprime p; p dvd a^n ⟧ ⟹ p dvd a
 apply (induct n, auto)
 prefer 2
 apply (frule-tac m=a and n=a^n in zprime-zdvd-zmult-general)
```

```
  apply (auto, simp add: zprime-def zdvd-not-zless)
done

lemma zpower-zdvd-mono: n ≠ 0 ⟹ (a^n dvd b^n) = (a dvd (b::int))
proof
  assume n ≠ 0
  then obtain m where mn: n = Suc m
    by (frule-tac n=n in not0-implies-Suc, auto)
  assume a^n dvd b^n
  then obtain k where k: b^n = a^n * k by (auto simp add: dvd-def)
  moreover have zgcd(a^n*1, a^n*k) = |a^n| * zgcd(1,k)
    by (rule-tac k=a^n in zgcd-zmult-distrib2-abs)
  ultimately have zgcd(a^n,b^n) = |a^n|
    by (auto simp add: zgcd-commute zgcd-1)
  hence zgcd(a,b)^n = |a|^n ∧ zgcd(a,b) ≥ 0 ∧ |a| ≥ 0
    by (simp add: zgcd-power-distrib power-abs zgcd-geq-zero)
  with mn have |a| = zgcd(a,b) by (rule-tac n=m in power-inject-base, auto)
  moreover have zgcd(a,b) dvd b by (rule-tac m=a in zgcd-zdvd2)
  ultimately have |a| dvd b by simp
  thus a dvd b by (simp add: zdvd-abs1)
next
  assume a dvd b
  then obtain k where k: b = a * k by (auto simp add: dvd-def)
  hence b^n = a^n * k^n by (simp only: power-mult-distrib)
  thus a^n dvd b^n by auto
qed

lemma zprime-power-zdvd-cancel-right:
  ⟦ zprime p; ¬ p dvd b; p^n dvd a*b ⟧ ⟹ p^n dvd a
proof −
  assume p: zprime p and pb: ¬ p dvd b
  hence p1: p > 1 by (simp add: zprime-def)
  have !!a. p^n dvd a*b ⟶ p^n dvd a
  proof (induct n)
    case 0 thus ?case by auto
  next
    case (Suc n) hence IH: !!a. p^n dvd a*b ⟶ p^n dvd a ..
    fix a show p^Suc n dvd a*b ⟶ p^Suc n dvd a
    proof (auto)
      assume ppnab: p*p^n dvd a*b
      hence p dvd a*b by (auto simp add: dvd-def mult-assoc)
      with p have p dvd a ∨ p dvd b by (rule zprime-zdvd-zmult-general)
      with pb have p dvd a by simp
      then obtain k where apk: a = p*k by (auto simp add: dvd-def)
      with ppnab have p*p^n dvd p*(k*b) by (auto simp add: mult-ac)
      with p1 have p^n dvd k*b by (auto dest: zdvd-mult-cancel)
      with IH have p^n dvd k ..
      with apk show p*p^n dvd a by (simp add: zdvd-zmult-mono)
    qed
  qed
  thus p^n dvd a*b ⟹ p^n dvd a ..
qed
```

lemma *zprime-power-zdvd-cancel-left*:
  ⟦ *zprime p*; ¬ *p dvd a*; *p^n dvd a*∗*b* ⟧ ⟹ *p^n dvd b*
  apply (*subgoal-tac p^n dvd b*∗*a*)
  apply (*auto dest*: *zprime-power-zdvd-cancel-right*)
  apply (*simp add*: *mult-ac*)
done

2.5 Facts about small powers of integers

lemma *zadd-power2*: ((*a*::*int*)+*b*)^*2* = *a^2* + *2*∗*a*∗*b* + *b^2*
  by (*simp add*: *nat-number ring-simps*)

lemma *zdiff-power2*: ((*a*::*int*)−*b*)^*2* = *a^2* − *2*∗*a*∗*b* + *b^2*
  by (*simp add*: *nat-number ring-simps*)

lemma *zspecial-product*: ((*a*::*int*) + *b*) ∗ (*a* − *b*) = *a^2* − *b^2*
  by (*simp add*: *nat-number ring-simps*)

lemma *abs-power2-distrib*: |*a^2*| = |*a*::*int*|^*2*
  by (*simp add*: *power2-eq-square abs-mult*)

lemma *power2-eq-iff-abs-eq*: ((*a*::*int*)^*2* = *b^2*) = (|*a*| = |*b*|)
proof
  assume *a^2* = *b^2*
  hence (*a*+*b*)∗(*a*−*b*) = *0* **by** (*simp add*: *zspecial-product*)
  hence *a*=*b* ∨ *a*=−*b* **by** *auto*
  thus |*a*| = |*b*| **by** *auto*
next
  assume |*a*| = |*b*|
  hence |*a*|^*2* = |*b*|^*2* **by** *simp*
  thus *a^2* = *b^2* **by** (*simp only*: *power2-abs*)
qed

lemma *power2-eq1-iff*: (*a*::*int*)^*2* = *1* ⟹ |*a*|=*1*
  by (*auto simp add*: *zmult-eq-1-iff power2-eq-square abs-mult*)

lemma *zadd-power3*: ((*a*::*int*)+*b*)^*3* = *a^3* + *3*∗*a^2*∗*b* + *3*∗*a*∗*b^2* + *b^3*
  by (*simp add*: *nat-number ring-simps*)

lemma *zdiff-power3*: ((*a*::*int*)−*b*)^*3* = *a^3* − *3*∗*a^2*∗*b* + *3*∗*a*∗*b^2* − *b^3*
  by (*simp add*: *nat-number ring-simps*)

lemma *power3-minus*: (−*a*::*int*)^*3* = −(*a^3*)
proof −
  have (*3*::*int*) ∈ *zOdd* ∧ (*3*::*int*) ≥ *0* **by** (*unfold zOdd-def*, *auto*)
  hence (−*a*)^(*nat 3*) = −(*a*^(*nat 3*)) **by** (*simp only*: *neg-odd-power*)
  thus *?thesis* **by** *simp*
qed

lemma *abs-power3-distrib*: |(*x*::*int*)^*3*| = |*x*|^*3*
  by (*simp add*: *nat-number ring-simps abs-mult*)

lemma *cube-square*: (*a*::*int*)∗*a* ^ *2* = *a* ^ *3*
by (*simp add*: *nat-number ring-simps*)

lemma *quartic-square-square*: (*x* ^ *2*) ^ *2* = (*x*::*int*) ^ *4*
by (*simp add*: *nat-number ring-simps*)

lemma *power2-ge-self*: *x* ^ *2* ≥ (*x*::*int*)
proof (*cases*)
assume *nonpos*: *x*≤*0*
have *0* ≤ *x* ^ *2* **by** (*rule zero-le-power2*)
with *nonpos* **show** *?thesis* **by** (*rule zle-trans*)
next
assume ¬ *x* ≤ *0* **hence** *x1*: *x* ≥ *1* **by** *simp*
thus *?thesis*
proof (*cases*)
assume *x* = *1*
thus *?thesis* **by** *simp*
next
assume ¬ *x* = *1* **with** *x1* **have** *x2*: *1*<*x* **by** *simp*
hence *0*<*x* **by** *simp*
with *x2* **have** *x*∗*1* < *x*∗*x* **by** (*rule zmult-zless-mono2*)
thus *?thesis* **by** (*simp only*: *power2-eq-square*)
qed
qed

end

3 Pythagorean triples and Fermat's last theorem, case $n = 4$

theory *Fermat4*
imports *InfDesc IntNatAux Parity*
begin

Proof of Fermat's last theorem for the case $n = 4$:

$$\forall x, y, z : \ x^4 + y^4 = z^4 \Longrightarrow xyz = 0.$$

lemma *even-eq-two-dvd*: *even* (*r*::*nat*) = (*2 dvd r*)
apply *safe*
apply (*simp only*: *even-nat-equiv-def2*, *arith*)
apply (*auto simp add*: *even-def dvd-eq-mod-eq-0*)
done

lemma *nat-power2-add*: ((*a*::*nat*)+*b*) ^ *2* = *a* ^ *2* + *b* ^ *2* + *2*∗*a*∗*b*
proof −
have (*a*+*b*) ^ *2* = (*a*+*b*)∗(*a*+*b*) **by** (*rule power2-eq-square*)
also have ... = *a* ^ *2* + *2*∗(*a*∗*b*) + *b* ^ *2*
by (*simp only*: *add-mult-distrib add-mult-distrib2 mult-commute power2-eq-square*)
finally show *?thesis* **by** *simp*

qed

lemma *nat-power2-diff*: $a \geq (b::nat) \Longrightarrow (a-b)\hat{}2 = a\hat{}2 + b\hat{}2 - 2*a*b$
proof −
assume *a-ge-b*: $a \geq b$
hence *a2-ge-b2*: $a\hat{}2 \geq b\hat{}2$ **by** (*simp only*: *power-mono*)
from *a-ge-b* **have** *ab-ge-b2*: $a*b \geq b\hat{}2$ **by** (*simp add*: *power2-eq-square*)
have $b*(a-b) + (a-b)\hat{}2 = a*(a-b)$ **by** (*simp add*: *power2-eq-square diff-mult-distrib*)
also have $\ldots = a*b + a\hat{}2 + (b\hat{}2 - b\hat{}2) - 2*a*b$
by (*simp add*: *diff-mult-distrib2 power2-eq-square*)
also with *a2-ge-b2* **have** $\ldots = a*b + (a\hat{}2 - b\hat{}2) + b\hat{}2 - 2*a*b$ **by** *simp*
also with *ab-ge-b2* **have** $\ldots = (a*b - b\hat{}2) + a\hat{}2 + b\hat{}2 - 2*a*b$ **by** *auto*
also have $\ldots = b*(a-b) + a\hat{}2 + b\hat{}2 - 2*a*b$
by (*simp only*: *diff-mult-distrib2 power2-eq-square mult-commute*)
finally show *?thesis* **by** *arith*
qed

lemma *nat-power-le-imp-le-base*: $[\![\ n \neq 0;\ a\hat{}n \leq b\hat{}n\ ]\!] \Longrightarrow (a::nat) \leq b$
proof −
assume $n \neq 0$ **and** *ab*: $a\hat{}n \leq b\hat{}n$
then obtain *m* **where** $n = Suc\ m$ **by** (*frule-tac n=n* **in** *not0-implies-Suc*, *auto*)
with *ab* **have** $a \geq 0$ **and** $a\hat{}Suc\ m \leq b\hat{}Suc\ m$ **and** $b \geq 0$ **by** *auto*
thus *?thesis* **by** (*rule-tac n=m* **in** *power-le-imp-le-base*)
qed

lemma *nat-power-inject-base*: $[\![\ n \neq 0;\ a\hat{}n = b\hat{}n\ ]\!] \Longrightarrow (a::nat) = b$
proof −
assume $n \neq 0$ **and** *ab*: $a\hat{}n = b\hat{}n$
then obtain *m* **where** $n = Suc\ m$ **by** (*frule-tac n=n* **in** *not0-implies-Suc*, *auto*)
with *ab* **have** $a\hat{}Suc\ m = b\hat{}Suc\ m$ **and** $a \geq 0$ **and** $b \geq 0$ **by** *auto*
thus *?thesis* **by** (*rule power-inject-base*)
qed

3.1 Parametrisation of Pythagorean triples (over $\mathbb{N}$ and $\mathbb{Z}$)

theorem *nat-euclid-pyth-triples*:
assumes *abc*: $a\hat{}2 + b\hat{}2 = c\hat{}2$ **and** *ab-relprime*: $gcd(a,b)=1$ **and** *aodd*: *odd a*
shows $\exists\ p\ q.\ a = p\hat{}2 - q\hat{}2 \wedge b = 2*p*q \wedge c = p\hat{}2 + q\hat{}2 \wedge gcd(p,q)=1$
proof −
have *two0*: $(2::nat) \neq 0$ **by** *simp*
from *abc* **have** *a2cb*: $a\hat{}2 = c\hat{}2 - b\hat{}2$ **by** *arith*
— factor a^2 in coprime factors $(c-b)$ and $(c+b)$; hence both are squares
have *a2factor*: $a\hat{}2 = (c-b)*(c+b)$
proof −
have $c*b - c*b = 0$ **by** *simp*
with *a2cb* **have** $a\hat{}2 = c*c + c*b - c*b - b*b$ **by** (*simp add*: *power2-eq-square*)
also have $\ldots = c*(c+b) - b*(c+b)$
by (*simp add*: *add-mult-distrib2 add-mult-distrib mult-commute*)
finally show *?thesis* **by** (*simp only*: *diff-mult-distrib*)
qed
have *a-nonzero*: $a \neq 0$
proof (*rule ccontr*)

```
  assume ¬ a ≠ 0 hence a = 0 by simp
  with aodd have odd (0::nat) by simp
  thus False by simp
qed
have b-less-c: b < c
proof −
  from abc have b^2 ≤ c^2 by auto
  with two0 have b ≤ c by (rule-tac n=2 in nat-power-le-imp-le-base)
  moreover haveb ≠ c
  proof
    assume b=c with a2cb have a^2 = 0 by simp
    with a-nonzero show False by (simp add: power2-eq-square)
  qed
  ultimately show ?thesis by auto
qed
hence b2-le-c2: b^2 ≤ c^2 by (simp add: power-mono)
have bc-relprime: gcd(b,c) = 1
proof −
  from b2-le-c2 have cancelb2: c^2−b^2+b^2 = c^2 by auto
  let ?g = gcd(b,c)
  have ?g^2 = gcd(b^2,c^2) by (simp only: gcd-power-distrib)
  with cancelb2 have ?g^2 = gcd(b^2,c^2−b^2+b^2) by simp
  hence ?g^2 = gcd(b^2, c^2−b^2) by simp
  with a2cb have ?g^2 dvd a^2 by (simp only: gcd-dvd2)
  hence ?g dvd a ∧ ?g dvd b by (simp add: nat-power-dvd-mono gcd-dvd1)
  hence ?g dvd gcd(a,b) by (simp only: gcd-greatest)
  with ab-relprime show ?thesis by auto
qed
have p2: prime 2 by (rule two-is-prime)
have factors-odd: odd (c−b) ∧ odd (c+b)
proof (auto simp only: ccontr)
  assume even (c−b) hence 2 dvd c−b by (simp only: even-eq-two-dvd)
  with a2factor have 2 dvd a^2 by (simp only: dvd-mult2)
  with p2 have 2 dvd a by (rule prime-dvd-power)
  hence even a by (simp only: even-eq-two-dvd)
  with aodd show False by simp
next
  assume even (c+b) hence 2 dvd c+b by (simp only: even-eq-two-dvd)
  with a2factor have 2 dvd a^2 by (simp only: dvd-mult)
  with p2 have 2 dvd a by (rule prime-dvd-power)
  hence even a by (simp only: even-eq-two-dvd)
  with aodd show False by simp
qed
have cb1: c−b + (c+b) = 2*c
proof −
  have c−b + (c+b) = ((c−b)+b)+c by simp
  also with b-less-c have ... = (c+b−b)+c by (simp only: diff-add-assoc2)
  also have ... = c+c by simp
  finally show ?thesis by simp
qed
have cb2: 2*b + (c−b) = c+b
proof −
```

```
  have 2*b + (c-b) = b+b + (c - b) by auto
  also have ... = b + ((c-b)+b) by simp
  also with b-less-c have  ... = b + (c+b-b) by (simp only: diff-add-assoc2)
  finally show ?thesis by simp
qed
have factors-relprime: gcd(c-b,c+b) =1
proof -
  let ?g = gcd(c-b,c+b)
  have cb1: c-b + (c+b) = 2*c
  proof -
    have c-b + (c+b) = ((c-b)+b)+c by simp
    also with b-less-c have ... = (c+b-b)+c by (simp only: diff-add-assoc2)
    also have ... = c+c by simp
    finally show ?thesis by simp
  qed
  have ?g = gcd(c-b + (c+b),c+b) by simp
  with cb1 have ?g = gcd(2*c,c+b) by (rule-tac a=c-b + (c+b) in back-subst)
  hence g2c: ?g dvd 2*c by (simp only: gcd-dvd1)
  have gcd(c-b,2*b + (c-b)) = gcd(c-b,2*b) by simp
  with cb2 have ?g = gcd(c-b,2*b) by (rule-tac a=2*b + (c-b) in back-subst)
  hence g2b: ?g dvd 2*b by (simp only: gcd-dvd2)
  with g2c have ?g dvd 2*gcd(b, c) by (simp only: gcd-greatest gcd-mult-distrib2)
  with bc-relprime have ?g dvd 2 by simp
  with p2 have g1or2: ?g = 2 ∨ ?g = 1 by (unfold prime-def, auto)
  thus ?thesis
  proof (auto)
    assume ?g = 2 hence 2 dvd ?g by simp
    hence 2 dvd c-b by (simp add: gcd-dvd1)
    with factors-odd show False by (simp add: even-eq-two-dvd)
  qed
qed
from a2factor have (c-b)*(c+b) = a^2 and (2::nat) >1  by auto
with factors-relprime have ∃ k. c-b = k^2 by (simp only: nat-relprime-power-divisors)
then obtain r where r: c-b = r^2 by auto
from a2factor have (c+b)*(c-b) = a^2 and (2::nat) >1  by auto
with factors-relprime have ∃ k. c+b = k^2
  by (simp only: nat-relprime-power-divisors gcd-commute)
then obtain s where s: c+b = s^2 by auto
— now p := (s + r)/2 and q := (s − r)/2 is our solution
have rs-odd: odd r ∧ odd s
proof (auto dest: ccontr)
  assume even r hence 2 dvd r by (simp only: even-eq-two-dvd )
  with r have  2 dvd (c-b) by (simp only: power2-eq-square dvd-mult)
  hence even (c-b) by (simp only: even-eq-two-dvd)
  with factors-odd show False by simp
next
  assume even s hence 2 dvd s by (simp only: even-eq-two-dvd)
  with s have 2 dvd (c+b) by (simp only: power2-eq-square dvd-mult)
  hence even (c+b) by (simp only: even-eq-two-dvd)
  with factors-odd show False by auto
qed
obtain m where m: m = s-r by simp
```

```
  from r s have r^2 ≤ s^2 by arith
  with two0 have r ≤ s by (rule-tac n=2 in nat-power-le-imp-le-base)
  with m have m2: s = r + m by simp
  have even m
  proof (rule ccontr)
    assume odd m with rs-odd and m2 show False by auto
  qed
  hence 2 dvd m by (simp only: even-eq-two-dvd)
  then obtain q where m = 2*q by (auto simp add: dvd-def)
  with m2 have q: s = r + 2*q by simp
  obtain p where p: p = r+q by simp
  have c: c = p^2 + q^2
  proof −
    from cb1 and r and s have 2*c = r^2 + s^2 by simp
    also with q have ... = 2*r^2+(2*q)^2+2*r*(2*q)
      by (simp add: nat-power2-add)
    also have ... = 2*r^2+2^2*q^2+2*2*q*r by (simp add: power-mult-distrib)
    also have ... = 2*(r^2+2*q*r+q^2)+2*q^2 by (simp add: power2-eq-square)
    also with p have ... = 2*p^2+2*q^2 by (simp add: nat-power2-add)
    finally show ?thesis by auto
  qed
  moreover have b: b = 2*p*q
  proof −
    from cb2 and r and s have 2*b = s^2 − r^2 by arith
    also with q have ... = (2*q)^2 + 2*r*(2*q) by (simp add: nat-power2-add)
    also with p have ... = 4*q*p by (simp add: power2-eq-square add-mult-distrib2)
    finally show ?thesis by auto
  qed
  moreover have a: a = p^2 − q^2
  proof −
    from p have p≥q by simp
    hence p2-ge-q2: p^2 ≥ q^2 by (simp only: power-mono)
    from a2cb and b and c have a^2 = (p^2 + q^2)^2 − (2*p*q)^2 by simp
    also have ... = (p^2)^2 + (q^2)^2 − 2*(p^2)*(q^2)
      by (auto simp add: nat-power2-add power-mult-distrib mult-ac)
    also with p2-ge-q2 have ... = (p^2 − q^2)^2 by (simp only: nat-power2-diff)
    finally have a^2 = (p^2 − q^2)^2 by simp
    with two0 show ?thesis by (rule-tac n=2 in nat-power-inject-base)
  qed
  moreover have gcd(p,q)=1
  proof −
    let ?k = gcd(p,q)
    have ?k dvd p ∧ ?k dvd q by (simp add: gcd-dvd1 gcd-dvd2)
    with b and a have ?k dvd a ∧ ?k dvd b
      by (simp add: dvd-mult power2-eq-square dvd-diff)
    hence ?k dvd gcd(a,b) by (simp only: gcd-greatest)
    with ab-relprime show ?thesis by auto
  qed
  ultimately show ?thesis by auto
qed
```

Now for the case of integers. Based on *nat-euclid-pyth-triples*.

corollary *int-euclid-pyth-triples*: ⟦ *zgcd(a,b) = 1*; *a ∈ zOdd*; *a^2 + b^2 = c^2* ⟧
⟹ ∃ *p q*. *a = p^2 − q^2 ∧ b = 2*p*q ∧ |c| = p^2 + q^2 ∧ zgcd(p,q)=1*
proof −
assume *ab-rel*: *zgcd(a,b) = 1* **and** *aodd*: *a ∈ zOdd* **and** *abc*: *a^2 + b^2 = c^2*
let *?a = nat|a|*
let *?b = nat|b|*
let *?c = nat|c|*
have *ab2-pos*: *a^2 ≥ 0 ∧ b^2 ≥ 0* **by** (*simp add*: *zero-le-power2*)
hence *nat(a^2) + nat(b^2) = nat(a^2 + b^2)* **by** (*simp only*: *nat-add-distrib*)
with *abc* **have** *nat(a^2) + nat(b^2) = nat(c^2)*
by (*auto simp add*: *power2-eq-square abs-power2-distrib*)
hence *nat(|a|^2) + nat(|b|^2) = nat(|c|^2)*
by (*simp add*: *abs-power2-distrib*)
hence *new-abc*: *?a^2 + ?b^2 = ?c^2*
by (*simp add*: *nat-mult-distrib power2-eq-square nat-add-distrib*)
moreover from *ab-rel* **have** *new-ab-rel*: *gcd(?a,?b)=1* **by** (*simp add*: *zgcd-def*)
moreover have *new-a-odd*: *odd ?a*
proof −
from *aodd* **obtain** *k* **where** *k*: *a = 2*k+1* **by** (*unfold zOdd-def*, *auto*)
show *?thesis*
proof (*cases*)
assume *apos*: *a ≥ 0* **with** *k* **have** *k≥0* **by** *auto*
with *k* **and** *apos* **have** *?a = 2*(nat k)+1* **by** *arith*
thus *?thesis* **by** *simp*
next
assume ¬ *a≥0* **hence** *aneg*: *a<0* **by** *simp*
with *k* **have** *k2*: *2*(−1−k)≥ 0* **by** *simp*
have *aux2*: *(2::int) ≥ 0* **by** *simp*
have *aux1*: *(1::int) ≥ 0* **by** *simp*
from *k* **and** *aneg* **have** *|a| = 2*(−1−k) + 1* **by** *simp*
with *k2 aux1* **have** *?a = nat (2*(−1−k)) + nat 1*
by (*simp only*: *nat-add-distrib*)
with *aux2* **have** *?a = (nat 2)*nat(−1−k) + nat 1*
by (*simp only*: *nat-mult-distrib*)
thus *?thesis* **by** *simp*
qed
qed
ultimately have
∃ *p q*. *?a = p^2−q^2 ∧ ?b = 2*p*q ∧ ?c = p^2 + q^2 ∧ gcd(p,q) =1*
by (*rule-tac a=?a* **and** *b = ?b* **and** *c=?c* **in** *nat-euclid-pyth-triples*)
then obtain *m* **and** *n* **where** *mn*:
*?a = m^2−n^2 ∧ ?b = 2*m*n ∧ ?c = m^2 + n^2 ∧ gcd(m,n) =1* **by** *auto*
have *n^2 ≤ m^2*
proof (*rule ccontr*)
assume ¬ *n^2 ≤ m^2* **hence** *n^2 > m^2* **by** *simp*
with *mn* **have** *?a = 0* **by** *simp*
with *new-a-odd* **show** *False* **by** *simp*
qed
moreover from *mn* **have** *int ?a = int(m^2 − n^2)* **and** *int ?b = int(2*m*n)*
and *int ?c = int(m^2 + n^2)* **by** *auto*
ultimately have *|a| = int(m^2) − int(n^2)* **and** *|b| = int(2*m*n)*
and *|c| = int(m^2) + int(n^2)* **by** (*auto simp only*: *int-nat-abs-eq-abs zdiff-int*)

```
  hence absabc: |a| = (int m) ^2 - (int n) ^2 ∧ |b| = 2*(int m)*int n
    ∧ |c| = (int m) ^2 + (int n) ^2 by (simp add: power2-eq-square int-mult)
  from mn have mn-rel: zgcd(int m,int n)=1 by (simp add: zgcd-def)
  show ∃ p q. a = p ^2 - q ^2 ∧ b = 2*p*q ∧ |c| = p ^2 + q ^2 ∧ zgcd(p,q)=1
    (is ∃ p q. ?Q p q)
  proof (cases)
    assume apos: a ≥ 0 then obtain p where p: p = int m by simp
    hence ∃ q. ?Q p q
    proof (cases)
      assume bpos: b ≥ 0 then obtain q where q = int n by simp
      with p apos bpos absabc mn-rel have ?Q p q by simp
      thus ?thesis by (rule exI)
    next
      assume ¬ b≥0 hence bneg: b<0 by simp
      then obtain q where q = - int n by simp
      with p apos bneg absabc mn-rel have ?Q p q by simp
      thus ?thesis by (rule exI)
    qed
    thus ?thesis by (simp only: exI)
  next
    assume ¬ a≥0 hence aneg: a<0 by simp
    then obtain p where p: p = int n by simp
    hence ∃ q. ?Q p q
    proof (cases)
      assume bpos: b ≥ 0 then obtain q where q = int m by simp
      with p aneg bpos absabc mn-rel have ?Q p q
        by (simp add: zgcd-commute)
      thus ?thesis by (rule exI)
    next
      assume ¬ b≥0 hence bneg: b<0 by simp
      then obtain q where q = - int m by simp
      with p aneg bneg absabc mn-rel have ?Q p q
        by (simp add: zgcd-commute mult-ac)
      thus ?thesis by (rule exI)
    qed
    thus ?thesis by (simp only: exI)
  qed
qed
```

3.2 Fermat's last theorem, case $n = 4$

Core of the proof. Constructs a smaller solution over $\mathbb{Z}$ of

$$a^4 + b^4 = c^2 \wedge \gcd(a, b) = 1 \wedge abc \neq 0 \wedge a \text{ odd}.$$

```
lemma smaller-fermat4:
  assumes abc: a^4+b^4=c^2 and abc0: a*b*c ≠ 0 and aodd: a ∈ zOdd
    and ab-relprime: zgcd(a,b)=1
  shows
  ∃ p q r. (p^4+q^4=r^2 ∧ p*q*r ≠ 0 ∧ p ∈ zOdd ∧ zgcd(p,q) = 1 ∧ r^2 < c^2)
proof -
```

— put equation in shape of a pythagorean triple and obtain u and v
from *ab-relprime* **have** *a2b2relprime*: *zgcd(a^2,b^2) = 1*
by (*simp only*: *zgcd-1-power-distrib*)
moreover from *aodd* **have** *a^2 ∈ zOdd* **by** (*simp only*: *power-preserves-odd*)
moreover from *abc* **have** *(a^2)^2 + (b^2)^2 = c^2* **by** (*simp only*: *quartic-square-square*)
ultimately obtain *u* **and** *v* **where** *uvabc*:
a^2 = u^2−v^2 ∧ b^2 = 2∗u∗v ∧ |c| = u^2 + v^2 ∧ zgcd(u,v)=1
by (*frule-tac a=a^2* **in** *int-euclid-pyth-triples*, *auto*)
with *abc0* **have** *uv0*: *u≠0 ∧ v≠0* **by** *auto*
have *av-relprime*: *zgcd(a,v) = 1*
proof −
from *uvabc* **have** *zgcd(v,a^2) dvd zgcd(b^2,a^2)* **by** (*simp only*: *zgcd-zdvd-zgcd-zmult*)
with *a2b2relprime* **have** *zgcd(a^2,v) dvd (1::int)* **by** (*simp only*: *zgcd-commute*)
moreover have *zgcd(a,v) dvd zgcd(a^2,v)*
by (*simp only*: *zgcd-zdvd-zgcd-zmult power2-eq-square*)
ultimately have *zgcd(a,v) dvd 1* **by** (*rule-tac m=zgcd(a,v)* **in** *zdvd-trans*)
hence *|zgcd(a,v)|= 1* **by** *auto*
thus *?thesis* **by** (*simp add*: *zgcd-geq-zero*)
qed
— make again a pythagorean triple and obtain k and l
from *uvabc* **have** *a^2 + v^2 = u^2* **by** *simp*
with *av-relprime* **and** *aodd* **obtain** *k l* **where**
klavu: *a = k^2−l^2 ∧ v = 2∗k∗l ∧ |u| = k^2+l^2* **and** *kl-rel*: *zgcd(k,l)=1*
by (*frule-tac a=a* **in** *int-euclid-pyth-triples*, *auto*)
— prove $b = 2m$ and $kl(k^2 + l^2) = m^2$, for coprime k, l and $k^2 + l^2$
from *uvabc* **have** *b^2 ∈ zEven* **by** (*unfold zEven-def*, *auto*)
hence *b ∈ zEven* **by** (*simp only*: *power-preserves-even*)
then obtain *m* **where** *bm*: *b = 2∗m* **by** (*auto simp only*: *zEven-def*)
have *|k|∗|l|∗|k^2+l^2| = m^2*
proof −
from *bm* **have** *4∗m^2 = b^2* **by** (*simp only*: *power2-eq-square mult-ac*)
also have *... = |b^2|* **by** *simp*
also with *uvabc* **have** *... = 2∗|v|∗|u|* **by** (*simp add*: *abs-mult*)
also with *klavu* **have** *... = 2∗|2∗k∗l|∗|k^2+l^2|* **by** *simp*
also have *... = 4∗|k|∗|l|∗|k^2+l^2|* **by** (*auto simp add*: *abs-mult*)
finally show *?thesis* **by** *simp*
qed
moreover have *(2::nat) > 1* **by** *auto*
moreover from *kl-rel* **have** *zgcd(|k|,|l|)=1* **by** (*unfold zgcd-def*, *auto*)
moreover have *zgcd(|l|,|k^2+l^2|)=1*
proof −
from *kl-rel* **have** *zgcd(k∗k,l)=1* **by** (*simp only*: *zgcd-zgcd-zmult*)
hence *zgcd(k∗k+l∗l,l)=1* **by** *simp*
hence *zgcd(l,k^2+l^2)=1* **by** (*simp only*: *power2-eq-square zgcd-commute*)
thus *?thesis* **by** (*unfold zgcd-def*, *auto*)
qed
moreover have *zgcd(|k^2+l^2|,|k|)=1*
proof −
from *kl-rel* **have** *zgcd(l,k)=1* **by** (*simp only*: *zgcd-commute*)
hence *zgcd(l∗l,k)=1* **by** (*simp only*: *zgcd-zgcd-zmult*)
hence *zgcd(l∗l+k∗k,k)=1* **by** *simp*
hence *zgcd(k^2+l^2,k)=1* **by** (*simp only*: *add-ac power2-eq-square*)

```
  thus ?thesis by (unfold zgcd-def, auto)
qed
ultimately have
  ∃ x y z. ||k|| = x^2 ∧ ||l|| = y^2 ∧ ||k^2+l^2|| = z^2
  by (rule int-triple-relprime-power-divisors)
then obtain α β γ where albega:
  |k| = α^2 ∧ |l| = β^2 ∧ |k^2+l^2| = γ^2
  by auto
— show this is a new solution
have k^2 = α^4
proof −
  from albega have |k|^2 = (α^2)^2 by simp
  thus ?thesis by (simp add: quartic-square-square abs-power2-distrib)
qed
moreover have l^2 = β^4
proof −
  from albega have |l|^2 = (β^2)^2 by simp
  thus ?thesis by (simp add: quartic-square-square abs-power2-distrib)
qed
moreover have gamma2: k^2 + l^2 = γ^2
proof −
  have k^2 ≥ 0 ∧ l^2 ≥ 0 by (simp add: zero-le-power2)
  with albega show ?thesis by auto
qed
ultimately have newabc: α^4 + β^4 = γ^2 by auto
from uv0 klavu albega have albega0: α * β * γ ≠ 0 by auto
— show the coprimality
have alphabeta-relprime: zgcd(α,β) = 1
proof (rule classical)
  let ?g = zgcd(α,β)
  assume gnot1: ?g ≠ 1
  have ?g > 1
  proof −
    have ?g ≠ 0
    proof
      assume ?g=0
      hence nat |α|=0 by (unfold zgcd-def, auto simp add: gcd-zero)
      hence α=0 by arith
      with albega0 show False by simp
    qed
    hence ?g>0 by (auto simp only: zgcd-geq-zero less-int-def)
    with gnot1 show ?thesis by simp
  qed
  moreover have ?g dvd zgcd(k,l)
  proof −
    have ?g dvd α ∧ ?g dvd β by auto
    with albega have ?g dvd |k| ∧ ?g dvd |l|
      by (simp add: zdvd-zmult power2-eq-square zmult-commute)
    hence ?g dvd k ∧ ?g dvd l by (simp add: zdvd-abs2)
    thus ?thesis by (simp add: zgcd-greatest-iff)
  qed
  ultimately have zgcd(k,l) ≠ 1 by auto
```

```
    with kl-rel show ?thesis by auto
  qed
  — choose p and q in the right way
  have ∃ p q. p^4 + q^4 = γ^2 ∧ p*q*γ ≠ 0 ∧ p ∈ zOdd ∧ zgcd(p,q)=1
  proof −
    have α ∈ zOdd ∨ β ∈ zOdd
    proof (rule ccontr)
      assume ¬ (α ∈ zOdd ∨ β ∈ zOdd)
      hence α ∈ zEven ∧ β ∈ zEven by (auto simp add: not-odd-impl-even)
      then have 2 dvd α ∧ 2 dvd β by (auto simp add: zEven-def)
      then have 2 dvd zgcd(α,β) by (simp add: zgcd-greatest-iff)
      with alphabeta-relprime show False by auto
    qed
    moreover
    { assume α ∈ zOdd
      with newabc albega0 alphabeta-relprime obtain p q where
        p=α ∧ q=β ∧ p^4 + q^4 = γ^2 ∧ p*q*γ ≠ 0 ∧ p ∈ zOdd ∧ zgcd(p,q)=1
        by auto
      hence ?thesis by auto }
    moreover
    { assume β ∈ zOdd
      with newabc albega0 alphabeta-relprime obtain p q where
        q=α ∧ p=β ∧ p^4 + q^4 = γ^2 ∧ p*q*γ ≠ 0 ∧ p ∈ zOdd ∧ zgcd(p,q)=1
        by (auto simp add: add-ac zgcd-commute)
      hence ?thesis by auto }
    ultimately show ?thesis by auto
  qed
  — show the solution is smaller
  moreover have γ^2 < c^2
  proof −
    from gamma2 klavu have γ^2 ≤ |u| by simp
    also have ... ≤ |u|^2 by (rule power2-ge-self)
    also have ... ≤ u^2 by (simp add: abs-power2-distrib)
    also have ... < u^2 + v^2
    proof −
      from uv0 have v2non0: 0 ≠ v^2
        by (auto simp add: power2-eq-square zero-le-power2)
      have 0 ≤ v^2 by (rule zero-le-power2)
      with v2non0 have 0 < v^2 by (auto simp add: less-int-def)
      thus ?thesis by auto
    qed
    also with uvabc have ... ≤ |c| by auto
    also have ... ≤ |c|^2 by (rule power2-ge-self)
    also have ... ≤ c^2 by (simp add: abs-power2-distrib)
    finally show ?thesis by simp
  qed
  ultimately show ?thesis by auto
qed
```

Show that no solution exists, by infinite descent of c^2.

```
lemma no-rewritten-fermat4:
  fixes c::int
```

```
  shows ¬ (∃ a b. (a^4 + b^4 = c^2 ∧ a*b*c ≠ 0 ∧ a ∈ zOdd ∧ zgcd(a,b)=1))
  (is ?Q c)
proof (rule-tac x=c and V = λc. nat(c^2) in val-infinite-descent)
  fix x
  assume x2zero: nat(x^2)=0
  have x^2 ≥ 0 by (rule zero-le-power2)
  with x2zero have int(nat(x^2)) = 0 by auto
  hence x = 0 by auto
  thus ?Q x by auto
next
  fix x
  assume x2pos: 0<nat(x^2) and ¬ ?Q x
  then obtain a b where a^4 + b^4 = x^2 and a*b*x ≠ 0
    and a ∈ zOdd and zgcd(a,b)=1 by auto
  hence ∃ p q r. (p^4+q^4=r^2 ∧ p*q*r ≠ 0 ∧ p ∈ zOdd
    ∧ zgcd(p,q)=1 ∧ r^2 < x^2) by (rule smaller-fermat4)
  then obtain p q r where pqr: p^4 + q^4 = r^2 ∧ p*q*r ≠ 0 ∧ p ∈ zOdd
    ∧ zgcd(p,q)=1 ∧ r^2 < x^2 by auto
  have r^2 ≥ 0 and x^2 ≥ 0 by (auto simp only: zero-le-power2)
  hence int(nat(r^2)) = r^2 ∧ int(nat(x^2)) = x^2 by auto
  with pqr have int(nat(r^2)) < int(nat(x^2)) by auto
  hence nat(r^2) < nat(x^2) by (simp only: zless-int)
  with pqr show ∃ y. nat(y^2) < nat(x^2) ∧ ¬ ?Q y by auto
qed
```

The theorem. Puts equation in requested shape.

```
theorem fermat4:
  assumes ass: (x::int)^4 + y^4 = z^4
  shows x*y*z=0
proof (rule ccontr)
  let ?g = zgcd(x,y)
  let ?c = (z div ?g)^2
  assume xyz0: x*y*z ≠ 0
  — divide out the g.c.d.
  hence x ≠ 0 ∨ y ≠ 0 by simp
  then obtain a b where ab: x = ?g*a ∧ y = ?g*b ∧ zgcd(a,b)=1
    by (frule-tac a=x in make-zrelprime, auto)
  moreover have abc: a^4 + b^4 = ?c^2 ∧ a*b*?c ≠ 0
  proof −
    have zgab: z^4 = ?g^4 * (a^4+b^4)
    proof −
      from ab ass have z^4 = (?g*a)^4+(?g*b)^4 by simp
      thus ?thesis by (simp only: power-mult-distrib zadd-zmult-distrib2)
    qed
    have cgz: z^2 = ?c * ?g^2
    proof −
      from zgab have ?g^4 dvd z^4 by simp
      hence ?g dvd z by (simp only: zpower-zdvd-mono)
      hence (z div ?g)*?g = z by (simp only: mult-ac zdvd-mult-div-cancel)
      with ab show ?thesis by (auto simp only: power2-eq-square mult-ac)
    qed
    with xyz0 have c0: ?c≠0 by (auto simp add: power2-eq-square)
```

```
    from xyz0 have g0: ?g≠0 by (simp add: zgcd-def gcd-zero)
    have a^4 + b^4 = ?c^2
    proof −
      have ?c^2 * ?g^4 = (a^4+b^4)*?g^4
      proof −
        have ?c^2 * ?g^4 = (?c*?g^2)^2
          by (simp only: quartic-square-square power-mult-distrib)
        also with cgz have ... = (z^2)^2 by simp
        also have ... = z^4 by (rule quartic-square-square)
        also with zgab have ... = ?g^4*(a^4+b^4) by simp
        finally show ?thesis by simp
      qed
      with g0 show ?thesis by auto
    qed
    moreover from ab xyz0 c0 have a*b*?c≠0 by auto
    ultimately show ?thesis by simp
  qed
  — choose the parity right
  have ∃ p q. p^4 + q^4 = ?c^2 ∧ p*q*?c≠0 ∧ p ∈ zOdd ∧ zgcd(p,q)=1
  proof −
    have a ∈ zOdd ∨ b ∈ zOdd
    proof (rule ccontr)
      assume ¬(a ∈ zOdd ∨ b ∈ zOdd)
      hence a ∈ zEven ∧ b ∈ zEven by (auto simp add: not-odd-impl-even)
      hence 2 dvd a ∧ 2 dvd b by (auto simp add: zEven-def)
      hence 2 dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
      with ab show False by auto
    qed
    moreover
    { assume a ∈ zOdd
      then obtain p q where p = a and q = b and p ∈ zOdd by simp
      with ab abc have ?thesis by auto }
    moreover
    { assume b ∈ zOdd
      then obtain p q where p = b and q = a and p ∈ zOdd by simp
      with ab abc have
        p^4 + q^4 = ?c^2 ∧ p*q*?c≠0 ∧ p ∈ zOdd ∧ zgcd(p,q)=1
        by (auto simp add: zgcd-commute zmult-commute)
      hence ?thesis by auto }
    ultimately show ?thesis by auto
  qed
  — show contradiction using the earlier result
  thus False by (auto simp only: no-rewritten-fermat4)
qed

corollary fermat-mult4:
  assumes xyz: (x::int)^n + y^n = z^n and n: 4 dvd n
  shows x*y*z=0
proof −
  from n obtain m where n = m*4 by (auto simp only: mult-ac dvd-def)
  with xyz have (x^m)^4 + (y^m)^4 = (z^m)^4 by (simp only: power-mult)
  hence (x^m)*(y^m)*(z^m) = 0 by (rule fermat4)
```

thus *?thesis* **by** *auto*
qed

end

4 The quadratic form $x^2 + Ny^2$

theory *QuadForm*
imports
~~/src/HOL/NumberTheory/Quadratic-Reciprocity
IntNatAux InfDesc
begin

Shows some properties of the quadratic form $x^2 + Ny^2$, such as how to multiply and divide them. The second part focuses on the case $N = 3$ and is used in the proof of the case $n = 3$ of Fermat's last theorem. The last part – not used for FLT3 – shows which primes can be written as $x^2 + 3y^2$.

4.1 Definitions and auxiliary results

constdefs
is-qfN :: *int* ⇒ *int* ⇒ *bool*
is-qfN A N == ∃ *x y*. *A* = *x*^*2* + *N*∗*y*^*2*

is-cube-form :: *int* ⇒ *int* ⇒ *bool*
is-cube-form a b == ∃ *p q*. *a* = *p*^*3* − *9*∗*p*∗*q*^*2* ∧ *b* = *3*∗*p*^*2*∗*q* − *3*∗*q*^*3*

lemma *abs-eq-impl-unitfactor*: |*a*::*int*| = |*b*| ⟹ ∃ *u*. *a* = *u*∗*b* ∧ |*u*|=*1*
proof −
assume |*a*| = |*b*|
hence *a* = *1*∗*b* ∨ *a* = (−*1*)∗*b* **by** *arith*
then obtain *u* **where** *a* = *u*∗*b* ∧ (*u*=*1* ∨ *u*=−*1*) **by** *blast*
thus *?thesis* **by** *auto*
qed

lemma *zprime-3*: *zprime 3*
proof (*auto simp add*: *zprime-def*)
fix *m*::*int* **assume** *m0*: *m* ≥ *0* **and** *mdvd3*: *m dvd 3* **and** *mn3*: *m* ≠ *3*
hence *m* ≤ *3* **by** (*auto simp only*: *zdvd-imp-le*)
with *mn3* **have** *m* < *3* **by** *simp*
moreover from *mdvd3* **have** *m* ≠ *0* **by** *auto*
moreover with *m0* **have** *m* > *0* **by** *simp*
ultimately have *m* = *1* ∨ *m* = *2* **by** *auto*
moreover from *mdvd3* **have** *m* = *2* ⟹ *False* **by** *arith*
ultimately show *m* = *1* **by** *auto*
qed

4.2 Basic facts if $N \geq 1$

lemma *qfN-pos*: ⟦ *N* ≥ *1*; *is-qfN A N* ⟧ ⟹ *A* ≥ *0*
proof −

```
  assume N: N ≥ 1 and is-qfN A N
  then obtain a b where ab: A = a^2 + N*b^2 by (auto simp add: is-qfN-def)
  have N*b^2 ≥ 0
  proof (cases)
    assume b = 0 thus ?thesis by auto
  next
    assume ¬ b = 0 hence  b^2 > 0 by (simp add: zero-less-power2)
    moreover from N have N>0 by simp
    ultimately have N*b^2 > N*0 by (auto simp only: zmult-zless-mono2)
    thus ?thesis by auto
  qed
  with ab have A ≥ a^2 by auto
  moreover have a^2 ≥ 0 by (rule zero-le-power2)
  ultimately show ?thesis by arith
qed

lemma qfN-zero: ⟦ (N::int) ≥ 1; a^2 + N*b^2 = 0 ⟧ ⟹ (a = 0 ∧ b = 0)
proof −
  assume N: N ≥ 1 and abN: a^2 + N*b^2 = 0
  show ?thesis
  proof (rule ccontr, auto)
    assume a ≠ 0 hence a^2 > 0 by (simp add: zero-less-power2)
    moreover have N*b^2 ≥ 0
    proof (cases)
      assume b = 0 thus ?thesis by auto
    next
      assume ¬ b = 0 hence  b^2 > 0 by (simp add: zero-less-power2)
      moreover from N have N>0 by simp
      ultimately have N*b^2 > N*0 by (auto simp only: zmult-zless-mono2)
      thus ?thesis by auto
    qed
    ultimately have a^2 + N*b^2 > 0 by arith
    with abN show False by auto
  next
    assume b ≠ 0 hence b^2>0 by (simp add: zero-less-power2)
    moreover from N have N>0 by simp
    ultimately have N*b^2>N*0 by (auto simp only: zmult-zless-mono2)
    hence N*b^2 > 0 by simp
    moreover have a^2 ≥ 0 by (rule zero-le-power2)
    ultimately have a^2 + N*b^2 > 0 by arith
    with abN show False by auto
  qed
qed
```

4.3 Multiplication and division

```
lemma qfN-mult1: ((a::int)^2 + N*b^2)*(c^2 + N*d^2)
  = (a*c+N*b*d)^2 + N*(a*d−b*c)^2
  by (simp add: nat-number ring-simps)

lemma qfN-mult2: ((a::int)^2 + N*b^2)*(c^2 + N*d^2)
  = (a*c−N*b*d)^2 + N*(a*d+b*c)^2
```

```
  by (simp add: nat-number ring-simps)

corollary is-qfN-mult: is-qfN A N ⟹ is-qfN B N ⟹ is-qfN (A*B) N
  by (unfold is-qfN-def, auto, auto simp only: qfN-mult1)

corollary is-qfN-power: (n::nat) > 0 ⟹ is-qfN A N ⟹ is-qfN (A^n) N
  by (induct n, auto, case-tac n=0, auto simp add: is-qfN-mult)

lemma qfN-div-prime:
  assumes ass: zprime (p^2+N*q^2) ∧ (p^2+N*q^2) dvd (a^2+N*b^2)
  shows ∃ u v. a^2+N*b^2 = (u^2+N*v^2)*(p^2+N*q^2)
              ∧ (∃ e. a = p*u+e*N*q*v ∧ b = p*v - e*q*u ∧ |e|=1)
proof -
  let ?P = p^2+N*q^2
  let ?A = a^2+N*b^2
  from ass obtain U where U: ?A = ?P*U by (auto simp only: dvd-def)
  have ∃ e. ?P dvd b*p + e*a*q ∧ |e| = 1
  proof -
    have ?P dvd (b*p + a*q)*(b*p - a*q)
    proof -
      have (b*p + a*q)*(b*p - a*q)= b^2*?P - q^2*?A
        by (simp add: nat-number ring-simps)
      also from U have ... = (b^2 - q^2*U)*?P by (simp add: ring-simps)
      finally show ?thesis by simp
    qed
    with ass have ?P dvd (b*p + a*q) ∨ ?P dvd (b*p - a*q)
      by (simp only: zprime-zdvd-zmult-general)
    moreover
    { assume ?P dvd b*p + a*q
      hence ?P dvd b*p + 1*a*q ∧ |1| = (1::int) by simp }
    moreover
    { assume ?P dvd b*p - a*q
      hence ?P dvd b*p + (-1)*a*q ∧ |-1| = (1::int) by simp }
    ultimately show ?thesis by blast
  qed
  then obtain v e where v: b*p + e*a*q = ?P*v and e: |e| = 1
    by (auto simp only: dvd-def)
  have ?P dvd a*p - e*N*b*q
  proof (cases)
    assume e1: e = 1
    from U have U * ?P^2 = ?A * ?P by (simp add: power2-eq-square)
    also with e1 have ... = (a*p-e*N*b*q)^2 + N*(b*p+e*a*q)^2
      by (simp only: qfN-mult2 add-commute zmult-1)
    also with v have ... = (a*p-e*N*b*q)^2 + N*v^2*?P^2
      by (simp only: power-mult-distrib mult-ac)
    finally have (a*p-e*N*b*q)^2 = ?P^2*(U-N*v^2)
      by (simp add: mult-ac zdiff-zmult-distrib)
    hence ?P^2 dvd (a*p - e*N*b*q)^2 by (rule dvdI)
    thus ?thesis by (simp only: zpower-zdvd-mono)
  next
    assume ¬ e=1 with e have e1: e=-1 by auto
    from U have U * ?P^2 = ?A * ?P by (simp add: power2-eq-square)
```

```
    also with e1 have ... = (a*p-e*N*b*q) ^2 + N*( -(b*p+e*a*q)) ^2
      by (simp add: qfN-mult1)
    also have ... = (a*p-e*N*b*q) ^2 + N*(b*p+e*a*q) ^2
      by (simp only: power2-minus)
    also with v have ... = (a*p-e*N*b*q) ^2 + N*v^2*?P^2
      by (simp only: power-mult-distrib mult-ac)
    finally have (a*p-e*N*b*q) ^2 = ?P^2*(U-N*v^2)
      by (simp add: mult-ac zdiff-zmult-distrib)
    hence ?P^2 dvd (a*p-e*N*b*q) ^2 by (rule dvdI)
    thus ?thesis by (simp only: zpower-zdvd-mono)
  qed
  then obtain u where u: a*p - e*N*b*q = ?P*u by (auto simp only: dvd-def)
  from e have e2-1: e*e = 1 by auto
  have a: a = p*u + e*N*q*v
  proof -
    have (p*u + e*N*q*v)*?P = p*(?P*u) + (e*N*q)*(?P*v)
      by (simp only: zadd-zmult-distrib mult-ac)
    also with v u have ... = p*(a*p - e*N*b*q) + (e*N*q)*(b*p + e*a*q)
      by simp
    also have ... = a*(p^2 + e*e*N*q^2)
      by (simp add: power2-eq-square zadd-zmult-distrib2 mult-ac zdiff-zmult-distrib2)
    also with e2-1 have ... = a*?P by simp
    finally have (a-(p*u+e*N*q*v))*?P = 0 by auto
    moreover from ass have ?P ≠ 0 by (unfold zprime-def, auto)
    ultimately show ?thesis by simp
  qed
  moreover have b: b = p*v-e*q*u
  proof -
    have (p*v-e*q*u)*?P = p*(?P*v) - (e*q)*(?P*u)
      by (simp only: zdiff-zmult-distrib mult-ac)
    also with v u have ... = p*(b*p+e*a*q) - e*q*(a*p-e*N*b*q) by simp
    also have ... = b*(p^2 + e*e*N*q^2)
      by (simp add: power2-eq-square zadd-zmult-distrib2 mult-ac zdiff-zmult-distrib2)
    also with e2-1 have ... = b * ?P by simp
    finally have (b-(p*v-e*q*u))*?P = 0 by auto
    moreover from ass have ?P ≠ 0 by (unfold zprime-def, auto)
    ultimately show ?thesis by simp
  qed
  moreover have ?A = (u^2 + N*v^2)*?P
  proof (cases)
    assume e=1
    with a and b show ?thesis by (simp add: qfN-mult1 zmult-1 mult-ac)
  next
    assume ¬ e=1 with e have e=-1 by simp
    with a and b show ?thesis by (simp add: qfN-mult2 zmult-1 mult-ac)
  qed
  moreover from e have |e| = 1 ..
  ultimately show ?thesis by blast
qed

corollary qfN-div-prime-weak:
  ⟦ zprime (p^2+N*q^2); (p^2+N*q^2) dvd (a^2+N*b^2) ⟧
```

```
  ⟹ ∃ u v. a^2+N*b^2 = (u^2+N*v^2)*(p^2+N*q^2)
  apply (subgoal-tac ∃ u v. a^2+N*b^2 = (u^2+N*v^2)*(p^2+N*q^2)
    ∧ (∃ e. a = p*u+e*N*q*v ∧ b = p*v − e*q*u ∧ |e|=1), blast)
  apply (rule qfN-div-prime, auto)
done

corollary qfN-div-prime-general: ⟦ zprime P; P dvd A; is-qfN A N; is-qfN P N ⟧
  ⟹ ∃ Q. A = Q*P ∧ is-qfN Q N
  apply (subgoal-tac ∃ u v. A = (u^2+N*v^2)*P)
  apply (unfold is-qfN-def, auto)
  apply (simp only: qfN-div-prime-weak)
done

lemma qfN-power-div-prime:
  assumes ass: zprime P ∧ P ∈ zOdd ∧ P dvd A ∧ P^n = p^2+N*q^2
  ∧ A^n = a^2+N*b^2 ∧ zgcd(a,b)=1 ∧ zgcd(p,N*q)=1 ∧ n>0
  shows ∃ u v. a^2+N*b^2 = (u^2 + N*v^2)*(p^2+N*q^2) ∧ zgcd(u,v)=1
              ∧ (∃ e. a = p*u+e*N*q*v ∧ b = p*v−e*q*u ∧ |e| = 1)
proof −
  from ass have P dvd A ∧ n>0 by simp
  hence P^n dvd A^n by (simp add: zpower-zdvd-mono)
  then obtain U where U: A^n = U*P^n by (auto simp only: dvd-def mult-ac)
  have ∃ e. P^n dvd b*p + e*a*q ∧ |e| = 1
  proof −
    have Pn-dvd-prod: P^n dvd (b*p + a*q)*(b*p − a*q)
    proof −
      have (b*p + a*q)*(b*p − a*q) = (b*p)^2 − (a*q)^2 by (rule zspecial-product)
      also have ... = b^2 * p^2 + b^2*N*q^2 − b^2*N*q^2 − a^2*q^2
        by (simp add: power-mult-distrib)
      also with ass have ... = b^2*P^n − q^2*A^n
        by (simp only: mult-ac zadd-zmult-distrib zadd-zmult-distrib2)
      also with U have ... = (b^2−q^2*U)*P^n by (simp only: zdiff-zmult-distrib)
      finally show ?thesis by (simp add: mult-ac)
    qed
    have P^n dvd (b*p + a*q) ∨ P^n dvd (b*p − a*q)
    proof −
      have PdvdPn: P dvd P^n
      proof −
        from ass have ∃ m. n = Suc m by (simp add: not0-implies-Suc)
        then obtain m where n = Suc m by auto
        hence P^n = P*(P^m) by auto
        thus ?thesis by auto
      qed
      have ¬ P dvd b*p+a*q ∨ ¬ P dvd b*p−a*q
      proof (rule ccontr, simp)
        assume P dvd b*p+a*q ∧ P dvd b*p−a*q
        hence P dvd (b*p+a*q)+(b*p−a*q) ∧ P dvd (b*p+a*q)−(b*p−a*q)
          by (simp only: zdvd-zadd, simp only: zdvd-zdiff)
        hence P dvd 2*(b*p) ∧ P dvd 2*(a*q) by (simp only: mult-2, auto)
        with ass have (P dvd 2 ∨ P dvd b*p) ∧ (P dvd 2 ∨ P dvd a*q)
          by (simp add: zprime-zdvd-zmult-general)
        hence P dvd 2 ∨ (P dvd b*p ∧ P dvd a*q) by auto
```

```
    moreover have ¬ P dvd 2
    proof (rule ccontr, simp)
      assume pdvd2: P dvd 2
      have P ≤ 2
      proof (rule ccontr)
        assume ¬ P ≤ 2 hence Pl2: P > 2 by simp
        with pdvd2 show False by (simp add: zdvd-not-zless)
      qed
      moreover from ass have P > 1 by (simp only: zprime-def)
      ultimately have P=2 by auto
      with ass have 2 ∈ zOdd by simp
      moreover have 2 ∈ zEven by (simp add: zEven-def)
      ultimately show False by (simp add: odd-iff-not-even)
    qed
    ultimately have P dvd b*p ∧ P dvd a*q by auto
    with ass have (P dvd b ∨ P dvd p) ∧ (P dvd a ∨ P dvd q)
      by (auto simp only: zprime-zdvd-zmult-general)
    moreover have ¬ P dvd p ∧ ¬ P dvd q
    proof (auto dest: ccontr)
      assume Pdvdp: P dvd p
      hence P dvd p^2 by (simp only: zdvd-zmult power2-eq-square)
      with PdvdPn have P dvd P^n-p^2 by (simp only: zdvd-zdiff)
      with ass have P dvd N*(q*q) by (simp add: power2-eq-square)
      with ass have P dvd N ∨ P dvd q by (auto dest: zprime-zdvd-zmult-general)
      hence P dvd N*q by (auto simp add: zdvd-zmult zdvd-zmult2)
      with Pdvdp have P dvd zgcd(p,N*q) by (simp add: zgcd-greatest-iff)
      with ass show False by (auto simp add: zprime-def)
    next
      assume P dvd q
      hence PdvdNq: P dvd N*q by (simp add: zdvd-zmult)
      hence P dvd N*q*q by (simp add: zdvd-zmult2)
      hence P dvd N*q^2 by (simp add: power2-eq-square mult-ac)
      with PdvdPn have P dvd P^n-N*q^2 by (simp only: zdvd-zdiff)
      with ass have P dvd p*p by (simp add: power2-eq-square)
      with ass have P dvd p by (auto dest: zprime-zdvd-zmult-general)
      with PdvdNq have P dvd zgcd(p,N*q) by (simp add: zgcd-greatest-iff)
      with ass show False by (auto simp add: zprime-def)
    qed
    ultimately have P dvd a ∧ P dvd b by auto
    hence P dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
    with ass show False by (auto simp add: zprime-def)
  qed
  moreover
  { assume ¬ P dvd b*p+a*q
    with Pn-dvd-prod and ass have P^n dvd b*p-a*q
      by (rule-tac a=b*p+a*q in zprime-power-zdvd-cancel-left, simp) }
  moreover
  { assume ¬ P dvd b*p-a*q
    with Pn-dvd-prod and ass have P^n dvd b*p+a*q
      by (rule-tac a=b*p+a*q in zprime-power-zdvd-cancel-right, simp) }
  ultimately show ?thesis by auto
qed
```

```
  moreover
  { assume P^n dvd b*p + a*q
    hence P^n dvd b*p + 1*a*q ∧ |1| = (1::int) by simp }
  moreover
  { assume P^n dvd b*p − a*q
    hence P^n dvd b*p + (−1)*a*q ∧ |−1| = (1::int) by simp }
  ultimately show ?thesis by blast
qed
then obtain v e where v: b*p + e*a*q = P^n*v and e: |e| = 1
  by (auto simp only: dvd-def)
have P^n dvd a*p − e*N*b*q
proof (cases)
  assume e1: e = 1
  from U have (P^n)^2*U = A^n*P^n by (simp add: power2-eq-square mult-ac)
  also with e1 ass have ... = (a*p−e*N*b*q)^2 + N*(b*p+e*a*q)^2
    by (simp only: qfN-mult2 add-commute zmult-1)
  also with v have ... = (a*p−e*N*b*q)^2 + (P^n)^2*(N*v^2)
    by (simp only: power-mult-distrib mult-ac)
  finally have (a*p−e*N*b*q)^2 = (P^n)^2*U − (P^n)^2*N*v^2 by simp
  also have ... = (P^n)^2 * (U − N*v^2) by (simp only: zdiff-zmult-distrib2)
  finally have (P^n)^2 dvd (a*p − e*N*b*q)^2 by (rule dvdI)
  thus ?thesis by (simp only: zpower-zdvd-mono)
next
  assume ¬ e=1 with e have e1: e=−1 by auto
  from U have (P^n)^2 * U = A^n * P^n by (simp add: power2-eq-square)
  also with e1 ass have ... = (a*p−e*N*b*q)^2 + N*( −(b*p+e*a*q))^2
    by (simp add: qfN-mult1)
  also have ... = (a*p−e*N*b*q)^2 + N*(b*p+e*a*q)^2
    by (simp only: power2-minus)
  also with v and ass have ... = (a*p−e*N*b*q)^2 + N*v^2*(P^n)^2
    by (simp only: power-mult-distrib mult-ac)
  finally have (a*p−e*N*b*q)^2 = (P^n)^2*U−(P^n)^2*N*v^2 by simp
  also have ... = (P^n)^2 * (U − N*v^2) by (simp only: zdiff-zmult-distrib2)
  finally have (P^n)^2 dvd (a*p−e*N*b*q)^2 by (rule dvdI)
  thus ?thesis by (simp only: zpower-zdvd-mono)
qed
then obtain u where u: a*p − e*N*b*q = P^n*u by (auto simp only: dvd-def)
from e have e2-1: e*e = 1 by auto
have a: a = p*u + e*N*q*v
proof −
  from ass have (p*u + e*N*q*v)*P^n = p*(P^n*u) + (e*N*q)*(P^n*v)
    by (simp only: zadd-zmult-distrib mult-ac)
  also with v and u have ... = p*(a*p − e*N*b*q) + (e*N*q)*(b*p + e*a*q)
    by simp
  also have ... = a*(p^2 + e*e*N*q^2)
    by (simp add: power2-eq-square zadd-zmult-distrib2 mult-ac zdiff-zmult-distrib2)
  also with e2-1 and ass have ... = a*P^n by simp
  finally have (a−(p*u+e*N*q*v))*P^n = 0 by auto
  moreover from ass have P^n ≠ 0
    by (unfold zprime-def, auto simp add: power-eq-0-iff)
  ultimately show ?thesis by auto
qed
```

```
  moreover have b: b = p*v-e*q*u
  proof -
    from ass have (p*v-e*q*u)*P^n = p*(P^n*v) - (e*q)*(P^n*u)
      by (simp only: zdiff-zmult-distrib mult-ac)
    also with v u have ... = p*(b*p+e*a*q) - e*q*(a*p-e*N*b*q) by simp
    also have ... = b*(p^2 + e*e*N*q^2)
      by (simp add: power2-eq-square zadd-zmult-distrib2 mult-ac zdiff-zmult-distrib2)
    also with e2-1 and ass have ... = b * P^n by simp
    finally have (b-(p*v-e*q*u))*P^n = 0 by auto
    moreover from ass have P^n ≠ 0
      by (unfold zprime-def, auto simp add: power-eq-0-iff)
    ultimately show ?thesis by auto
  qed
  moreover have A^n = (u^2 + N*v^2)*P^n
  proof (cases)
    assume e=1
    with a and b and ass show ?thesis by (simp add: qfN-mult1 zmult-1 mult-ac)
  next
    assume ¬ e=1 with e have e=-1 by simp
    with a and b and ass show ?thesis by (simp add: qfN-mult2 zmult-1 mult-ac)
  qed
  moreover have zgcd(u,v)=1
  proof -
    let ?g = zgcd(u,v)
    have ?g dvd u ∧ ?g dvd v by auto
    hence ?g dvd u*p + v*(e*N*q) ∧ ?g dvd v*p - u*(e*q)
      by (simp add: zdvd-zmult2 zdvd-zadd zdvd-zdiff)
    with a and b have ?g dvd a ∧ ?g dvd b by (auto simp only: mult-ac)
    hence ?g dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
    with ass have ?g = 1 ∨ ?g = -1 by simp
    moreover have ?g ≥ 0 by (rule zgcd-geq-zero)
    ultimately show ?thesis by auto
  qed
  moreover from e and ass have
    |e| = 1 ∧ A^n = a^2+N*b^2 ∧ P^n = p^2+N*q^2 by simp
  ultimately show ?thesis by auto
qed

lemma qfN-primedivisor-not:
  assumes ass: zprime P ∧ Q > 0 ∧ is-qfN (P*Q) N ∧ ¬ is-qfN P N
  shows ∃ R. (zprime R ∧ R dvd Q ∧ ¬ is-qfN R N)
proof (rule ccontr, auto)
  assume ass2: ∀ R. R dvd Q ⟶ zprime R ⟶ is-qfN R N
  have ∃ ps. primel ps ∧ int (prod ps) = Q
  proof -
    from ass have Q=1 ∨ nat(Q) > Suc 0 by auto
    moreover
    { assume Q=1 hence primel [] ∧ int (prod []) = Q by (simp add: primel-def)
      hence ?thesis by auto }
    moreover
    { assume nat(Q) > Suc 0
      then have ∃ ps. primel ps ∧ prod ps = nat(Q) by (simp only: factor-exists)
```

```
    with ass have ?thesis by auto }
   ultimately show ?thesis by blast
 qed
 then obtain ps where ps: primel ps ∧ int(prod ps) = Q by auto
 have ps-lemma: (primel ps ∧ is-qfN (P*int(prod ps)) N
  ∧ (∀ R. (zprime R ∧ R dvd int(prod ps)) ⟶ is-qfN R N)) ⟹ False
  (is ?B ps ⟹ False)
 proof (induct ps)
  case Nil hence is-qfN P N by simp
  with ass show False by simp
 next
  case (Cons p ps)
  hence ass3: ?B ps ⟹ False
   and IH: ?B (p#ps) by simp
  hence p: zprime (int p) and int p dvd int(prod(p#ps))
   by (auto simp add: primel-def prime-impl-zprime-int int-mult)
  moreover with IH have pqfN: is-qfN (int p) N
   and int p dvd P*int(prod(p#ps)) and is-qfN (P*int(prod(p#ps))) N
   by (auto simp add: zdvd-zmult)
  ultimately obtain S where S: P*int(prod(p#ps)) = S*(int p) ∧ is-qfN S N
   by (auto dest: qfN-div-prime-general)
  hence (int p)*(P* int(prod ps) − S) = 0 by (auto simp add: int-mult)
  with p S have is-qfN (P*int(prod ps)) N by (auto simp add: zprime-def)
  moreover from IH have primel ps by (simp add: primel-def)
  moreover from IH have ∀ R. zprime R ∧ R dvd int(prod ps) ⟶ is-qfN R N
   by (auto simp add: int-mult zdvd-zmult)
  ultimately have ?B ps by simp
  with ass3 show False by simp
 qed
 with ps ass2 ass show False by auto
qed

lemma qfN-oddprime-cube:
 ⟦ zprime (p^2+N*q^2); (p^2+N*q^2) ∈ zOdd; p ≠ 0; N ≥ 1 ⟧
 ⟹ ∃ a b. (p^2+N*q^2)^3 = a^2 + N*b^2 ∧ zgcd(a, N*b)=1
proof −
 let ?P = p^2+N*q^2
 assume P: zprime ?P and Podd: ?P ∈ zOdd and p0: p ≠ 0 and N1: N ≥ 1
 have suc23: 3 = Suc 2 by simp
 let ?a = p*(p^2 − 3*N*q^2)
 let ?b = q*(3*p^2 − N*q^2)
 have abP: ?P^3 = ?a^2 + N*?b^2 by (simp add: nat-number ring-simps)
 have zgcd(?b,?a) ≠ 1 ⟹ ?P dvd p
 proof −
  let ?h = zgcd(?b,?a)
  assume h1: ?h ≠ 1
  have ?h ≥ 0 by (rule zgcd-geq-zero)
  hence ?h = 0 ∨ ?h = 1 ∨ ?h > 1 by auto
  with h1 have ?h =0 ∨ ?h >1 by auto
  moreover
  { assume ?h = 0 hence nat|?b| = 0 ∧ nat|?a| = 0
     by (unfold zgcd-def, auto simp add: gcd-zero)
```

```
  hence ?a = 0 ∧ ?b = 0 by arith
  with abP have ?P^3 = 0 by auto
  with P have False by (unfold zprime-def, auto)
  hence ?thesis by simp }
moreover
{ assume ?h > 1 hence ∃ g. zprime g ∧ g dvd ?h by (rule zprime-factor-exists)
  then obtain g where g: zprime g ∧ g dvd ?h by blast
  hence g dvd ?b ∧ g dvd ?a by (simp add: zgcd-greatest-iff)
  with g have g1: g dvd q ∨ g dvd 3*p^2-N*q^2
    and g2: g dvd p ∨ g dvd p^2 - 3*N*q^2
    by (auto simp add: zprime-zdvd-zmult-general)
  from g have gpos: g ≥ 0 by (auto simp only: zprime-def)
  have g dvd ?P
  proof (cases)
    assume g dvd q
    hence gNq: g dvd N*q^2 by (auto simp add: dvd-def power2-eq-square)
    show ?thesis
    proof (cases)
      assume gp: g dvd p
      hence g dvd p^2 by (auto simp add: dvd-def power2-eq-square)
      with gNq show ?thesis by (auto simp add: zdvd-zadd)
    next
      assume ¬ g dvd p with g2 have g dvd p^2 - 3*N*q^2 by auto
      moreover from gNq have g dvd 4*(N*q^2) by (rule zdvd-zmult)
      ultimately have g dvd p^2 - 3*(N*q^2) + 4*(N*q^2)
        by (simp only: zdvd-zadd mult-ac)
      moreover have p^2 - 3*(N*q^2)+4*(N*q^2) = p^2 + N*q^2 by arith
      ultimately show ?thesis by simp
    qed
  next
    assume ¬ g dvd q with g1 have gpq: g dvd 3*p^2-N*q^2 by simp
    show ?thesis
    proof (cases)
      assume g dvd p
      hence g dvd 4*p^2 by (auto simp add: dvd-def power2-eq-square)
      with gpq have  g dvd 4*p^2 - (3*p^2 - N*q^2) by (simp only: zdvd-zdiff)
      moreover have 4*p^2 - (3*p^2 - N*q^2) = p^2 + N*q^2 by arith
      ultimately show ?thesis by simp
    next
      assume ¬ g dvd p with g2 have g dvd p^2 - 3*N*q^2 by auto
      with gpq have g dvd 3*p^2-N*q^2 - (p^2 - 3*N*q^2)
        by (simp only: zdvd-zdiff)
      moreover have 3*p^2-N*q^2 - (p^2 - 3*N*q^2) = 2*?P by auto
      ultimately have g dvd 2*?P by simp
      with g have g dvd 2 ∨ g dvd ?P by (simp only: zprime-zdvd-zmult)
      moreover have ¬ g dvd 2
      proof (rule ccontr, simp)
        assume gdvd2: g dvd 2
        have g ≤ 2
        proof (rule ccontr)
          assume ¬ g ≤ 2 hence g > 2 by simp
          moreover have (0::int) < 2 by auto
```

```
      ultimately have ¬ g dvd 2 by (auto simp only: zdvd-not-zless)
      with gdvd2 show False by simp
    qed
    moreover from g have g ≥ 2 by (simp add: zprime-def)
    ultimately have g = 2 by auto
    with g have 2 dvd ?a ∧ 2 dvd ?b by (auto simp add: zgcd-greatest-iff)
    hence 2 dvd ?a^2 ∧ 2 dvd N*?b^2
      by (auto simp only: power2-eq-square zdvd-zmult)
    with abP have 2 dvd ?P^3 by (simp only: zdvd-zadd)
    hence ?P^3 ∈ zEven by (auto simp add: dvd-def zEven-def)
    moreover have ?P^3 ∈ zOdd
    proof −
      from Podd have ?P*?P^2 ∈ zOdd
        by (simp only: odd-times-odd power2-eq-square)
      thus ?thesis by (simp only: cube-square)
    qed
    ultimately show False by (auto simp only: odd-iff-not-even)
  qed
  ultimately show ?thesis by simp
 qed
qed
with P gpos have g = 1 ∨ g = ?P by (auto simp only: zprime-def)
with g have g = ?P by (simp add: zprime-def)
with g have Pab: ?P dvd ?a ∧ ?P dvd ?b by (auto simp add: zgcd-greatest-iff)
have ?thesis
proof −
 from Pab P have ?P dvd p ∨ ?P dvd p^2− 3*N*q^2
  by (auto simp add: zprime-zdvd-zmult-general)
 moreover
 { assume ?P dvd p^2 − 3*N*q^2
  moreover have ?P dvd 3*(p^2 + N*q^2)
   by (auto simp only: zdvd-refl zdvd-zmult)
  ultimately have ?P dvd p^2− 3*N*q^2 + 3*(p^2+N*q^2)
   by (simp only: zdvd-zadd)
  hence ?P dvd 4*p^2 by auto
  with P have ?P dvd 4 ∨ ?P dvd p^2
   by (simp only: zprime-zdvd-zmult-general)
  moreover have ¬ ?P dvd 4
  proof (rule ccontr, simp)
   assume Pdvd4: ?P dvd 4
   have ?P ≤ 4
   proof (rule ccontr)
    assume ¬ ?P ≤ 4 hence ?P > 4 by simp
    moreover have (0::int) < 4 by auto
    ultimately have ¬ ?P dvd 4 by (auto simp only: zdvd-not-zless)
    with Pdvd4 show False by simp
   qed
   moreover from P have ?P ≥ 2 by (auto simp add: zprime-def)
   moreover have ?P ≠ 2 ∧ ?P ≠ 4
   proof (rule ccontr, simp)
    assume ?P = 2 ∨ ?P = 4 hence ?P ∈ zEven
     by (auto simp add: zEven-def)
```

```
        with Podd show False by (simp add: odd-iff-not-even)
      qed
      ultimately have ?P = 3 by auto
      with Pdvd4 have (3::int) dvd 4 by simp
      thus False by arith
    qed
    ultimately have ?P dvd p*p by (simp add: power2-eq-square)
    with P have ?thesis by (auto dest: zprime-zdvd-zmult-general) }
  ultimately show ?thesis by auto
 qed }
 ultimately show ?thesis by blast
qed
moreover have zgcd(N,?a) ≠ 1 ⟹ ?P dvd p
proof −
 let ?h = zgcd(N,?a)
 assume h1: ?h ≠ 1
 have ?h ≥ 0 by (rule zgcd-geq-zero)
 hence ?h = 0 ∨ ?h = 1 ∨ ?h > 1 by auto
 with h1 have ?h =0 ∨ ?h >1 by auto
 moreover
 { assume ?h = 0 hence nat|N| = 0 ∧ nat|?a| = 0
    by (unfold zgcd-def, auto simp add: gcd-zero)
  hence N = 0 by arith
  with N1 have False by auto
  hence ?thesis by simp }
 moreover
 { assume ?h > 1 hence ∃ g. zprime g ∧ g dvd ?h by (rule zprime-factor-exists)
  then obtain g where g: zprime g ∧ g dvd ?h by blast
  hence gN: g dvd N and g dvd ?a by (auto simp add: zgcd-greatest-iff)
  hence g dvd p*p^2 − N*(3*p*q^2)
   by (auto simp only: zdiff-zmult-distrib2 mult-ac)
  with gN have g dvd p*p^2 − N*(3*p*q^2) + N*(3*p*q^2)
   by (simp only: zdvd-zadd zdvd-zmult2)
  hence g dvd p*p^2 by simp
  with g have g dvd p ∨ g dvd p*p
   by (simp add: zprime-zdvd-zmult-general power2-eq-square)
  with g have gp: g dvd p by (auto dest: zprime-zdvd-zmult-general)
  hence g dvd p^2 by (simp add: zdvd-zmult power2-eq-square)
  with gN have gP: g dvd ?P by (auto simp add: zdvd-zmult2 zdvd-zadd)
  from g have g ≥ 0 by (simp add: zprime-def)
  with gP P have g = 1 ∨ g = ?P by (auto simp only: zprime-def)
  with g have g = ?P by (auto simp only: zprime-def)
  with gp have ?thesis by simp }
 ultimately show ?thesis by auto
qed
moreover have ¬ ?P dvd p
proof (rule ccontr, clarsimp)
 assume Pdvdp: ?P dvd p
 have p^2 ≥ ?P^2
 proof (rule ccontr)
  assume ¬ p^2 ≥ ?P^2 hence pP: p^2 < ?P^2 by simp
  moreover with p0 have p^2 > 0 by (simp add: zero-less-power2)
```

```
    ultimately have ¬ ?P^2 dvd p^2 by (simp add: zdvd-not-zless)
    with Pdvdp show False by (simp add: zpower-zdvd-mono)
  qed
  moreover with P have ?P*1 < ?P*?P
    by (unfold zprime-def, auto simp only: zmult-zless-mono2)
  ultimately have p^2 > ?P by (auto simp add: power2-eq-square)
  hence neg: N*q^2 < 0 by auto
  show False
  proof −
    have is-qfN (0^2 + N*q^2) N by (auto simp only: is-qfN-def)
    with N1 have 0^2 +N*q^2 ≥ 0 by (rule qfN-pos)
    with neg show False by simp
  qed
 qed
 ultimately have zgcd(?b,?a)=1 ∧ zgcd(N,?a)=1 by auto
 hence zgcd(N*?b,?a) = 1 by (simp only: zgcd-zmult-cancel)
 with abP show ?thesis by (auto simp only: zgcd-commute)
qed
```

4.4 Uniqueness ($N > 1$)

```
lemma qfN-prime-unique:
 ⟦ zprime (a^2+N*b^2); N > 1; a^2+N*b^2 = c^2+N*d^2 ⟧
 ⟹ (|a| = |c| ∧ |b| = |d|)
proof −
 let ?P = a^2+N*b^2
 assume P: zprime ?P and N: N > 1 and abcdN: ?P = c^2 + N*d^2
 have mult: (a*d+b*c)*(a*d−b*c) = ?P*(d^2−b^2)
 proof −
  have (a*d+b*c)*(a*d−b*c) = (a^2 + N*b^2)*d^2 − b^2*(c^2 + N*d^2)
    by (simp add: nat-number ring-simps)
  with abcdN show ?thesis by (simp add: ring-simps)
 qed
 have ?P dvd a*d+b*c ∨ ?P dvd a*d−b*c
 proof −
  from mult have ?P dvd (a*d+b*c)*(a*d−b*c) by simp
  with P show ?thesis by (simp add: zprime-zdvd-zmult-general)
 qed
 moreover
 { assume ?P dvd a*d+b*c
  then obtain Q where Q: a*d+b*c = ?P*Q by (auto simp add: dvd-def)
  from abcdN have ?P^2 = (a^2 + N*b^2) * (c^2 + N*d^2)
    by (simp add: power2-eq-square)
  also have ... = (a*c−N*b*d)^2 + N*(a*d+b*c)^2 by (rule qfN-mult2)
  also with Q have ... = (a*c−N*b*d)^2 + N*Q^2*?P^2
    by (simp add: mult-ac power-mult-distrib)
  also have ... ≥ N*Q^2*?P^2 by (simp add: zero-le-power2)
  finally have pos: ?P^2 ≥ ?P^2*(Q^2*N) by (simp add: mult-ac)
  have b^2 = d^2
  proof (rule ccontr)
    assume b^2 ≠ d^2
    with P mult Q have Q ≠ 0 by (unfold zprime-def, auto)
```

```
    hence Q^2 > 0 by (simp add: zero-less-power2)
    moreover with N have Q^2*N > Q^2*1 by (simp only: zmult-zless-mono2)
    ultimately have Q^2*N > 1 by arith
    moreover with P have ?P^2 > 0 by (simp add: zprime-def zero-less-power2)
    ultimately have ?P^2*1 < ?P^2*(Q^2*N) by (simp only: zmult-zless-mono2)
    with pos show False by simp
   qed }
  moreover
  { assume ?P dvd a*d-b*c
   then obtain Q where Q: a*d-b*c = ?P*Q by (auto simp add: dvd-def)
   from abcdN have ?P^2 = (a^2 + N*b^2) * (c^2 + N*d^2)
    by (simp add: power2-eq-square)
   also have ... = (a*c+N*b*d)^2 + N*(a*d-b*c)^2 by (rule qfN-mult1)
   also with Q have ... = (a*c+N*b*d)^2 + N*Q^2*?P^2
    by (simp add: mult-ac power-mult-distrib)
   also have ... ≥ N*Q^2*?P^2 by (simp add: zero-le-power2)
   finally have pos: ?P^2 ≥ ?P^2*(Q^2*N) by (simp add: mult-ac)
   have b^2 = d^2
   proof (rule ccontr)
    assume b^2 ≠ d^2
    with P mult Q have Q ≠ 0 by (unfold zprime-def, auto)
    hence Q^2 > 0 by (simp add: zero-less-power2)
    moreover with N have Q^2*N > Q^2*1 by (simp only: zmult-zless-mono2)
    ultimately have Q^2*N > 1 by arith
    moreover with P have ?P^2 > 0 by (simp add: zprime-def zero-less-power2)
    ultimately have ?P^2*1 < ?P^2 * (Q^2*N) by (simp only: zmult-zless-mono2)
    with pos show False by simp
   qed }
  ultimately have bd: b^2 = d^2 by blast
  moreover with abcdN have a^2 = c^2 by auto
  ultimately show ?thesis by (auto simp only: power2-eq-iff-abs-eq)
qed

lemma qfN-square-prime:
  assumes ass:
  zprime (p^2+N*q^2) ∧ N>1 ∧ (p^2+N*q^2)^2 = r^2+N*s^2 ∧ zgcd(r,s)=1
  shows |r| = |p^2-N*q^2| ∧ |s| = |2*p*q|
proof -
  let ?P = p^2 + N*q^2
  let ?A = r^2 + N*s^2
  from ass have P1: ?P > 1 by (simp add: zprime-def)
  from ass have APP: ?A = ?P*?P by (simp only: power2-eq-square)
  with ass have zprime ?P ∧ ?P dvd ?A by (simp add: dvdI)
  then obtain u v e where uve:
    ?A = (u^2+N*v^2)*?P ∧ r = p*u+e*N*q*v ∧ s = p*v - e*q*u ∧ |e|=1
    by (frule-tac p=p in qfN-div-prime, auto)
  with APP P1 ass have zprime (u^2+N*v^2) ∧ N>1 ∧ u^2 + N*v^2 = ?P
    by auto
  hence |u| = |p| ∧ |v| = |q| by (auto dest: qfN-prime-unique)
  then obtain f g where f: u = f*p ∧ |f| = 1 and g: v = g*q ∧ |g| = 1
    by (blast dest: abs-eq-impl-unitfactor)
  with uve have r = f*p*p + (e*g)*N*q*q ∧ s = g*p*q - (e*f)*p*q by simp
```

```
  hence rs: r = f*p^2 + (e*g)*N*q^2 ∧ s = (g - e*f)*p*q
    by (auto simp only: power2-eq-square zdiff-zmult-distrib)
  moreover have s ≠ 0
  proof (rule ccontr, simp)
    assume s0: s=0
    hence zgcd(r,s) = |r| by (simp only: zgcd-0)
    with ass have |r| = 1 by simp
    hence r^2 = 1 by (auto simp add: abs-power2-distrib)
    with s0 have ?A = 1 by simp
    moreover have ?P^2 > 1
    proof -
      from P1 have 1 < ?P ∧ (0::int) ≤ 1 ∧ (0::nat) < 2 by auto
      hence ?P^2 > 1^2 by (simp only: power-strict-mono)
      thus ?thesis by auto
    qed
    moreover from ass have ?A = ?P^2 by simp
    ultimately show False by auto
  qed
  ultimately have g ≠ e*f by auto
  moreover from f g uve have |g| = |e*f| by auto
  ultimately have g = -(e*f) by arith
  with rs uve have r = f*(p^2 - N*q^2) ∧ s = (-e*f)*2*p*q
    by (auto simp add: power2-eq-square zdiff-zmult-distrib2)
  hence |r| = |f| * |p^2-N*q^2|
    ∧ |s| = |e|*|f|*|2*p*q|
    by (auto simp add: abs-mult)
  with uve f g show ?thesis by (auto simp only: zmult-1)
qed

lemma qfN-cube-prime:
  assumes ass: zprime (p^2 + N*q^2) ∧ N > 1
  ∧ (p^2 + N*q^2)^3 = a^2 + N*b^2 ∧ zgcd(a, b)=1
  shows |a| = |p^3- 3*N*p*q^2| ∧ |b| = |3*p^2*q-N*q^3|
proof -
  let ?P = p^2 + N*q^2
  let ?A = a^2 + N*b^2
  from ass have P1: ?P > 1 by (simp add: zprime-def)
  with ass have APP: ?A = ?P*?P^2 by (auto simp only: cube-square)
  with ass have zprime ?P ∧ ?P dvd ?A by (simp add: dvdI)
  then obtain u v e where uve:
    ?A = (u^2+N*v^2)*?P ∧ a = p*u+e*N*q*v ∧ b = p*v-e*q*u ∧ |e|=1
    by (frule-tac p=p in qfN-div-prime, auto)
  have zgcd(u,v)=1
  proof -
    let ?g = zgcd(u,v)
    have ?g dvd u ∧ ?g dvd v by (auto simp add: zgcd-greatest-iff)
    with uve have ?g dvd a ∧ ?g dvd b
      by (auto simp add: zdvd-zmult zdvd-zadd zdvd-zdiff)
    hence ?g dvd zgcd(a,b) by (auto simp add: zgcd-greatest-iff)
    with ass have ?g dvd 1 by simp
    moreover have ?g ≥ 0 by (rule zgcd-geq-zero)
    ultimately show ?thesis by auto
```

```
  qed
  with P1 uve APP ass have zprime ?P ∧ N > 1 ∧ ?P^2 = u^2+N*v^2
    ∧ zgcd(u,v)=1 by (auto simp add: mult-ac)
  hence |u| = |p^2−N*q^2| ∧ |v| = |2*p*q| by (rule qfN-square-prime)
  then obtain f g where f: u = f*(p^2−N*q^2) ∧ |f| = 1
    and g: v = g*(2*p*q) ∧ |g| = 1 by (blast dest: abs-eq-impl-unitfactor)
  with uve have a = p*f*(p^2−N*q^2) + e*N*q*g*2*p*q
    ∧ b = p*g*2*p*q −e*q*f*(p^2−N*q^2) by auto
  hence ab: a = f*p*p^2 + −f*N*p*q^2 + 2*e*g*N*p*q^2
    ∧ b = 2*g*p^2*q − e*f*p^2*q + e*f*N*q*q^2
    by (auto simp add: mult-ac zdiff-zmult-distrib2 power2-eq-square)
  from f have f2: f^2 = 1 by (auto simp add: abs-power2-distrib)
  from g have g2: g^2 = 1 by (auto simp add: abs-power2-distrib)
  have e ≠ f*g
  proof (rule ccontr, simp)
    assume efg: e = f*g
    with ab g have a = f*p*p^2+f*N*p*q^2 by (auto simp add: power2-eq-square)
    hence a = (f*p)*?P by (auto simp add: zadd-zmult-distrib2 mult-ac)
    hence Pa: ?P dvd a by auto
    from efg f ab have b = g*p^2*q+g*N*q*q^2 by (auto simp add: power2-eq-square)
    hence b = (g*q)*?P by (auto simp add: zadd-zmult-distrib2 mult-ac)
    hence ?P dvd b by auto
    with Pa have ?P dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
    with ass have ?P dvd 1 by auto
    with P1 show False by auto
  qed
  moreover from f g uve have |e| = |f*g| by auto
  ultimately have e = −(f*g) by arith
  with ab f g have a = f*p*p^2 − 3*f*N*p*q^2 ∧ b = 3*g*p^2*q − g*N*q*q^2
    by (auto simp add: power2-eq-square)
  hence a = f*(p^3 − 3*N*p*q^2) ∧ b = g*( 3*p^2*q − N*q^3 )
    by (auto simp only: zdiff-zmult-distrib2 mult-ac cube-square)
  with f g show ?thesis by (auto simp add: zmult-1 abs-mult)
qed
```

4.5 The case $N = 3$

```
lemma qf3-even: a^2+3*b^2 ∈ zEven ⟹ ∃ B. a^2+3*b^2 = 4*B ∧ is-qfN B 3
proof −
  let ?A = a^2+3*b^2
  assume even: ?A ∈ zEven
  have (a ∈ zOdd ∧ b ∈ zOdd) ∨ (a ∈ zEven ∧ b ∈ zEven)
  proof (rule ccontr, auto dest: not-odd-impl-even)
    assume a ∉ zOdd and b ∉ zEven
    hence a ∈ zEven ∧ b ∈ zOdd by (auto simp only: odd-iff-not-even)
    hence a^2 ∈ zEven ∧ b^2 ∈ zOdd
      by (auto simp only: power2-eq-square odd-times-odd even-times-either)
    moreover have 3 ∈ zOdd by (unfold zOdd-def, auto)
    ultimately have ?A ∈ zOdd by (auto simp add: odd-times-odd even-plus-odd)
    with even show False by (simp add: odd-iff-not-even)
  next
    assume a ∉ zEven and b ∉ zOdd
```

```
    hence a ∈ zOdd ∧ b ∈ zEven by (auto simp only: odd-iff-not-even)
    hence a^2 ∈ zOdd ∧ b^2 ∈ zEven
      by (auto simp only: power2-eq-square odd-times-odd even-times-either)
    moreover hence b^2*3 ∈ zEven by (simp only: even-times-either)
    ultimately have b^2*3+a^2 ∈ zOdd by (auto simp add: even-plus-odd)
    hence ?A ∈ zOdd by (simp only: mult-ac add-ac)
    with even show False by (simp add: odd-iff-not-even)
  qed
  moreover
  { assume a ∈ zEven ∧ b ∈ zEven
    then obtain c d where abcd: a = 2*c ∧ b = 2*d by (unfold zEven-def, auto)
    hence ?A = 4*(c^2 + 3*d^2) by (simp add: power-mult-distrib)
    moreover have is-qfN (c^2+3*d^2) 3 by (unfold is-qfN-def, auto)
    ultimately have ?thesis by blast }
  moreover
  { assume a ∈ zOdd ∧ b ∈ zOdd
    then obtain c d where abcd: a = 2*c+1 ∧ b = 2*d+1
      by (unfold zOdd-def, auto)
    have c−d ∈ zOdd ∨ c−d ∈ zEven by (rule-tac x=c−d in even-odd-disj)
    moreover
    { assume c−d ∈ zEven
      then obtain e where c−d = 2*e by (auto simp add: zEven-def)
      with abcd have e1: a−b = 4*e by arith
      hence e2: a+3*b = 4*(e+b) by auto
      have 4*?A = (a+3*b)^2 + 3*(a−b)^2
        by (simp add: nat-number ring-simps)
      also with e1 e2 have ... = (4*(e+b))^2+3*(4*e)^2 by (simp(no-asm-simp))
      finally have ?A = 4*((e+b)^2 + 3*e^2) by (simp add: nat-number ring-simps)
      moreover have is-qfN ((e+b)^2 + 3*e^2) 3 by (unfold is-qfN-def, auto)
      ultimately have ?thesis by blast }
    moreover
    { assume c−d ∈ zOdd
      then obtain e where c−d = 2*e+1 by (auto simp add: zOdd-def)
      with abcd have e1: a+b = 4*(e+d+1) by auto
      hence e2: a− 3*b = 4*(e+d−b+1) by auto
      have 4*?A = (a− 3*b)^2 + 3*(a+b)^2
        by (simp add: nat-number ring-simps)
      also with e1 e2 have ... = (4*(e+d−b+1))^2 +3*(4*(e+d+1))^2
        by (simp (no-asm-simp))
      finally have ?A = 4*((e+d−b+1)^2+3*(e+d+1)^2)
        by (simp add: nat-number ring-simps)
      moreover have is-qfN ((e+d−b+1)^2 + 3*(e+d+1)^2) 3
        by (unfold is-qfN-def, auto)
      ultimately have ?thesis by blast }
    ultimately have ?thesis by auto }
  ultimately show ?thesis by auto
qed

lemma qf3-even-general: ⟦ is-qfN A 3; A ∈ zEven ⟧
  ⟹ ∃ B. A = 4*B ∧ is-qfN B 3
proof −
  assume A ∈ zEven and is-qfN A 3
```

```
  then obtain a b where A = a^2 + 3*b^2
    and a^2 + 3*b^2 ∈ zEven by (unfold is-qfN-def, auto)
  thus ?thesis by (auto simp add: qf3-even)
qed

lemma qf3-oddprimedivisor-not:
  assumes ass: zprime P ∧ P ∈ zOdd ∧ Q>0 ∧ is-qfN (P*Q) 3 ∧ ¬ is-qfN P 3
  shows ∃ R. zprime R ∧ R ∈ zOdd ∧ R dvd Q ∧ ¬ is-qfN R 3
proof (rule ccontr, simp)
  assume ass2: ∀ R. R dvd Q ⟶ R ∈ zOdd ⟶ zprime R ⟶ is-qfN R 3
  (is ?A Q)
  obtain n::nat where n = nat Q by auto
  with ass have n: Q = int n by auto
  have (n > 0 ∧ is-qfN (P*int n) 3 ∧ ?A(int n)) ⟹ False (is ?B n ⟹ False)
  proof (induct n rule: less-induct)
    case (less n)
    hence IH: !!m. m<n ∧ ?B m ⟹ False
      and Bn: ?B n by auto
    show False
    proof (cases)
      assume odd: (int n) ∈ zOdd
      from Bn ass have zprime P ∧ int n > 0 ∧ is-qfN (P*int n) 3 ∧ ¬ is-qfN P 3
        by simp
      hence ∃ R. zprime R ∧ R dvd int n ∧ ¬ is-qfN R 3
        by (rule qfN-primedivisor-not)
      then obtain R where R: zprime R ∧ R dvd int n ∧ ¬ is-qfN R 3 by auto
      moreover with odd have R ∈ zOdd
      proof −
        from R obtain U where int n = R*U by (auto simp add: dvd-def)
        with odd show ?thesis by (auto dest: odd-mult-odd-prop)
      qed
      moreover from Bn have ?A (int n) by simp
      ultimately show False by auto
    next
      assume ¬ (int n) ∈ zOdd
      hence even: int n ∈ zEven by (rule not-odd-impl-even)
      hence (int n)*P ∈ zEven by (rule even-times-either)
      with Bn have  P*int n ∈ zEven ∧ is-qfN (P*int n) 3 by (simp add: mult-ac)
      hence ∃ B. P*(int n) = 4*B ∧ is-qfN B 3 by (simp only: qf3-even-general)
      then obtain B where B: P*(int n) = 4*B ∧ is-qfN B 3 by auto
      hence 2^2 dvd (int n)*P by (simp add: mult-ac)
      moreover have ¬ 2 dvd P
      proof (rule ccontr, simp)
        assume 2 dvd P
        with ass have P ∈ zOdd ∧ P ∈ zEven by (simp add: dvd-def zEven-def)
        thus False by (simp only: even-odd-conj)
      qed
      moreover have zprime 2 by (rule zprime-2)
      ultimately have 2^2 dvd int n
        by (rule-tac p=2 in zprime-power-zdvd-cancel-right)
      then obtain im::int where int n = 4*im by (auto simp add: dvd-def)
      moreover obtain m::nat where m = nat im by auto
```

```
    ultimately have m: n = 4*m by arith
    with B have is-qfN (P*int m) 3 by (auto simp add: int-mult)
    moreover from m Bn have m > 0 by auto
    moreover from m Bn have ?A (int m)
      by (auto simp add: zdvd-zmult int-mult)
    ultimately have Bm: ?B m by simp
    from Bn m have m < n by arith
    with IH Bm show False by auto
  qed
 qed
 with ass ass2 n show False by auto
qed

lemma qf3-oddprimedivisor:
 ⟦ zprime P; P ∈ zOdd; zgcd(a,b)=1; P dvd (a^2+3*b^2) ⟧
 ⟹ is-qfN P 3
proof −
 have lem: ∀ a b. (zprime P ∧ P ∈ zOdd ∧ zgcd(a,b)=1 ∧ P dvd (a^2+3*b^2))
   ⟶ is-qfN P 3 (is ?B P)
 proof (rule-tac x=P and V=λP. nat|P| in val-infinite-descent)
   fix x
   assume nat|x| = 0
   hence x = 0 by arith
   thus ?B x by (simp add: zprime-def)
 next
   fix x
   assume nat|x| > 0 and ¬ ?B x
   then obtain a b where abx: zprime x ∧ x ∈ zOdd ∧ zgcd(a,b)=1
     ∧ x dvd (a^2+3*b^2) ∧ ¬ is-qfN x 3 by auto
   then obtain M where M: a^2+3*b^2 = x*M by (auto simp add: dvd-def)
   let ?A = a^2 + 3*b^2
   from abx have x0: x > 0 ∧ x ∈ zOdd by (simp add: zprime-def)
   then obtain m where 2*|a−m*x|<x by (auto dest: best-odd-division-abs)
   then obtain c where cm: c = a−m*x ∧ 2*|c| < x by auto
   from x0 obtain n where 2*|b−n*x|<x by (auto dest: best-odd-division-abs)
   then obtain d where dn: d = b−n*x ∧ 2*|d| < x by auto
   let ?C = c^2+3*d^2
   have C3: is-qfN ?C 3 by (unfold is-qfN-def, auto)
   have C0: ?C > 0
   proof −
     have hlp: (3::int) ≥ 1 by simp
     with C3 have ?C ≥ 0 by (simp only: qfN-pos)
     hence ?C = 0 ∨ ?C > 0 by auto
     moreover
     { assume ?C = 0
       with hlp have c=0 ∧ d=0 by (rule qfN-zero)
       with cm dn have a = m*x ∧ b = n*x by simp
       hence x dvd a ∧ x dvd b by simp
       hence x dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
       with abx have False by (auto simp add: zprime-def) }
     ultimately show ?thesis by blast
   qed
```

```
have x dvd ?C
proof
  have ?C = |c| ^2 + 3*|d| ^2 by (simp only: power2-abs)
  also with cm dn have ... = (a−m*x) ^2 + 3*(b−n*x) ^2 by simp
  also have ... =
    a ^2 − 2*a*(m*x) + (m*x) ^2 + 3*(b ^2 − 2*b*(n*x) + (n*x) ^2)
    by (simp only: zdiff-power2)
  also with abx M have ... =
    x*M − x*(2*a*m + 3*2*b*n) + x ^2*(m ^2 + 3*n ^2)
    by (simp only: power-mult-distrib zadd-zmult-distrib2 mult-ac, auto)
  finally show ?C = x*(M − (2*a*m + 3*2*b*n) + x*(m ^2 + 3*n ^2))
    by (simp add: power2-eq-square zadd-zmult-distrib2 zdiff-zmult-distrib2)
qed
then obtain y where y: ?C = x*y by (auto simp add: dvd-def)
have yx: y < x
proof (rule ccontr)
  assume ¬ y < x hence xy: x−y ≤ 0 by simp
  have hlp: 2*|c| ≥ 0 ∧ 2*|d| ≥ 0 ∧ (3::nat) > 0 by simp
  from y have 4*x*y = 2 ^2*c ^2 + 3*2 ^2*d ^2 by simp
  hence 4*x*y = (2*|c|) ^2 + 3*(2*|d|) ^2
    by (auto simp add: power2-abs power-mult-distrib)
  with cm dn hlp have 4*x*y < x ^2 + 3*(2*|d|) ^2
    and (3::int) > 0 ∧ (2*|d|) ^2 < x ^2
    by (auto simp add: power-strict-mono)
  hence x*4*y < x ^2 + 3*x ^2 by (auto)
  also have ... = x*4*x by (simp add: power2-eq-square)
  finally have contr: (x−y)*(4*x) > 0 by (auto simp add: zdiff-zmult-distrib2)
  show False
  proof (cases)
    assume x−y = 0 with contr show False by auto
  next
    assume ¬ x−y =0 with xy have x−y < 0 by simp
    moreover from x0 have 4*x > 0 by simp
    ultimately have 4*x*(x−y) < 4*x*0 by (simp only: zmult-zless-mono2)
    with contr show False by auto
  qed
qed
have y0: y > 0
proof (rule ccontr)
  assume ¬ y > 0
  hence y ≤ 0 by simp
  moreover have y ≠ 0
  proof (rule ccontr)
    assume ¬ y≠0 hence y=0 by simp
    with y and C0 show False by auto
  qed
  ultimately have y < 0 by simp
  with x0 have x*y < x*0 by (simp only: zmult-zless-mono2)
  with C0 y show False by simp
qed
let ?g = zgcd(c,d)
have c ≠ 0 ∨ d ≠ 0
```

proof (*rule ccontr*)
assume ¬ (*c≠0* ∨ *d≠0*) **hence** *c=0* ∧ *d=0* **by** *simp*
with *C0* **show** *False* **by** *simp*
qed
then obtain *e f* **where** *ef*: *c* = *?g∗e* ∧ *d* = *?g* ∗ *f* ∧ *zgcd*(*e*,*f*)=*1*
by (*frule-tac a=c* **in** *make-zrelprime*, *auto*)
have *g2nonzero*: *?g^2* ≠ *0*
proof (*rule ccontr*, *clarsimp*)
assume *?g* =*0*
hence *nat*|*c*|=*0* ∧ *nat*|*d*|=*0*
by (*unfold zgcd-def*, *auto simp add*: *gcd-zero*)
hence *c=0* ∧ *d=0* **by** *arith*
with *C0* **show** *False* **by** *simp*
qed
let *?E* = *e^2* + *3∗f^2*
have *E3*: *is-qfN ?E 3* **by** (*unfold is-qfN-def*, *auto*)
have *CgE*: *?C* = *?g^2* ∗ *?E*
proof −
have *?g^2* ∗ *?E* = (*?g∗e*) *^2* + *3∗*(*?g∗f*) *^2*
by (*simp add*: *zadd-zmult-distrib2 power-mult-distrib*)
with *ef* **show** *?thesis* **by** *simp*
qed
hence *?g^2 dvd ?C* **by** (*simp add*: *dvd-def*)
with *y* **have** *g2dvdxy*: *?g^2 dvd y∗x* **by** (*simp add*: *mult-ac*)
moreover have *zgcd*(*x*, *?g^2*) = *1*
proof −
let *?h* = *zgcd*(*?g*, *x*)
have *?h dvd ?g* **and** *?g dvd c* **by** *auto*
hence *hc*: *?h dvd c* **by** (*rule zdvd-trans*)
have *?h dvd ?g* **and** *?g dvd d* **by** *auto*
hence *hd*: *?h dvd d* **by** (*rule zdvd-trans*)
have *hx*: *?h dvd x* **by** *simp*
hence *?h dvd m∗x* **by** (*rule zdvd-zmult*)
with *hc* **have** *?h dvd c+m∗x* **by** (*rule zdvd-zadd*)
with *cm* **have** *ha*: *?h dvd a* **by** *simp*
from *hx* **have** *?h dvd n∗x* **by** (*rule zdvd-zmult*)
with *hd* **have** *?h dvd d+n∗x* **by** (*rule zdvd-zadd*)
with *dn* **have** *hb*: *?h dvd b* **by** *simp*
with *ha* **have** *?h dvd zgcd*(*a*,*b*) **by** (*simp add*: *zgcd-greatest-iff*)
with *abx* **have** *?h dvd 1* **by** *simp*
hence *?h* = *1* **by** (*simp add*: *zgcd-geq-zero*)
hence *zgcd*(*?g^2*,*x*)=*1* **by** (*rule zgcd-1-power-left-distrib*)
thus *?thesis* **by** (*simp only*: *zgcd-commute*)
qed
ultimately have *?g^2 dvd y* **by** (*auto dest*: *zrelprime-zdvd-zmult*)
then obtain *w* **where** *w*: *y* = *?g^2* ∗ *w* **by** (*auto simp add*: *dvd-def*)
with *CgE y g2nonzero* **have** *Ewx*: *?E* = *x∗w* **by** *auto*
have *w0*: *w>0*
proof (*rule ccontr*)
assume ¬ *w>0* **hence** *w* ≤ *0* **by** *auto*
hence *w=0* ∨ *w<0* **by** *auto*
moreover

```
    { assume w=0 with w y0 have False by auto }
    moreover
    { assume wneg: w<0
      have ?g^2 ≥ 0 by (rule zero-le-power2)
      with g2nonzero have ?g^2 > 0 by arith
      with wneg have ?g^2*w < ?g^2*0 by (simp only: zmult-zless-mono2)
      with w y0 have False by auto }
    ultimately show False by blast
  qed
  have w-le-y: w ≤ y
  proof (rule ccontr)
    assume ¬ w ≤ y
    hence wy: w > y by simp
    have ?g^2 = 1 ∨ ?g^2 > 1
    proof −
      have ?g^2 ≥ 0 by (rule zero-le-power2)
      hence ?g^2 =0 ∨ ?g^2 > 0 by auto
      with g2nonzero show ?thesis by arith
    qed
    moreover
    { assume ?g^2 =1 with w wy have False by simp }
    moreover
    { assume g1: ?g^2 >1
      with w0 have w*1 < w*?g^2 by (auto dest: zmult-zless-mono2)
      with w have w < y by (simp add: zmult-1 mult-ac)
      with wy have False by auto }
    ultimately show False by blast
  qed
  from Ewx E3 abx w0 have
    zprime x ∧ x ∈ zOdd ∧ w > 0 ∧ is-qfN (x*w) 3 ∧ ¬ is-qfN x 3 by simp
  then obtain z where z: zprime z ∧ z ∈ zOdd ∧ z dvd w ∧ ¬ is-qfN z 3
    by (frule-tac P=x in qf3-oddprimedivisor-not, auto)
  from Ewx have w dvd ?E by simp
  with z have z dvd ?E by (auto dest: zdvd-trans)
  with z ef have zprime z ∧ z ∈ zOdd ∧ zgcd(e,f)=1 ∧ z dvd ?E ∧ ¬ is-qfN z 3
    by auto
  moreover have nat|z| < nat|x|
  proof −
    have z ≤ w
    proof (rule ccontr)
      assume ¬ z ≤ w hence w < z by auto
      with w0 have ¬ z dvd w by (rule zdvd-not-zless)
      with z show False by simp
    qed
    with w-le-y yx have z < x by simp
    with z have |z| < |x| by (simp add: zprime-def)
    thus ?thesis by auto
  qed
  ultimately show ∃ z. nat|z| < nat|x| ∧ ¬ ?B z by auto
qed
assume zprime P and P ∈ zOdd and zgcd(a,b)=1 and P dvd a^2+3*b^2
with lem show ?thesis by blast
```

```
qed

lemma qf3-cube-prime-impl-cube-form:
  assumes ab-relprime: zgcd(a,b)=1 and abP: P^3 = a^2 + 3*b^2
  and P: zprime P ∧ P ∈ zOdd
  shows is-cube-form a b
proof −
  from abP have qfP3: is-qfN (P^3) 3 by (auto simp only: is-qfN-def)
  have PvdP3: P dvd P^3 by (simp add: nat-number)
  with abP ab-relprime P have qfP: is-qfN P 3 by (simp only: qf3-oddprimedivisor)
  then obtain p q where pq: P = p^2 + 3*q^2 by (auto simp only: is-qfN-def)
  with P abP ab-relprime have zprime (p^2 + 3*q^2) ∧ (3::int) > 1
    ∧ (p^2+3*q^2)^3 = a^2+3*b^2 ∧ zgcd(a,b)=1 by auto
  hence ab: |a| = |p^3 − 3*3*p*q^2| ∧ |b| = |3*p^2*q − 3*q^3|
    by (rule qfN-cube-prime)
  hence a: a = p^3 − 9*p*q^2 ∨ a = −(p^3) + 9*p*q^2 by arith
  from ab have b: b = 3*p^2*q − 3*q^3 ∨ b = −(3*p^2*q) + 3*q^3 by arith
  obtain r s where r: r = −p and s: s = −q  by simp
  show ?thesis
  proof (cases)
    assume a1: a = p^3− 9*p*q^2
    show ?thesis
    proof (cases)
      assume b1: b = 3*p^2*q − 3*q^3
      with a1 show ?thesis by (unfold is-cube-form-def, auto)
    next
      assume ¬ b = 3*p^2*q − 3*q^3
      with b have b = − 3*p^2*q + 3*q^3 by simp
      with s have b = 3*p^2*s − 3*s^3 by (simp add: power3-minus)
      moreover from a s have a = p^3 − 9*p*s^2 by (simp add: power2-minus)
      ultimately show ?thesis by (unfold is-cube-form-def, auto)
    qed
  next
    assume ¬ a = p^3 − 9*p*q^2
    with a have a = −(p^3) + 9*p*q^2 by simp
    with r have ar: a = r^3 − 9*r*q^2 by (simp add: power3-minus)
    show ?thesis
    proof (cases)
      assume b1: b = 3*p^2*q − 3*q^3
      with r have b = 3*r^2*q − 3*q^3 by (simp add: power2-minus)
      with ar show ?thesis by (unfold is-cube-form-def, auto)
    next
      assume ¬ b = 3*p^2*q − 3*q^3
      with b have b = − 3*p^2*q + 3*q^3 by simp
      with r s have b = 3*r^2*s − 3*s^3
        by (simp add: power2-minus power3-minus)
      moreover from ar s have a = r^3 − 9*r*s^2 by (simp add: power2-minus)
      ultimately show ?thesis by (unfold is-cube-form-def, auto)
    qed
  qed
qed
```

```
lemma cube-form-mult: ⟦ is-cube-form a b; is-cube-form c d; |e| = 1 ⟧
  ⟹ is-cube-form (a*c+e*3*b*d) (a*d−e*b*c)
proof −
  assume ab: is-cube-form a b and c-d: is-cube-form c d and e: |e| = 1
  from ab obtain p q where pq: a = p^3 − 9*p*q^2 ∧ b = 3*p^2*q − 3*q^3
    by (auto simp only: is-cube-form-def)
  from c-d obtain r s where rs: c = r^3 − 9*r*s^2 ∧ d = 3*r^2*s − 3*s^3
    by (auto simp only: is-cube-form-def)
  let ?t = p*r + e*3*q*s
  let ?u = p*s − e*r*q
  have e2: e^2=1
  proof −
    from e have e=1 ∨ e=−1 by simp
    moreover
    { assume e=1 hence ?thesis by auto }
    moreover
    { assume e=−1 hence ?thesis by (simp add: power2-minus) }
    ultimately show ?thesis by blast
  qed
  hence e*e^2 = e by simp
  hence e3: e*1 = e^3 by (simp only: cube-square)
  have a*c+e*3*b*d = ?t^3 − 9*?t*?u^2
  proof −
    have ?t^3 − 9*?t*?u^2 = p^3*r^3 + e*9*p^2*q*r^2*s + e^2*27*p*q^2*r*s^2
    + e^3*27*q^3*s^3 − 9*p*p^2*r*s^2 + e*18*p^2*q*r^2*s − e^2*9*p*q^2*(r*r^2)
      − e*27*p^2*q*(s*s^2) + e^2*54*p*q^2*r*s^2 − e*e^2*27*(q*q^2)*r^2*s
      by (simp add: nat-number ring-simps)
    also with e2 e3 have ... =
      p^3*r^3 + e*27*p^2*q*r^2*s + 81*p*q^2*r*s^2 + e*27*q^3*s^3
      − 9*p^3*r*s^2 − 9*p*q^2*r^3 − e*27*p^2*q*s^3 − e*27*q^3*r^2*s
      by (simp add: cube-square zmult-1)
    also with pq rs have ... = a*c + e*3*b*d
      by (simp only: zdiff-zmult-distrib zdiff-zmult-distrib2 mult-ac)
    finally show ?thesis by auto
  qed
  moreover have a*d−e*b*c = 3*?t^2*?u − 3*?u^3
  proof −
    have 3*?t^2*?u − 3*?u^3 =
      3*(p*p^2)*r^2*s − e*3*p^2*q*(r*r^2) + e*18*p^2*q*r*s^2
      − e^2*18*p*q^2*r^2*s + e^2*27*p*q^2*(s*s^2) − e*e^2*27*(q*q^2)*r*s^2
      − 3*p^3*s^3 + e*9*p^2*q*r*s^2 − e^2*9*p*q^2*r^2*s + e^3*3*r^3*q^3
      by (simp add: nat-number ring-simps)
    also with e2 e3 have ... = 3*p^3*r^2*s − e*3*p^2*q*r^3 + e*18*p^2*q*r*s^2
      − 18*p*q^2*r^2*s + 27*p*q^2*s^3 − e*27*q^3*r*s^2 − 3*p^3*s^3
      + e*9*p^2*q*r*s^2 − 9*p*q^2*r^2*s + e*3*r^3*q^3
      by (simp add: cube-square zmult-1)
    also with pq rs have ... = a*d−e*b*c
      by (simp only: zdiff-zmult-distrib zdiff-zmult-distrib2 mult-ac)
    finally show ?thesis by auto
  qed
  ultimately show ?thesis by (auto simp only: is-cube-form-def)
qed
```

lemma *qf3-cube-primelist-impl-cube-form*: ⟦ *primel ps*; *int* (*prod ps*) ∈ *zOdd* ⟧ ⟹
(!! *a b*. *zgcd*(*a*,*b*)=*1* ⟹ *a*^*2* + *3*∗*b*^*2* = (*int*(*prod ps*))^*3* ⟹ *is-cube-form a b*)
proof (*induct ps*)
case *Nil* **hence** *ab1*: *a*^*2* + *3*∗*b*^*2* = *1* **by** *simp*
have *b0*: *b*=*0*
proof (*rule ccontr*)
assume *b*≠*0*
hence *b*^*2*>*0* **by** (*simp add*: *zero-less-power2*)
hence *3*∗*b*^*2* > *1* **by** *arith*
with *ab1* **have** *a*^*2* < *0* **by** *arith*
moreover have *a*^*2* ≥ *0* **by** (*rule zero-le-power2*)
ultimately show *False* **by** *auto*
qed
with *ab1* **have** *a1*: (*a*=*1* ∨ *a*=−*1*) **by** (*auto simp add*: *power2-eq-square zmult-eq-1-iff*)
then obtain *p* **and** *q* **where** *p*=*a* **and** *q*=(*0*::*int*) **by** *simp*
with *a1* **and** *b0* **have** *a* = *p*^*3* − *9*∗*p*∗*q*^*2* ∧ *b* = *3*∗*p*^*2*∗*q* − *3*∗*q*^*3* **by** *auto*
thus *is-cube-form a b* **by** (*auto simp only*: *is-cube-form-def*)
next
case (*Cons p ps*) **hence** *ass*: *zgcd*(*a*,*b*)=*1* ∧ *int*(*prod* (*p*#*ps*)) ∈ *zOdd*
∧ *a*^*2*+*3*∗*b*^*2* = *int*(*prod* (*p*#*ps*))^*3* ∧ *primel ps* ∧ *zprime* (*int p*)
and *IH*: !! *u v*. *zgcd*(*u*,*v*)=*1* ∧ *u*^*2*+*3*∗*v*^*2* = *int*(*prod ps*)^*3*
∧ *int*(*prod ps*) ∈ *zOdd* ⟹ *is-cube-form u v*
by (*auto simp add*: *primel-def prime-impl-zprime-int*)
let *?w* = *int* (*prod* (*p*#*ps*))
let *?X* = *int* (*prod ps*)
let *?p* = *int p*
have *ge3-1*: (*3*::*int*) ≥ *1* **by** *auto*
have *pw*: *?w* = *?p* ∗ *?X* ∧ *?p* ∈ *zOdd* ∧ *?X* ∈ *zOdd*
proof (*safe*)
have *prod* (*p*#*ps*) = *p* ∗ *prod ps* **by** *simp*
thus *wpx*: *?w* = *?p* ∗ *?X* **by** (*auto simp only*: *zmult-int*)
with *ass* **show** *?p* ∈ *zOdd* **by** (*auto dest*: *odd-mult-odd-prop*)
from *wpx* **have** *?w* = *?X*∗*?p* **by** *simp*
with *ass* **show** *?X* ∈ *zOdd* **by** (*auto dest*: *odd-mult-odd-prop*)
qed
have *is-qfN ?p 3*
proof −
from *ass* **have** *a*^*2*+*3*∗*b*^*2* = (*?p*∗*?X*)^*3* **by** (*simp add*: *zmult-int*)
hence *?p dvd a*^*2*+*3*∗*b*^*2* **by** (*simp add*: *nat-number ring-simps*)
moreover from *ass* **have** *zprime ?p* **and** *zgcd*(*a*,*b*)=*1* **by** *simp*
moreover from *pw* **have** *?p* ∈ *zOdd* **by** *simp*
ultimately show *?thesis* **by** (*simp only*: *qf3-oddprimedivisor*)
qed
then obtain α β **where** *alphabeta*: *?p* = α^*2* + *3*∗β^*2*
by (*auto simp add*: *is-qfN-def*)
have α ≠ *0*
proof (*rule ccontr*, *simp*)
assume α = *0* **with** *alphabeta* **have** *3 dvd ?p* **by** *auto*
with *pw* **have** *w3*: *3 dvd ?w* **by** (*simp only*: *zdvd-zmult2*)
then obtain *v* **where** *?w* = *3*∗*v* **by** (*auto simp add*: *dvd-def*)
with *ass* **have** *vab*: *27*∗*v*^*3* = *a*^*2* + *3*∗*b*^*2* **by** (*simp add*: *power-mult-distrib*)

```
    hence a^2 = 3*(9*v^3 - b^2) by auto
    hence 3 dvd a^2 by (unfold dvd-def, blast)
    moreover have zprime 3 by (rule zprime-3)
    ultimately have a3: 3 dvd a by (rule-tac p=3 in zprime-zdvd-power)
    then obtain c where c: a = 3*c by (auto simp add: dvd-def)
    with vab have 27*v^3 = 9*c^2 + 3*b^2 by (simp add: power-mult-distrib)
    hence b^2 = 3*(3*v^3 - c^2) by auto
    hence 3 dvd b^2 by (unfold dvd-def, blast)
    moreover have zprime 3 by (rule zprime-3)
    ultimately have 3 dvd b by (rule-tac p=3 in zprime-zdvd-power)
    with a3 have 3 dvd zgcd(a,b) by (simp add: zgcd-greatest-iff)
    with ass show False by simp
  qed
  moreover from alphabeta pw ass have
    zprime (α^2 + 3*β^2) ∧ α^2+3*β^2 ∈ zOdd ∧ (3::int) ≥ 1 by auto
  ultimately obtain c d where cdp:
    (α^2+3*β^2)^3 = c^2+3*d^2 ∧ zgcd(c, 3*d)=1
    by (blast dest: qfN-oddprime-cube)
  with ass pw alphabeta have ∃ u v. a^2+3*b^2 = (u^2 + 3*v^2)*(c^2+3*d^2)
    ∧ zgcd(u,v)=1 ∧ (∃ e. a = c*u+e*3*d*v ∧ b = c*v−e*d*u ∧ |e| = 1)
    by (rule-tac A=?w and n=3 in qfN-power-div-prime, auto)
  then obtain u v e where uve: a^2+3*b^2 = (u^2+3*v^2)*(c^2+3*d^2)
    ∧ zgcd(u,v)=1 ∧ a = c*u+e*3*d*v ∧ b = c*v−e*d*u ∧ |e| = 1 by blast
  moreover have is-cube-form u v
  proof −
    have uvX: u^2+3*v^2 = ?X^3
    proof −
      from ass have p0: ?p ≠ 0 by (simp add: zprime-def)
      from pw have ?p^3*?X^3 = ?w^3 by (simp add: power-mult-distrib)
      also with ass have ... = a^2+3*b^2 by simp
      also with uve have ... = (u^2+3*v^2)*(c^2+3*d^2) by auto
      also with cdp alphabeta have ... = ?p^3 * (u^2+3*v^2) by (simp only: mult-ac)
      finally have ?p^3*(u^2+3*v^2−?X^3) = 0 by auto
      with p0 show ?thesis by auto
    qed
    with pw IH uve show ?thesis by simp
  qed
  moreover have is-cube-form c d
  proof −
    have zgcd(c,d) = 1
    proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
      fix h::int assume h dvd c and h dvd d and h: zprime h
      hence h dvd c*u + d*(e*3*v) ∧ h dvd c*v−d*(e*u)
        by (simp add: zdvd-zmult2 zdvd-zadd zdvd-zdiff)
      with uve have h dvd a ∧ h dvd b by (auto simp only: mult-ac)
      with ass h show False by (auto simp add: zgcd1-iff-no-common-primedivisor)
    qed
    with pw cdp ass alphabeta show ?thesis
      by (rule-tac P=?p in qf3-cube-prime-impl-cube-form, auto)
  qed
  ultimately show is-cube-form a b by (simp only: cube-form-mult)
qed
```

```
lemma qf3-cube-impl-cube-form:
  assumes ass: zgcd(a,b)=1 ∧ a^2 + 3*b^2 = w^3 ∧ w ∈ zOdd
  shows is-cube-form a b
proof −
  have ∃ ps. primel ps ∧ int (prod ps) = w
  proof −
    have wpos: w ≥ 1
    proof −
      have b^2 ≥ 0 by (rule zero-le-power2)
      hence 3*b^2 ≥ 0 by arith
      moreover have a^2≥0 by (rule zero-le-power2)
      ultimately have a^2 + 3*b^2 ≥ 0 by arith
      with ass have w3pos: w^3 ≥ 0 by simp
      have w≥0
      proof (rule ccontr)
        assume ¬ w≥0 hence −w > 0 by auto
        hence (−1 ∗ w) ^3 > 0 by (auto simp only: zero-less-power)
        hence (−1) ^3∗ (w^3) > 0 by (simp only: power-mult-distrib)
        hence w^3 < 0 by (simp add: neg-one-odd-power)
        with w3pos show False by auto
      qed
      moreover have w ≠ 0
      proof (rule ccontr)
        assume ¬w≠0 with ass have 0 ∈ zOdd by simp
        moreover have 0 ∈ zEven by (simp add: zEven-def)
        ultimately show False by (auto simp add: odd-iff-not-even)
      qed
      ultimately show ?thesis by (auto)
    qed
    hence w=1 ∨ Suc 0 < nat w by auto
    moreover
    { assume w=1
      hence primel [] ∧ int (prod []) = w by (auto simp add: primel-def)
      hence ?thesis by (simp only: exI) }
    moreover
    { assume Suc 0 < nat w
      hence ∃ l. primel l ∧ prod l = nat w by (rule factor-exists)
      then obtain ps where ps: primel ps ∧ prod ps = nat w by auto
      with wpos have ?thesis by auto }
    ultimately show ?thesis by blast
  qed
  with ass show ?thesis by (auto dest: qf3-cube-primelist-impl-cube-form)
qed
```

4.6 Existence ($N = 3$)

This part contains the proof that all prime numbers $\equiv 1 \bmod 6$ can be written as $x^2 + 3y^2$.

First show $(\frac{a}{p})(\frac{b}{p}) = (\frac{ab}{p})$, where p is an odd prime.

lemma *Legendre-zmult*: ⟦ *p > 2*; *zprime p* ⟧
⟹ (*Legendre* (*a*b*) *p*) = (*Legendre a p*)*(*Legendre b p*)
proof −
assume *p2*: *p > 2* **and** *prp*: *zprime p*
let *?p12* = *nat*(((*p*) − *1*) *div 2*)
let *?Labp* = *Legendre* (*a*b*) *p*
let *?Lap* = *Legendre a p*
let *?Lbp* = *Legendre b p*
from *p2 prp* **have** [*?Labp* = (*a*b*)^*?p12*] (*mod p*)
by (*simp only*: *Euler-Criterion*)
hence [*a^?p12* * *b^?p12* = *?Labp*] (*mod p*)
by (*simp only*: *power-mult-distrib zcong-sym*)
moreover from *p2 prp* **have** [*?Lap* * *?Lbp* = *a^?p12*b^?p12*] (*mod p*)
by (*simp only*: *Euler-Criterion zcong-zmult*)
ultimately have [*?Lap* * *?Lbp* = *?Labp*] (*mod p*)
by (*rule-tac b*=*a^?p12* * *b^?p12* **in** *zcong-trans*)
then obtain *k* **where** *k*: *?Labp* = (*?Lap*?Lbp*) + *p* * *k*
by (*auto simp add*: *zcong-iff-lin*)
have *k*=*0*
proof (*rule ccontr*)
assume *k* ≠ *0* **hence** |*k*| = *1* ∨ |*k*| > *1* **by** *arith*
moreover
{ assume |*k*|= *1*
with *p2* **have** |*k*|*p > *2* **by** *auto* **}**
moreover
{ assume *k1*: |*k*| > *1*
with *p2* **have** |*k*|*2 < |*k*|*p
by (*simp only*: *zmult-zless-mono2*)
with *k1* **have** |*k*|*p > *2* **by** *auto* **}**
ultimately have |*k*|*p > *2* **by** *auto*
moreover from *p2* **have** |*p*| = *p* **by** *auto*
ultimately have |*k*p*| > *2* **by** (*auto simp only*: *abs-mult*)
moreover from *k* **have** *?Labp* − *?Lap*?Lbp* = *k*p* **by** *auto*
ultimately have |*?Labp* − *?Lap*?Lbp*| > *2* **by** *auto*
moreover have *?Labp* = *1* ∨ *?Labp* = *0* ∨ *?Labp* = −*1*
by (*simp add*: *Legendre-def*)
moreover have *?Lap*?Lbp* = *1* ∨ *?Lap*?Lbp* = *0* ∨ *?Lap*?Lbp* = −*1*
by (*auto simp add*: *Legendre-def*)
ultimately show *False* **by** *auto*
qed
with *k* **show** *?thesis* **by** *auto*
qed

Now show $(\frac{-3}{p}) = +1$ for primes $p \equiv 1 \bmod 6$.

lemma *Legendre-1mod6*: *zprime* (*6*m+1*) ⟹ *Legendre* (−*3*) (*6*m+1*) = *1*
proof −
let *?p* = *6*m+1*
let *?L* = *Legendre* (−*3*) *?p*
let *?L1* = *Legendre* (−*1*) *?p*
let *?L3* = *Legendre 3 ?p*
assume *p*: *zprime ?p*

```
have neg1cube: (−1::int) ^3 = −1 by (simp add: power3-minus)
have m1: m ≥ 1
proof (rule ccontr)
  assume ¬ m ≥ 1 hence m ≤ 0 by simp
  with p show False by (auto simp add: zprime-def)
qed
hence pn3: ?p ≠ 3 and p2: ?p > 2 by auto
with p have ?L = (Legendre (−1) ?p) * (Legendre 3 ?p)
  by (frule-tac a=−1 and b=3 in Legendre-zmult, auto)
moreover have [Legendre (−1) ?p = (−1) ^nat m] (mod ?p)
proof −
  from p p2 have [?L1 = (−1) ^(nat(((?p) − 1) div 2))] (mod ?p)
    by (simp only: Euler-Criterion)
  moreover have nat ((?p − 1) div 2) = 3* nat m
  proof −
    have (?p − 1) div 2 = 3*m by auto
    hence nat((?p − 1) div 2) = nat (3*m) by simp
    moreover have (3::int) ≥ 0 by simp
    ultimately show ?thesis by (simp add: nat-mult-distrib)
  qed
  moreover with neg1cube have (−1::int) ^(3*nat m) = (−1) ^nat m
    by (simp only: power-mult)
  ultimately show ?thesis by auto
qed
moreover have ?L3 = (−1) ^nat m
proof −
  have ?L3 * (Legendre ?p 3) = (−1) ^nat m
  proof −
    have 3 ∈ zOdd ∧ ?p ∈ zOdd by (unfold zOdd-def, auto)
    with p pn3 have ?L3 * (Legendre ?p 3) = (−1::int) ^(3*nat m)
      by (simp add: zprime-3 Quadratic-Reciprocity nat-mult-distrib)
    with neg1cube show ?thesis by (simp add: power-mult)
  qed
  moreover have Legendre ?p 3 = 1
  proof −
    have [1^2 = ?p] (mod 3) by (unfold zcong-def dvd-def, auto)
    hence QuadRes 3 ?p by (unfold QuadRes-def, blast)
    moreover have ¬ [?p = 0] (mod 3)
    proof (rule ccontr, simp)
      assume [?p = 0] (mod 3)
      hence 3 dvd ?p by (simp add: zcong-def)
      moreover have 3 dvd 6*m by (auto simp add: dvd-def)
      ultimately have 3 dvd ?p− 6*m by (simp only: zdvd-zdiff)
      hence (3::int) dvd 1 by simp
      thus False by auto
    qed
    ultimately show ?thesis by (unfold Legendre-def, auto)
  qed
  ultimately show ?thesis by auto
qed
ultimately have [?L = (−1) ^(nat m)*(−1) ^(nat m)] (mod ?p)
  by (auto dest: zcong-scalar)
```

hence [*?L* = (−1) ^ ((*nat m*)+(*nat m*))] (*mod ?p*) **by** (*simp only*: *power-add*)
moreover have (*nat m*)+(*nat m*) = 2∗(*nat m*) **by** *auto*
ultimately have [*?L* = (−1) ^ (2∗(*nat m*))] (*mod ?p*) **by** *simp*
hence [*?L* = ((−1) ^ 2) ^ (*nat m*)] (*mod ?p*) **by** (*simp only*: *power-mult*)
hence [1 = *?L*] (*mod ?p*) **by** (*auto simp add*: *zcong-sym*)
hence *?p dvd* 1 − *?L* **by** (*simp only*: *zcong-def*)
moreover have *?L* = −1 ∨ *?L* = 0 ∨ *?L* = 1 **by** (*simp add*: *Legendre-def*)
ultimately have *?p dvd* 2 ∨ *?p dvd* 1 ∨ *?L* = 1 **by** *auto*
moreover
{ assume *?p dvd* 2 ∨ *?p dvd* 1
with *p2* **have** *False* **by** (*auto simp add*: *zdvd-not-zless*) **}**
ultimately show *?thesis* **by** *auto*
qed

Use this to prove that such primes can be written as $x^2 + 3y^2$.

lemma *qf3-prime-exists*: *zprime* (6∗m+1) ⟹ ∃ *x y*. 6∗m+1 = *x* ^ 2 + 3∗*y* ^ 2
proof −
let *?p* = 6∗m+1
assume *p*: *zprime ?p*
hence *Legendre* (−3) *?p* = 1 **by** (*rule Legendre-1mod6*)
moreover
{ assume ¬ *QuadRes ?p* (−3)
hence *Legendre* (−3) *?p* ≠ 1 **by** (*unfold Legendre-def*, *auto*) **}**
ultimately have *QuadRes ?p* (−3) **by** *auto*
then obtain *s* **where** *s*: [*s* ^ 2 = −3] (*mod ?p*) **by** (*auto simp add*: *QuadRes-def*)
hence *?p dvd s* ^ 2 − (−3::*int*) **by** (*unfold zcong-def*, *simp*)
moreover have *s* ^ 2 −(−3::*int*) = *s* ^ 2 + 3 **by** *arith*
ultimately have *?p dvd s* ^ 2 + 3∗1 ^ 2 **by** *auto*
moreover have *zgcd*(*s*,1) = 1 **by** (*unfold zgcd-def*, *auto*)
moreover have *?p* ∈ *zOdd*
proof −
have *?p* = 2∗(3∗*m*)+1 **by** *simp*
thus *?thesis* **by** (*unfold zOdd-def*, *blast*)
qed
moreover from *p* **have** *zprime ?p* **by** *simp*
ultimately have *is-qfN ?p* 3 **by** (*simp only*: *qf3-oddprimedivisor*)
thus *?thesis* **by** (*unfold is-qfN-def*, *auto*)
qed

end

5 Fermat's last theorem, case $n = 3$

theory *Fermat3*
imports *QuadForm*
begin

Proof of Fermat's last theorem for the case $n = 3$:

$$\forall x, y, z : \ x^3 + y^3 = z^3 \Longrightarrow xyz = 0.$$

lemma *factor-sum-cubes*: (*x*::*int*) ^ 3 + *y* ^ 3 = (*x*+*y*)∗(*x* ^ 2 − *x*∗*y* + *y* ^ 2)

```
  by (simp add: nat-number ring-simps)

lemma two-not-abs-cube: |x^3| = (2::int) ⟹ False
proof −
  assume |x^3| = 2
  hence x32: |x| ^3 = 2 by (simp only: abs-power3-distrib)
  have |x| ≥ 0 by simp
  moreover
  { assume |x| = 0 ∨ |x| = 1 ∨ |x| = 2
    with x32 have False by (auto simp add: power-0-left) }
  moreover
  { assume |x| > 2
    moreover have (0::int) ≤ 2 and (0::nat) < 3 by auto
    ultimately have |x| ^3 > 2^3 by (simp only: power-strict-mono)
    with x32 have False by simp }
  ultimately show False by arith
qed
```

Shows there exists no solution $v^3 + w^3 = x^3$ with $vwx \neq 0$ and $\gcd(v, w) = 1$ and x even, by constructing a solution with a smaller $|x^3|$.

```
lemma no-rewritten-fermat3:
  ¬ (∃ v w. v^3+w^3 = x^3 ∧ v*w*x ≠ 0 ∧ x ∈ zEven ∧ zgcd(v,w)=1) (is ?Q x)
proof (rule-tac x=x and V=λx. nat|x^3| in val-infinite-descent)
  fix x
  assume nat|x^3| = 0 hence x^3 = 0 by arith
  hence x=0 by auto
  thus ?Q x by auto
next
  fix x
  assume x3pos: 0 < nat|x^3| and ¬ ?Q x
  then obtain v w where vwx:
    v^3+w^3=x^3 ∧ v*w*x ≠ 0 ∧ x ∈ zEven ∧ zgcd(v,w)=1 (is ?P v w x)
    by auto
  have ∃ α β γ. ?P α β γ ∧ nat|γ ^3| < nat|x^3|
  proof −
    — obtain coprime p and q such that v = p + q and w = p − q
    have vwOdd: v ∈ zOdd ∧ w ∈ zOdd
    proof (rule ccontr, case-tac v ∈ zOdd, simp-all)
      assume v ∉ zOdd hence ve: v ∈ zEven by (rule not-odd-impl-even)
      hence v^3 ∈ zEven by (simp only: power-preserves-even)
      moreover from vwx have x^3 ∈ zEven by (simp only: power-preserves-even)
      ultimately have (x^3−v^3) ∈ zEven by (simp only: even-minus-even)
      moreover from vwx have x^3−v^3 = w^3 by simp
      ultimately have w^3 ∈ zEven by simp
      hence w ∈ zEven by (simp only: power-preserves-even)
      with ve have 2 dvd v ∧ 2 dvd w by (auto simp add: zEven-def)
      hence 2 dvd zgcd(v,w) by (simp add: zgcd-greatest-iff)
      with vwx show False by simp
    next
      assume vo: v ∈ zOdd and w ∉ zOdd
      hence w ∈ zEven by (simp add: not-odd-impl-even)
```

```
  with vo have v^3 ∈ zOdd and w^3 ∈ zEven
    by (auto simp only: power-preserves-even power-preserves-odd)
  hence w^3 + v^3 ∈ zOdd by (simp only: even-plus-odd)
  with vwx have x^3 ∈ zOdd by (simp add: zadd-commute)
  hence x ∈ zOdd by (simp only: power-preserves-odd)
  with vwx show False by (auto simp add: odd-iff-not-even)
qed
hence v+w ∈ zEven ∧ v−w ∈ zEven by (simp add: odd-minus-odd odd-plus-odd)
then obtain p q where pq: v+w = 2*p ∧ v−w = 2*q
  by (auto simp add: zEven-def)
hence vw: v = p+q ∧ w = p−q by auto
— show that x^3 = (2p)(p^2 + 3q^2) and that these factors are
— either coprime (first case), or have 3 as g.c.d. (second case)
have vwpq: v^3 + w^3 = (2*p)*(p^2 + 3*q^2)
proof −
  have 2*(v^3 + w^3) = 2*(v+w)*(v^2 − v*w + w^2)
    by (simp only: factor-sum-cubes)
  also from pq have ... = 4*p*(v^2 − v*w + w^2) by auto
  also have ... = p*((v+w)^2 + 3*(v−w)^2)
    by (simp add: nat-number ring-simps)
  also with pq have ... = p*((2*p)^2 + 3*(2*q)^2) by simp
  also have ... = 2*(2*p)*(p^2+3*q^2) by (simp add: power-mult-distrib)
  finally show ?thesis by simp
qed
let ?g = zgcd(2*p, p^2+3*q^2)
have g1: ?g ≥ 1
proof (rule ccontr)
  assume ¬ ?g ≥ 1 hence ?g < 0 ∨ ?g = 0 by auto
  moreover have ?g ≥ 0 by (rule zgcd-geq-zero)
  ultimately have ?g = 0 by simp
  hence nat|2*p| = 0 by (unfold zgcd-def, simp add: gcd-zero)
  hence p = 0 by arith
  with vwpq vwx x3pos show False by auto
qed
have gOdd: ¬ 2 dvd ?g
proof (rule ccontr, simp)
  assume 2 dvd ?g
  hence2 dvd p^2+3*q^2 by (simp only: zgcd-greatest-iff)
  then obtain k where k: p^2 + 3*q^2 = 2*k by (auto simp add: dvd-def)
  hence 2*(k − 2*q^2) = p^2−q^2 by auto
  with vw have v*w = 2*(k − 2*q^2) by (simp add: zspecial-product)
  hence v*w ∈ zEven by (auto simp only: zEven-def)
  hence v ∈ zEven ∨ w ∈ zEven by (simp add: even-product)
  with vwOdd show False by (auto simp add: odd-iff-not-even)
qed
— first case: p is not a multiple of 3; hence 2p and p^2 + 3q^2
— are coprime; hence both are cubes
{ assume p3: ¬ 3 dvd p
  have g3: ¬ 3 dvd ?g
  proof (rule ccontr, simp)
    assume 3 dvd ?g hence 3 dvd 2*p by (simp add: zgcd-greatest-iff)
    hence (3::int) dvd 2 ∨ 3 dvd p
```

```
    by (auto simp only: zprime-3 zprime-zdvd-zmult-general)
  with p3 show False by arith
qed
have pq-relprime: zgcd(p,q)=1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and zp: z dvd p and zq: z dvd q
  hence z dvd p+q ∧ z dvd p−q by (auto simp only: zdvd-zadd zdvd-zdiff)
  with vw have z dvd v ∧ z dvd w by simp
  with z vwx show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
have factors-relprime: ?g = 1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and z2p: z dvd 2*p and zpq: z dvd p^2+3*q^2
  hence zg: z dvd ?g by (simp add: zgcd-greatest-iff)
  with gOdd have z ≠ 2 by auto
  with z have zg2: z > 2 by (auto simp add: zprime-def)
  from z z2p have z dvd 2 ∨ z dvd p by (simp only: zprime-zdvd-zmult-general)
  moreover
  { assume z dvd 2
    hence z ≤ 2 by (auto simp add: zdvd-imp-le)
    with zg2 have False by simp }
  ultimately have zp: z dvd p by auto
  hence z dvd p^2 by (auto simp add: power2-eq-square zdvd-zmult2)
  with zpq have z dvd p^2+3*q^2−p^2 by (simp only: zdvd-zdiff)
  hence z dvd 3*q^2 by auto
  with z have z dvd 3 ∨ z dvd q^2 by (simp only: zprime-zdvd-zmult-general)
  moreover
  { assume z dvd 3
    hence z ≤ 3 by (auto simp add: zdvd-imp-le)
    with zg2 have z = 3 by auto
    with zg g3 have False by auto }
  ultimately have z dvd q^2 by auto
  with z have z dvd q by (rule zprime-zdvd-power)
  with zp z pq-relprime show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
moreover from vwx vwpq have pqx: (2*p)*(p^2 + 3*q^2) = x^3 by auto
moreover have triv3: 3 = nat 3 ∧ nat 3 > 1 ∧ 3 ∈ zOdd
  by (unfold zOdd-def, auto)
ultimately have ∃ c. 2*p = c^3
  by (simp only: int-relprime-odd-power-divisors)
then obtain c where c: c^3 = 2*p by auto
from pqx factors-relprime have zgcd(p^2 + 3*q^2, 2*p) = 1
  and (p^2 + 3*q^2)*(2*p) = x^3 by (auto simp add: zgcd-commute mult-ac)
with triv3 have ∃ d. p^2 + 3*q^2 = d^3
  by (simp only: int-relprime-odd-power-divisors)
then obtain d where d: p^2 + 3*q^2 = d^3 by auto
have d ∈ zOdd
proof (rule ccontr)
  assume d ∉ zOdd hence d ∈ zEven by (rule not-odd-impl-even)
  hence d^3 ∈ zEven by (simp only: power-preserves-even)
```

```
  hence 2 dvd d^3 by (simp add: zEven-def dvd-def)
  moreover have 2 dvd 2*p by (rule zdvd-triv-left)
  ultimately have 2 dvd zgcd(2*p, d^3) by (simp add: zgcd-greatest-iff)
  with d factors-relprime show False by auto
qed
with d pq-relprime have zgcd(p,q)=1 ∧ p^2 + 3*q^2 = d^3 ∧ d ∈ zOdd
  by simp
hence is-cube-form p q by (rule qf3-cube-impl-cube-form)
then obtain a b where p = a^3 - 9*a*b^2 ∧ q = 3*a^2*b - 3*b^3
  by (unfold is-cube-form-def, auto)
hence ab: p = a*(a+3*b)*(a- 3*b) ∧ q = b*(a+b)*(a-b)*3
  by (simp add: nat-number ring-simps)
with c have abc: (2*a)*(a+3*b)*(a- 3*b) = c^3 by auto
have ab-relprime: zgcd(a,b)=1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and za: z dvd a and zb: z dvd b
  with ab have z dvd p ∧ z dvd q by (simp add: zdvd-zmult2)
  with z pq-relprime show
    False by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
have ab1: zgcd(2*a, a+3*b) = 1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and z dvd 2*a and zab: z dvd a+3*b
  hence z dvd 2 ∨ z dvd a by (simp add: zprime-zdvd-zmult)
  moreover have zn2: ¬ z dvd 2
  proof (rule ccontr, simp)
    assume z2: z dvd 2
    hence z ≤ 2 by (simp only: zdvd-imp-le)
    with z have z = 2 by (unfold zprime-def, auto)
    with zab have ab2: 2 dvd a+3*b by simp
    moreover have 2 dvd 2*b by (rule zdvd-triv-left)
    ultimately have 2 dvd a+3*b- 2*b by (rule zdvd-zdiff)
    hence 2 dvd a+b by arith
    hence 2 dvd (a+b)*((a-b)*b*3) by (rule zdvd-zmult2)
    with ab have qEven: 2 dvd q by (simp only: mult-ac)
    from ab2 have 2 dvd (a+3*b)*((a- 3*b)*a) by (rule zdvd-zmult2)
    with ab have 2 dvd p by (simp only: mult-ac)
    with qEven have 2 dvd zgcd(p,q) by (simp add: zgcd-greatest-iff)
    with pq-relprime show False by auto
  qed
  ultimately have za: z dvd a by auto
  with zab have z dvd a+3*b-a by (simp only: zdvd-zdiff)
  hence z dvd 3*b by simp
  with z have z dvd 3 ∨ z dvd b by (simp only: zprime-zdvd-zmult-general)
  moreover
  { assume z dvd 3
    with z have z ≤ 3 by (auto simp add: zdvd-imp-le)
    moreover from zn2 have z≠2 by auto
    moreover from z have z > 1 by (simp add: zprime-def)
    ultimately have z=3 by auto
    with za have 3 dvd a by simp
    with ab have 3 dvd p by (auto simp add: zdvd-zmult2)
```

```
    with p3 have False by auto }
  ultimately have z dvd b by auto
  with za z ab-relprime show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
have ab2: zgcd(a+3*b, a- 3*b) = 1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and zab1: z dvd a+3*b and zab2: z dvd a- 3*b
  hence z dvd (a+3*b)+(a- 3*b) by (simp only: zdvd-zadd)
  hence z dvd 2*a by simp
  with zab1 z ab1 show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
have zgcd(a- 3*b, 2*a) = 1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and z2a: z dvd 2*a and zab: z dvd a- 3*b
  hence z dvd 2*a-(a- 3*b) by (simp only: zdvd-zdiff)
  moreover have 2*a-(a- 3*b) = a+3*b by simp
  ultimately have z dvd a+3*b by simp
  with z2a z ab1 show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
with abc ab1 ab2 triv3 have ∃ k l m. 2*a=k^3 ∧ a+3*b=l^3 ∧ a- 3*b=m^3
  by (simp only: int-triple-relprime-odd-power-divisors)
then obtain α β γ where albega:
  2*a = γ^3 ∧ a - 3*b = α^3 ∧ a+3*b = β^3 by auto
— show this is a (smaller) solution
hence α^3 + β^3 = γ^3 by auto
moreover have α*β*γ ≠ 0
proof (rule ccontr, safe)
  assume α * β * γ = 0
  with albega ab have p=0 by (auto simp add: power-0-left)
  with vwpq vwx show False by auto
qed
moreover have γ ∈ zEven
proof -
  have 2*a ∈ zEven by (simp add: zEven-def)
  with albega have γ^3 ∈ zEven by simp
  thus ?thesis by (simp only: power-preserves-even)
qed
moreover have zgcd(α, β)=1
proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and za: z dvd α and zb: z dvd β
  hence z dvd α * α^2 ∧ z dvd β * β^2 by (simp add: zdvd-zmult2)
  hence z dvd α^Suc 2 ∧ z dvd β^Suc 2 by (auto simp only: power-Suc)
  with albega have z dvd a- 3*b ∧ z dvd a+3*b by auto
  with ab2 z show False
    by (auto simp add: zgcd1-iff-no-common-primedivisor)
qed
moreover have nat|γ^3| < nat|x^3|
proof -
  let ?A = p^2 + 3*q^2
```

```
    from vwx vwpq have x^3 = 2*p*?A by auto
    also with ab have ... = 2*a*((a+3*b)*(a- 3*b)*?A) by auto
    also with albega have ... = γ^3 *((a+3*b)*(a- 3*b)*?A) by auto
    finally have eq: |x^3| = |γ^3| * |(a+3*b)*(a- 3*b)*?A|
      by (auto simp add: abs-mult)
    with x3pos have |(a+3*b)*(a- 3*b)*?A| > 0 by auto
    hence eqpos: |(a+3*b)*(a- 3*b)| > 0 by auto
    moreover have Ag1: |?A| > 1
    proof −
      have Aqf3: is-qfN ?A 3 by (auto simp add: is-qfN-def)
      moreover have triv3b: (3::int) ≥ 1 by simp
      ultimately have ?A ≥ 0 by (simp only: qfN-pos)
      hence ?A > 1 ∨ ?A = 0 ∨ ?A =1 by auto
      moreover
      { assume ?A = 0 with triv3b have p = 0 ∧ q = 0 by (rule qfN-zero)
        with vwpq vwx have False by auto }
      moreover
      { assume A1: ?A = 1
        have q=0
        proof (rule ccontr)
          assume q ≠ 0
          hence q^2 > 0 by (simp add: zero-less-power2)
          hence 3*q^2 > 1 by arith
          moreover have p^2 ≥ 0 by (rule zero-le-power2)
          ultimately have ?A > 1 by arith
          with A1 show False by simp
        qed
        with A1 have p21: p^2 = 1 by simp
        hence |p| = 1 by (rule power2-eq1-iff)
        with vwpq vwx A1 have |x^3| = 2 by auto
        hence False by (rule two-not-abs-cube) }
      ultimately show ?thesis by auto
    qed
    ultimately have
      |(a+3*b)*(a- 3*b)|*1 < |(a+3*b)*(a- 3*b)|*|?A|
      by (simp only: zmult-zless-mono2)
    with eqpos have |(a+3*b)*(a- 3*b)|*|?A| > 1 by arith
    hence |(a+3*b)*(a- 3*b)*?A| > 1 by (auto simp add: abs-mult)
    moreover have |γ^3| > 0
    proof −
      from eq have |γ^3| = 0 ⟹ |x^3|=0 by auto
      with x3pos show ?thesis by auto
    qed
    ultimately have |γ^3| * 1 < |γ^3| * |(a+3*b)*(a- 3*b)*?A|
      by (rule zmult-zless-mono2)
    with eq have |x^3| > |γ^3| by auto
    thus ?thesis by arith
  qed
  ultimately have ?thesis by auto }
moreover
— second case: p = 3r and hence x^3 = (18r)(q^2 + 3r^2) and these
— factors are coprime; hence both are cubes
```

```
{ assume p3: 3 dvd p
 then obtain r where r: p = 3*r by (auto simp add: dvd-def)
 moreover have 3 dvd 3*(3*r^2 + q^2) by (rule zdvd-triv-left)
 ultimately have pq3: 3 dvd p^2+3*q^2 by (simp add: power-mult-distrib)
 moreover from p3 have 3 dvd 2*p by (rule zdvd-zmult)
 ultimately have g3: 3 dvd ?g by (simp add: zgcd-greatest-iff)
 have qr-relprime: zgcd(q,r) = 1
 proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
  fix z assume z: zprime z and zq: z dvd q and z dvd r
  with r have z dvd p by (simp add: zdvd-zmult)
  with zq have z dvd p+q ∧ z dvd p−q by (simp add: zdvd-zadd zdvd-zdiff)
  with vw have z dvd zgcd(v, w) by (simp add: zgcd-greatest-iff)
  with vwx z show False by (auto simp add: zprime-def)
 qed
 have factors-relprime: zgcd(18*r, q^2 + 3*r^2) = 1
 proof −
  from g3 obtain k where k: ?g = 3*k by (auto simp add: dvd-def)
  have k = 1
  proof (rule ccontr)
   assume k ≠ 1
   with g1 k have k > 1 by auto
   then obtain h where h: zprime h ∧ h dvd k
    by (frule-tac a=k in zprime-factor-exists, blast)
   with k have hg: 3*h dvd ?g by (auto simp add: zdvd-zmult-mono)
   hence 3*h dvd p^2 + 3*q^2 and hp: 3*h dvd 2*p
    by (auto simp only: zgcd-greatest-iff)
   then obtain s where s: p^2 + 3*q^2 = (3*h)*s
    by (auto simp add: dvd-def)
   with r have rqh: 3*r^2+q^2 = h*s by (simp add: power-mult-distrib)
   from hp r have 3*h dvd 3*(2*r) by simp
   moreover have (3::int) ≠ 0 by simp
   ultimately have h dvd 2*r by (rule zdvd-mult-cancel)
   with h have h dvd 2 ∨ h dvd r by (simp only: zprime-zdvd-zmult-general)
   moreover have ¬ h dvd 2
   proof (rule ccontr, simp)
    assume h dvd 2
    with h have h=2 by (auto simp add: zdvd-not-zless zprime-def)
    with hg have 2*3 dvd ?g by auto
    hence 2 dvd ?g by (rule zdvd-zmultD2)
    with gOdd show False by simp
   qed
   ultimately have hr: h dvd r by simp
   then obtain t where r = h*t by (auto simp add: dvd-def)
   hence t: r^2 = h*(h*t^2) by (auto simp add: power2-eq-square)
   with rqh have h*s = h*(3*h*t^2) + q^2 by simp
   hence q^2 = h*(s − 3*h*t^2) by (simp add: zdiff-zmult-distrib2)
   hence h dvd q^2 by simp
   with h have h dvd q by (auto dest: zprime-zdvd-power)
   with hr have h dvd zgcd(q,r) by (simp add: zgcd-greatest-iff)
   with h qr-relprime show False by (unfold zprime-def, auto)
  qed
  with k r have 3 = zgcd(2*(3*r), (3*r)^2 + 3*q^2) by auto
```

```
  also have ... = zgcd(3*(2*r), 3*(3*r^2 + q^2))
    by (simp add: power-mult-distrib)
  also have ... = 3 * zgcd(2*r, 3*r^2 + q^2)
    by (simp only: zgcd-zmult-distrib2)
  finally have zgcd(2*r, 3*r^2 + q^2) = 1 by auto
  moreover have zgcd(3*3, 3*r^2 + q^2) = 1
  proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
    fix h::int assume h dvd 3*3 and h: zprime h and hrq: h dvd 3*r^2 + q^2
    hence h dvd 3 ∨ h dvd 3 by (simp only: zprime-zdvd-zmult-general)
    hence h3: h dvd 3 by simp
    have h ≤ 3
    proof (rule ccontr)
      assume ¬ h ≤ 3 hence h > 3 by simp
      with h3 show False by (auto simp add: zdvd-not-zless)
    qed
    with h have h = 2 ∨ h = 3 by (unfold zprime-def, auto)
    with h h3 have h = 3 ∨ (2::int) dvd 3 by auto
    hence h=3 by arith
    with hrq obtain s where 3*r^2+q^2 = 3*s by (auto simp add: dvd-def)
    hence q^2 = 3*(s- r^2) by auto
    hence 3 dvd q^2 and zprime 3 by (auto simp only: zdvd-triv-left zprime-3)
    hence 3 dvd q by (rule-tac p=3 in zprime-zdvd-power)
    with p3 have 3 dvd p+q ∧ 3 dvd p-q by (simp add: zdvd-zdiff zdvd-zadd)
    with vw have 3 dvd zgcd(v,w) by (simp add: zgcd-greatest-iff)
    with vwx show False by auto
  qed
  ultimately have zgcd((3*3)*(2*r), 3*r^2 + q^2) = 1
    by (simp only: zgcd-zmult-cancel)
  thus ?thesis by (auto simp add: mult-ac add-ac)
qed
moreover have rqx: (18*r)*(q^2 + 3*r^2) = x^3
proof -
  from vwx vwpq have x^3 = 2*p*(p^2 + 3*q^2) by auto
  also with r have ... = 2*(3*r)*(9*r^2 + 3*q^2)
    by (auto simp add: power2-eq-square)
  finally show ?thesis by auto
qed
moreover have triv3: 3 = nat 3 ∧ nat 3 > 1 ∧ 3 ∈ zOdd
  by (unfold zOdd-def, auto)
ultimately have ∃ c. 18*r = c^3
  by (simp only: int-relprime-odd-power-divisors)
then obtain c1 where c1: c1^3 = 3*(6*r) by auto
hence 3 dvd c1^3 and zprime 3 by (auto simp only: zdvd-triv-left zprime-3)
hence 3 dvd c1 by (rule-tac p=3 in zprime-zdvd-power)
with c1 obtain c where c: 3*c^3 = 2*r
  by (auto simp add: power-mult-distrib dvd-def)
from rqx factors-relprime have zgcd(q^2 + 3*r^2, 18*r) = 1
  and (q^2 + 3*r^2)*(18*r) = x^3 by (auto simp add: zgcd-commute mult-ac)
with triv3 have ∃ d. q^2 + 3*r^2 = d^3
  by (simp only: int-relprime-odd-power-divisors)
then obtain d where d: q^2 + 3*r^2 = d^3 by auto
have d ∈ zOdd
```

proof (*rule ccontr*)
assume $d \notin zOdd$ **hence** $d \in zEven$ **by** (*rule not-odd-impl-even*)
hence $d\hat{}3 \in zEven$ **by** (*simp only*: *power-preserves-even*)
hence *2 dvd* $d\hat{}3$ **by** (*simp add*: *zEven-def dvd-def*)
moreover have *2 dvd* $2*(9*r)$ **by** (*rule zdvd-triv-left*)
ultimately have *2 dvd* $zgcd(2*(9*r), d\hat{}3)$ **by** (*simp add*: *zgcd-greatest-iff*)
with *d factors-relprime* **show** *False* **by** *auto*
qed
with *d qr-relprime* **have** $zgcd(q,r)=1 \wedge q\hat{}2 + 3*r\hat{}2 = d\hat{}3 \wedge d \in zOdd$
by *simp*
hence *is-cube-form q r* **by** (*rule qf3-cube-impl-cube-form*)
then obtain *a b* **where** $q = a\hat{}3 - 9*a*b\hat{}2 \wedge r = 3*a\hat{}2*b - 3*b\hat{}3$
by (*unfold is-cube-form-def*, *auto*)
hence *ab*: $q = a*(a+3*b)*(a- 3*b) \wedge r = b*(a+b)*(a-b)*3$
by (*simp add*: *nat-number ring-simps*)
with *c* **have** *abc*: $(2*b)*(a+b)*(a-b) = c\hat{}3$ **by** *auto*
have *ab-relprime*: $zgcd(a,b)=1$
proof (*simp only*: *zgcd1-iff-no-common-primedivisor*, *clarify*)
fix *z* **assume** *z*: *zprime z* **and** *za*: *z dvd a* **and** *zb*: *z dvd b*
with *ab* **have** *z dvd q* $\wedge$ *z dvd r* **by** (*simp add*: *zdvd-zmult2*)
with *z qr-relprime* **show** *False*
by (*auto simp add*: *zgcd1-iff-no-common-primedivisor*)
qed
have *ab1*: $zgcd(2*b, a+b) = 1$
proof (*simp only*: *zgcd1-iff-no-common-primedivisor*, *clarify*)
fix *z* **assume** *z*: *zprime z* **and** *z dvd* $2*b$ **and** *zab*: *z dvd* $a+b$
hence *z dvd 2* $\vee$ *z dvd b* **by** (*simp add*: *zprime-zdvd-zmult*)
moreover
{ assume *z2*: *z dvd 2*
hence $z \leq 2$ **by** (*simp only*: *zdvd-imp-le*)
with *z* **have** $z = 2$ **by** (*unfold zprime-def*, *auto*)
with *zab* **have** *ab2*: *2 dvd* $a+b$ **by** *simp*
moreover have *2 dvd* $2*b$ **by** (*rule zdvd-triv-left*)
ultimately have *2 dvd* $a+b+2*b$ **by** (*rule zdvd-zadd*)
hence *2 dvd* $a+3*b$ **by** *arith*
hence *2 dvd* $(a+3*b)*((a- 3*b)*a)$ **by** (*rule zdvd-zmult2*)
with *ab* **have** *qEven*: *2 dvd q* **by** (*simp only*: *mult-ac*)
from *ab2* **have** *2 dvd* $(a+b)*((a-b)*3*b)$ **by** (*rule zdvd-zmult2*)
with *ab* **have** *2 dvd r* **by** (*simp only*: *mult-ac*)
with *qEven* **have** *2 dvd* $zgcd(q,r)$ **by** (*simp add*: *zgcd-greatest-iff*)
with *qr-relprime* **have** *False* **by** *auto* **}**
moreover
{ assume *zb*: *z dvd b*
with *zab* **have** *z dvd* $a+b-b$ **by** (*simp only*: *zdvd-zdiff*)
hence *z dvd a* **by** *simp*
with *zb ab-relprime z* **have** *False*
by (*auto simp add*: *zgcd1-iff-no-common-primedivisor*) **}**
ultimately show *False* **by** *auto*
qed
have *ab2*: $zgcd(a+b, a-b) = 1$
proof (*simp only*: *zgcd1-iff-no-common-primedivisor*, *clarify*)
fix *z* **assume** *z*: *zprime z* **and** *zab1*: *z dvd* $a+b$ **and** *zab2*: *z dvd* $a-b$

hence *z dvd* $(a+b)-(a-b)$ **by** (*simp only*: *zdvd-zdiff*)
hence *z dvd* $2*b$ **by** *simp*
with *zab1 z ab1* **show** *False*
by (*auto simp add*: *zgcd1-iff-no-common-primedivisor*)
qed
have $zgcd(a-b,\ 2*b) = 1$
proof (*simp only*: *zgcd1-iff-no-common-primedivisor*, *clarify*)
fix *z* **assume** *z*: *zprime z* **and** *z2b*: *z dvd* $2*b$ **and** *zab*: *z dvd* $a-b$
hence *z dvd* $a-b+2*b$ **by** (*simp only*: *zdvd-zadd*)
moreover have $a-b+2*b = a+b$ **by** *simp*
ultimately have *z dvd* $a+b$ **by** *simp*
with *z2b z ab1* **show** *False*
by (*auto simp add*: *zgcd1-iff-no-common-primedivisor*)
qed
with *abc ab1 ab2 triv3* **have** $\exists\ k\ l\ m.\ 2*b = k\hat{}3 \wedge a+b = l\hat{}3 \wedge a-b = m\hat{}3$
by (*simp only*: *int-triple-relprime-odd-power-divisors*)
then obtain $\alpha 1\ \beta\ \gamma$ **where** *a1*: $2*b = \gamma\hat{}3 \wedge a-b = \alpha 1\hat{}3 \wedge a+b = \beta\hat{}3$
by *auto*
then obtain α **where** $\alpha = -\alpha 1$ **by** *auto*
— show this is a (smaller) solution
with *triv3 a1* **have** *a2*: $\alpha\hat{}3 = b-a$ **by** (*auto simp only*: *neg-odd-power*)
with *a1* **have** $\alpha\hat{}3 + \beta\hat{}3 = \gamma\hat{}3$ **by** *auto*
moreover have $\alpha*\beta*\gamma \neq 0$
proof (*rule ccontr*, *safe*)
assume $\alpha * \beta * \gamma = 0$
with *a1 a2 ab* **have** *r=0* **by** (*auto simp add*: *power-0-left*)
with *r vwpq vwx* **show** *False* **by** *auto*
qed
moreover have $\gamma \in zEven$
proof −
have $2*b \in zEven$ **by** (*simp add*: *zEven-def*)
with *a1* **have** $\gamma\hat{}3 \in zEven$ **by** *simp*
thus *?thesis* **by** (*simp only*: *power-preserves-even*)
qed
moreover have $zgcd(\alpha,\ \beta)=1$
proof (*simp only*: *zgcd1-iff-no-common-primedivisor*, *clarify*)
fix *z* **assume** *z*: *zprime z* **and** *za*: *z dvd* α **and** *zb*: *z dvd* β
hence *z dvd* $\alpha * \alpha\hat{}2 \wedge z$ *dvd* $\beta * \beta\hat{}2$ **by** (*simp add*: *zdvd-zmult2*)
hence *z dvd* $\alpha\hat{}Suc\ 2 \wedge z$ *dvd* $\beta\hat{}Suc\ 2$ **by** (*auto simp only*: *power-Suc*)
with *a1 a2* **have** *z dvd* $b-a \wedge z$ *dvd* $a+b$ **by** *auto*
hence *z dvd* $-(b-a) \wedge z$ *dvd* $a+b$ **by** (*auto simp only*: *zdvd-zminus-iff*)
with *ab2 z* **show** *False*
by (*auto simp add*: *zgcd1-iff-no-common-primedivisor*)
qed
moreover have $nat|\gamma\hat{}3| < nat|x\hat{}3|$
proof −
let $?A = p\hat{}2 + 3*q\hat{}2$
from *vwx vwpq* **have** $x\hat{}3 = 2*p*?A$ **by** *auto*
also with *r* **have** $\ldots = 6*r*?A$ **by** *auto*
also with *ab* **have** $\ldots = 2*b*(9*(a+b)*(a-b)*?A)$ **by** *auto*
also with *a1* **have** $\ldots = \gamma\hat{}3 *(9*(a+b)*(a-b)*?A)$ **by** *auto*
finally have *eq*: $|x\hat{}3| = |\gamma\hat{}3| * |9*(a+b)*(a-b)*?A|$

```
      by (auto simp add: abs-mult)
    with x3pos have |9*(a+b)*(a-b)*?A| > 0 by auto
    hence |(a+b)*(a-b)*?A| ≥ 1 by arith
    hence |9*(a+b)*(a-b)*?A| > 1 by arith
    moreover have |γ^3| > 0
    proof -
      from eq have |γ^3| = 0 ⟹ |x^3|=0 by auto
      with x3pos show ?thesis by auto
    qed
    ultimately have |γ^3| * 1 < |γ^3| * |9*(a+b)*(a-b)*?A|
      by (rule zmult-zless-mono2)
    with eq have |x^3| > |γ^3| by auto
    thus ?thesis by arith
  qed
  ultimately have ?thesis by auto }
 ultimately show ?thesis by auto
qed
thus ∃ y. nat|y^3| < nat|x^3| ∧ ¬ ?Q y by auto
qed
```

The theorem. Puts equation in requested shape.

```
theorem fermat3:
 assumes ass: (x::int)^3 + y^3 = z^3
 shows x*y*z=0
proof (rule ccontr)
 let ?g = zgcd(x,y)
 let ?c = z div ?g
 assume xyz0: x*y*z≠0
 — divide out the g.c.d.
 hence x ≠ 0 ∨ y ≠ 0 by simp
 then obtain a b where ab: x = ?g*a ∧ y = ?g*b ∧ zgcd(a,b)=1
   by (frule-tac a=x in make-zrelprime, auto)
 moreover have abc: ?c*?g = z ∧ a^3 + b^3 = ?c^3 ∧ a*b*?c ≠ 0
 proof -
   from xyz0 have g0: ?g≠0 by (simp add: zgcd-def gcd-zero)
   have zgab: z^3 = ?g^3 * (a^3+b^3)
   proof -
     from ab and ass have z^3 = (?g*a)^3+(?g*b)^3 by simp
     thus ?thesis by (simp only: power-mult-distrib zadd-zmult-distrib2)
   qed
   have cgz: ?c * ?g = z
   proof -
     from zgab have ?g^3 dvd z^3 by simp
     hence ?g dvd z by (simp only: zpower-zdvd-mono)
     thus ?thesis by (simp only: mult-ac zdvd-mult-div-cancel)
   qed
   moreover have a^3 + b^3 = ?c^3
   proof -
     have ?c^3 * ?g^3 = (a^3+b^3)*?g^3
     proof -
       have ?c^3 * ?g^3 = (?c*?g)^3 by (simp only: power-mult-distrib)
       also with cgz have ... = z^3 by simp
```

```
    also with zgab have ... = ?g^3*(a^3+b^3) by simp
    finally show ?thesis by simp
  qed
  with g0 show ?thesis by auto
 qed
 moreover from ab and xyz0 and cgz have a*b*?c≠0 by auto
 ultimately show ?thesis by simp
qed
— make both sides even
have ∃ u v w. u^3 + v^3 = w^3 ∧ u*v*w≠0 ∧ w ∈ zEven ∧ zgcd(u,v)=1
proof −
 let ?Q u v w = u^3 + v^3 = w^3 ∧ u*v*w≠0 ∧ w ∈ zEven ∧ zgcd(u,v)=1
 have a ∈ zEven ∨ b ∈ zEven ∨ ?c ∈ zEven
 proof (rule ccontr)
  assume ¬(a ∈ zEven ∨ b ∈ zEven ∨ ?c ∈ zEven)
  hence aodd: a ∈ zOdd and b ∈ zOdd ∧ ?c ∈ zOdd
   by (auto simp add: odd-iff-not-even)
  hence ?c^3 − b^3 ∈ zEven by (simp only: power-preserves-odd odd-minus-odd)
  moreover from abc have ?c^3−b^3 = a^3 by simp
  ultimately have a^3 ∈ zEven by auto
  hence a ∈ zEven by (simp only: power-preserves-even)
  with aodd show False by (simp add: odd-iff-not-even)
 qed
 moreover
 { assume a ∈ zEven
  then obtain u v w where uvwabc: u = −b ∧ v = ?c ∧ w = a ∧ w ∈ zEven
   by auto
  moreover with abc have u*v*w≠0 by auto
  moreover have uvw: u^3+v^3=w^3
  proof −
   from uvwabc have u^3 + v^3 = (−1*b)^3 + ?c^3 by simp
   also have ... = (−1)^3*b^3 + ?c^3 by (simp only: power-mult-distrib)
   also have ... = − (b^3) + ?c^3 by (auto simp add: neg-one-odd-power)
   also with abc and uvwabc have ... = w^3 by auto
   finally show ?thesis by simp
  qed
  moreover have zgcd(u,v)=1
  proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
   fix h::int assume hu: h dvd u and h dvd v and h: zprime h
   with uvwabc have h dvd ?c*?c^2 by (simp only: zdvd-zmult2)
   with abc have h dvd a^3+b^3 by (simp only: cube-square)
   moreover from hu uvwabc have h dvd b*b^2 by (simp add: zdvd-zmult2)
   ultimately have h dvd a^3+b^3−b^3 by (simp only: cube-square zdvd-zdiff)
   with h hu uvwabc have h dvd a ∧ h dvd b by (auto dest: zprime-zdvd-power)
   with h ab show False by (auto simp add: zgcd1-iff-no-common-primedivisor)
  qed
  ultimately have ?Q u v w by simp
  hence ?thesis by auto }
 moreover
 { assume b ∈ zEven
  then obtain u v w where uvwabc: u = −a ∧ v = ?c ∧ w = b ∧ w ∈ zEven
   by auto
```

```
    moreover with abc have u*v*w≠0 by auto
    moreover have uvw: u^3+v^3=w^3
    proof −
      from uvwabc have u^3 + v^3 = (−1*a) ^3 + ?c^3 by simp
      also have ... = (−1) ^3*a^3 + ?c^3 by (simp only: power-mult-distrib)
      also have ... = − (a^3) + ?c^3 by (auto simp add: neg-one-odd-power)
      also with abc and uvwabc have ... = w^3 by auto
      finally show ?thesis by simp
    qed
    moreover have zgcd(u,v)=1
    proof (simp only: zgcd1-iff-no-common-primedivisor, clarify)
      fix h::int assume hu: h dvd u and h dvd v and h: zprime h
      with uvwabc have h dvd ?c*?c^2 by (simp only: zdvd-zmult2)
      with abc have h dvd a^3+b^3 by (simp only: cube-square)
      moreover from hu uvwabc have h dvd a*a^2 by (simp add: zdvd-zmult2)
      ultimately have h dvd a^3+b^3−a^3 by (simp only: cube-square zdvd-zdiff)
      with h hu uvwabc have h dvd a ∧ h dvd b by (auto dest: zprime-zdvd-power)
      with h ab show False by (auto simp add: zgcd1-iff-no-common-primedivisor)
    qed
    ultimately have ?Q u v w by simp
    hence ?thesis by auto }
  moreover
  { assume ?c ∈ zEven
    then obtain u v w where uvwabc: u = a ∧ v = b ∧ w = ?c ∧ w ∈ zEven
      by auto
    with abc ab have ?thesis by auto }
  ultimately show ?thesis by auto
 qed
 hence ∃ w. ∃ u v. u^3 + v^3 = w^3 ∧ u*v*w ≠ 0 ∧ w ∈ zEven ∧ zgcd(u,v)=1
  by auto
 — show contradiction using the earlier result
 thus False by (auto simp only: no-rewritten-fermat3)
qed

corollary fermat-mult3:
 assumes xyz: (x::int) ^n + y^n = z^n and n: 3 dvd n
 shows x*y*z=0
proof −
 from n obtain m where n = m*3 by (auto simp only: mult-ac dvd-def)
 with xyz have (x^m) ^3 + (y^m) ^3 = (z^m) ^3 by (simp only: power-mult)
 hence (x^m)*(y^m)*(z^m) = 0 by (rule fermat3)
 thus ?thesis by auto
qed

end
```

Sums of two and four squares

Roelof Oosterhuis
University of Groningen

August 24, 2007

Abstract

This document gives the formal proofs, verified by the proof assistant 'Isabelle'[1], of the following results about the sums of two and four squares:

1. Any prime number $p \equiv 1 \bmod 4$ can be written as the sum of two squares.
2. (Lagrange) Any natural number can be written as the sum of four squares.

The proofs are largely based on chapter II and III of André Weil, *Number theory: an approach through history; From Hammurapi to Legendre.* Boston (etc.): Birkhäuser, 1983.
We remind the reader that the results have been formalised before in the proof assistant HOL-light[2]. A more complete study of the sum of two squares, including the first result, has been formalised in 'Coq'[3]. Moreover it should be mentioned that the results can be found as numbers 20 and 19 on the list of 'top 100 mathematical theorems'.[4]
This research is part of a M.Sc. thesis under supervision of Jaap Top and Wim H. Hesselink (RU Groningen).
More information: http://www.roelofoosterhuis.nl/MScthesis.pdf

[1] See http://isabelle.in.tum.de/
[2] See http://www.cl.cam.ac.uk/~jrh13/hol-light/
[3] See http://coq.inria.fr/contribs/SumOfTwoSquare.html
[4] See http://www.cs.ru.nl/~freek/100/

Contents

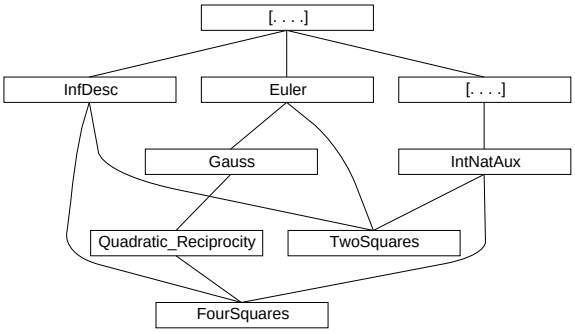

Figure 1: The depence on existing files in the Isabelle library.

1 Sums of two squares

theory *TwoSquares*
imports *IntNatAux InfDesc*
~~/*src*/*HOL*/*NumberTheory*/*Euler*
begin

Show that $(\frac{-1}{p}) = +1$ for primes $p \equiv 1 \bmod 4$.

constdefs
sum2sq :: *int* × *int* ⇒ *int*
sum2sq == λ(*a*,*b*). *a*^2+*b*^2

is-sum2sq :: *int* ⇒ *bool*
is-sum2sq x == ∃ *a b*. *sum2sq*(*a*,*b*) = *x*

lemma *mult-sum2sq*: *sum2sq*(*a*,*b*) ∗ *sum2sq*(*p*,*q*) =
sum2sq(*a*∗*p*+*b*∗*q*, *a*∗*q*−*b*∗*p*)
by (*unfold sum2sq-def*, *simp add*: *nat-number ring-simps*)

lemma *is-mult-sum2sq*: *is-sum2sq x* ⟹ *is-sum2sq y* ⟹ *is-sum2sq* (*x*∗*y*)
by (*unfold is-sum2sq-def*, *auto simp only*: *mult-sum2sq*, *blast*)

lemma *Legendre-1mod4*: *zprime* (*4*∗*m*+*1*) ⟹ (*Legendre* (−*1*) (*4*∗*m*+*1*)) = *1*
proof −
let *?p* = *4*∗*m*+*1*
let *?L* = *Legendre* (−*1*) *?p*
assume *p*: *zprime ?p*
have *m* ≥ *1*
proof (*rule ccontr*)
assume ¬ *m* ≥ *1* **hence** *m* ≤ *0* **by** *auto*
hence *?p* ≤ *1* **by** *auto*
with *p* **show** *False* **by** (*simp add*: *zprime-def*)
qed
hence *p2*: *?p* > *2* **by** *simp*
with *p* **have** [*?L* = (−*1*)^*nat*((*?p* − *1*) *div* *2*)] (*mod* *?p*)
by (*simp only*: *Euler-Criterion*)
hence [*?L* = (−*1*)^(*2*∗*nat m*)] (*mod* *?p*) **by** (*auto simp add*: *nat-mult-distrib*)
hence [*1* = *?L*] (*mod* *?p*) **by** (*auto simp add*: *power-mult power2-minus zcong-sym*)
hence *?p dvd* *1*−*?L* **by** (*simp add*: *zcong-def*)
moreover have *?L*=*1* ∨ *?L*=*0* ∨ *?L*=−*1* **by** (*simp add*: *Legendre-def*)
ultimately have *?L* = *1* ∨ *?p dvd* *1* ∨ *?p dvd* *2* **by** *auto*
moreover
{ assume *?p dvd* *1* ∨ *?p dvd* *2*
with *p2* **have** *False* **by** (*auto simp add*: *zdvd-not-zless*) **}**
ultimately show *?thesis* **by** *auto*
qed

Use this to prove that such primes can be written as the sum of two squares.

lemma *qf1-prime-exists*: *zprime* (*4*∗*m*+*1*) ⟹ ∃ *x y*. *x*^2 + *y*^2 = *4*∗*m*+*1*
proof −
let *?p* = *4*∗*m*+*1*

assume *p*: *zprime ?p*
hence *Legendre* (−*1*) *?p* = *1* **by** (*rule Legendre-1mod4*)
moreover
{ assume ¬ *QuadRes ?p* (−*1*)
hence *Legendre* (−*1*) *?p* ≠ *1* **by** (*unfold Legendre-def*, *auto*) **}**
ultimately have *QuadRes ?p* (−*1*) **by** *auto*
then obtain *s1* **where** *s1*: [*s1^2* = −*1*] (*mod ?p*) **by** (*auto simp add*: *QuadRes-def*)
from *p* **have** *p0*: *?p* > *0* **by** (*simp add*: *zprime-def*)
hence ∃! *s*. *0* ≤ *s* ∧ *s* < *?p* ∧ [*s1* = *s*] (*mod ?p*)
by (*simp only*: *zcong-zless-unique*)
then obtain *s* **where** *s0p*: *0* ≤ *s* ∧ *s* < *?p* ∧ [*s1* = *s*] (*mod ?p*)
by *auto*
hence [*s^2* = *s1^2*] (*mod ?p*) **by** (*simp only*: *zcong-sym power2-eq-square zcong-zmult*)
with *s1* **have** *s*: [*s^2* = −*1*] (*mod ?p*) **by** (*auto dest*: *zcong-trans*)
hence *?p dvd s^2* − (−*1*::*int*) **by** (*unfold zcong-def*, *simp*)
moreover have *s^2* −(−*1*::*int*) = *s^2* + *1* **by** *arith*
ultimately have *?p dvd s^2* + *1* **by** *simp*
then obtain *t* **where** *t*: *s^2* + *1* = *?p*∗*t* **by** (*auto simp add*: *dvd-def*)
hence *sum2sq*(*s*,*1*) = *?p*∗*t* **by** (*simp add*: *sum2sq-def*)
hence *qf1pt*: *is-sum2sq* (*?p*∗*t*) **by** (*auto simp add*: *is-sum2sq-def*)
have *t-l-p*: *t* < *?p*
proof (*rule ccontr*)
assume ¬ *t* < *?p* **hence** *t* > *?p* − *1* **by** *simp*
with *p0* **have** *?p*∗(*?p* − *1*) < *?p*∗*t* **by** (*simp only*: *zmult-zless-mono2*)
also with *t* **have** ... = *s^2* + *1* **by** *simp*
also have ... ≤ *?p*∗(*?p* − *1*) − *?p* + *2*
proof −
from *s0p* **have** *s* ≤ *?p* − *1* **by** (*auto simp add*: *less-int-def*)
with *s0p* **have** *s^2* ≤ (*?p* − *1*)^*2* **by** (*simp only*: *power-mono*)
also have ... = *?p*∗(*?p* − *1*) − *1*∗(*?p* − *1*)
by (*simp only*: *power2-eq-square zdiff-zmult-distrib*)
finally show *?thesis* **by** *auto*
qed
finally have *?p* < *2* **by** *arith*
with *p* **show** *False* **by** (*unfold zprime-def*, *auto*)
qed
have *tpos*: *t* ≥ *1*
proof (*rule ccontr*)
assume ¬ *t* ≥ *1* **hence** *t* < *1* **by** *auto*
moreover
{ assume *t* = *0* **with** *t* **have** *s^2* + *1* = *0* **by** *simp* **}**
moreover
{ assume *t* < *0*
with *p0* **have** *?p*∗*t* < *?p*∗*0* **by** (*simp only*: *zmult-zless-mono2*)
with *t* **have** *s^2* + *1* < *0* **by** *auto* **}**
moreover have *s^2* ≥ *0* **by** (*simp only*: *zero-le-power2*)
ultimately show *False* **by** (*auto simp add*: *less-int-def*)
qed
moreover
{ assume *t* = *1*
with *qf1pt* **have** *is-sum2sq ?p* **by** *auto*
hence *?thesis* **by** (*unfold is-sum2sq-def sum2sq-def*, *auto*) **}**

```
moreover
{ assume t1: t > 1
 then obtain tn where tn: tn = (nat t) − 1 and tn0: tn > 0 by auto
 have is-sum2sq (?p*(1+ int 0)) (is ?Q 0)
  — So, Q n = there exist x, y such that x^2 + y^2 = (p * (1 + int(n)))
 proof (rule ccontr)
  assume nQ1: ¬ ?Q 0
  have (1+int tn) < ?p ⟹ ¬ ?Q tn
  proof (induct tn rule: infinite-descent)
   case 0
   from nQ1 show 1+int 0 < ?p ⟹ ¬ ?Q 0 by simp
  next
   case (smaller n)
   hence n0: n > 0 and IH: 1+int n < ?p ∧ ?Q n by auto
   then obtain x y where xy: x^2 + y^2 = ?p*(1+int n)
    by (unfold is-sum2sq-def sum2sq-def, auto)
   let ?n1 = (1+int n)
   from n0 have n1pos: ?n1 > 0 by simp
   then obtain r v where rv: v = x−r*?n1 ∧ 2*|v| ≤ ?n1
    by (frule-tac x=?n1 in best-division-abs, auto)
   from n1pos obtain s w where sw: w = y−s*?n1 ∧ 2*|w| ≤ ?n1
    by (frule-tac x=?n1 in best-division-abs, auto)
   let ?C = v^2 + w^2
   have ?n1 dvd ?C
   proof
    from rv sw have ?C = (x−r*?n1)^2 + (y−s*?n1)^2 by simp
    also have ... =
     x^2 + y^2 − 2*x*(r*?n1) − 2*y*(s*?n1) + (r*?n1)^2 + (s*?n1)^2
     by (simp only: zdiff-power2)
    also with xy have ... =
     ?n1*?p − ?n1*(2*x*r) − ?n1*(2*y*s) + ?n1^2*r^2 + ?n1^2*s^2
     by (simp only: mult-ac power-mult-distrib)
    finally show ?C = ?n1*(?p − 2*x*r − 2*y*s + ?n1*(r^2 + s^2))
     by (simp only: power-mult-distrib zadd-zmult-distrib2 mult-ac
    zdiff-zmult-distrib zdiff-zmult-distrib2 power2-eq-square)
   qed
   then obtain m1 where m1: ?C = ?n1*m1 by (auto simp add: dvd-def)
   have mn: m1 < ?n1
   proof (rule ccontr)
    assume ¬ m1 < ?n1 hence ?n1−m1 ≤ 0 by simp
    hence 4*?n1 − 4*m1 ≤ 0 by simp
    with n1pos have 2*?n1 − 4*m1 < 0 by simp
    moreover from n1pos have ?n1 > 0 by simp
  ultimately have ?n1*(2*?n1 − 4*m1) < ?n1*0 by (simp only: zmult-zless-mono2)
    hence contr: ?n1*(2*?n1− 4*m1) < 0 by simp
    have hlp: 2*|v| ≥ 0 ∧ 2*|w| ≥ 0 by simp
    from m1 have 4*?n1*m1 = 4*v^2 + 4*w^2 by arith
    also have ... = (2*|v|)^2 + (2*|w|)^2
     by (auto simp add: power2-abs power-mult-distrib)
    also from rv hlp have ... ≤ ?n1^2 + (2*|w|)^2
     by (auto simp add: power-mono)
    also from sw hlp have ... ≤ ?n1^2 + ?n1^2
```

```
    by (auto simp add: power-mono)
  finally have ?n1*m1*4 ≤ ?n1*?n1*2
    by (simp add: power2-eq-square mult-ac)
  hence ?n1*(2*?n1- 4*m1) ≥ 0
    by (auto simp add: zdiff-zmult-distrib2 mult-ac)
  hence ?n1*(2*?n1- 4*m1) > -1 by auto
  with contr show False by auto
qed
have (r*v + s*w + m1) ^2 + (r*w - s*v) ^2 = ?p*m1
proof -
  from m1 xy have (?p*?n1)*?C = (x^2+y^2)*(v^2+w^2) by simp
  also have ... = (x*v + y*w) ^2 + (x*w - y*v) ^2
    by (simp add: nat-number ring-simps)
  also with rv sw have ... =
    ((r*?n1+v)*v + (s*?n1+w)*w) ^2 + ((r*?n1+v)*w - (s*?n1+w)*v) ^2
    by simp
  also have ... =
     (?n1*(r*v) + ?n1*(s*w) + (v^2+w^2)) ^2 + (?n1*(r*w) - ?n1*(s*v)) ^2
    by (simp add: nat-number ring-simps)
  also from m1 have ... =
     (?n1*(r*v) + ?n1*(s*w) + ?n1*m1) ^2 + (?n1*(r*w) - ?n1*(s*v)) ^2
    by simp
  finally have
    (?p*?n1)*?C = ?n1^2*(r*v + s*w + m1) ^2 + ?n1^2*(r*w - s*v) ^2
    by (simp add: nat-number ring-simps)
  with m1 have
    ?n1^2*(?p*m1) = ?n1^2*((r*v + s*w + m1) ^2 + (r*w - s*v) ^2)
    by (simp only: mult-ac power2-eq-square, simp add: zadd-zmult-distrib2)
  hence ?n1^2*(?p*m1 - (r*v+s*w+m1) ^2 - (r*w-s*v) ^2) = 0
    by (auto simp add: zadd-zmult-distrib2 zdiff-zmult-distrib2)
  moreover from n1pos have ?n1^2 ≠ 0 by (simp add: power2-eq-square)
  ultimately show ?thesis by simp
qed
hence qf1pm1: is-sum2sq (?p*m1) by (unfold is-sum2sq-def sum2sq-def, auto)
have m1pos: m1 > 0
proof -
  { assume v^2 + w^2 = 0
    moreover
    { assume v ≠ 0
      hence v^2 > 0 by (simp add: zero-less-power2)
      moreover have w^2 ≥ 0 by (rule zero-le-power2)
      ultimately have v^2 + w^2 > 0 by arith }
    moreover
    { assume w ≠ 0
      hence w^2 > 0 by (simp add: zero-less-power2)
      moreover have v^2 ≥ 0 by (rule zero-le-power2)
      ultimately have v^2 + w^2 > 0 by arith }
    ultimately have v = 0 ∧ w = 0 by auto
    with rv sw have ?n1 dvd x ∧ ?n1 dvd y by (unfold dvd-def, auto)
    hence ?n1^2 dvd x^2 ∧ ?n1^2 dvd y^2 by (simp add: zpower-zdvd-mono)
    hence ?n1^2 dvd x^2 + y^2 by (simp only: zdvd-zadd)
    with xy have ?n1*?n1 dvd ?n1*?p
```

```
      by (simp only: power2-eq-square mult-ac)
     moreover from n1pos have ?n1 ≠ 0 by simp
     ultimately have ?n1 dvd ?p by (rule zdvd-mult-cancel)
     with n1pos have ?n1 ≥ 0 ∧ ?n1 dvd ?p by simp
     with p have ?n1 = 1 ∨ ?n1 = ?p by (unfold zprime-def, blast)
     with IH have ?Q 0 by auto
     with nQ1 have False by simp }
   moreover
   { assume v^2 + 1*w^2 ≠ 0
     moreover have v^2 + w^2 ≥ 0
     proof −
       have v^2 ≥ 0 ∧ w^2 ≥ 0 by (auto simp only: zero-le-power2)
       thus ?thesis by arith
     qed
     ultimately have vwpos: v^2 + w^2 > 0 by simp
     with m1 have m1 ≠ 0 by auto
     moreover have m1 ≥ 0
     proof (rule ccontr)
       assume ¬ m1 ≥ 0 hence m1 < 0 by simp
       with n1pos have ?n1*m1 < ?n1*0 by (simp only: zmult-zless-mono2)
       with m1 vwpos show False by simp
     qed
     ultimately have ?thesis m1 > 0 by auto }
   ultimately show ?thesis by auto
 qed
 hence 1 + int((nat m1) − 1) = m1 by arith
 with qf1pm1 have Qm1: ?Q ((nat m1) − 1) by auto
 then obtain m where tmp: m = (nat m1) − 1 ∧ ?Q m by auto
 moreover have m<n
 proof −
   from tmp mn m1pos have int m < int n by arith
   thus ?thesis by arith
 qed
 moreover with IH have 1 + int m < ?p by auto
 ultimately show ∃ m. m<n ∧ ¬ (1 + int m < ?p ⟶ ¬ ?Q m) by auto
qed
moreover from tn tpos t-l-p have 1 + int tn < ?p ∧ tn = nat t− 1
 by arith
ultimately have ¬ ?Q ((nat t) − 1) by simp
moreover from tpos have 1 + int ((nat t) − 1) = t by arith
ultimately have ¬ is-sum2sq (?p*t) by auto
with qf1pt show False by simp
qed
hence ?thesis by (unfold is-sum2sq-def sum2sq-def, auto) }
ultimately show ?thesis by (auto simp add: less-int-def)
qed

end
```

2 Lagrange's four-square theorem

theory *FourSquares*
imports *IntNatAux InfDesc*
~~/*src*/*HOL*/*NumberTheory*/*Quadratic-Reciprocity*
begin

Shows that all nonnegative integers can be written as the sum of four squares. The proof consists of the following steps:

- For every prime $p = 2n + 1$ the two sets of residue classes

  $$\{x^2 \bmod p \mid 0 \leq x \leq n\} \text{ and } \{-1 - y^2 \bmod p \mid 0 \leq y \leq n\}$$

  both contain $n + 1$ different elements and therefore they must have at least one element in common.

- Hence there exist x, y such that $x^2 + y^2 + 1^2 + 0^2$ is a multiple of p.

- The next step is to show, by an infinite descent, that p itself can be written as the sum of four squares.

- Finally, using the multiplicity of this form, the same holds for all positive numbers.

constdefs
sum4sq :: *int* × *int* × *int* × *int* ⇒ *int*
sum4sq == λ(*a*,*b*,*c*,*d*). *a*^2+*b*^2+*c*^2+*d*^2

is-sum4sq :: *int* ⇒ *bool*
is-sum4sq x == ∃ *a b c d*. *sum4sq*(*a*,*b*,*c*,*d*) = *x*

lemma *mult-sum4sq*: *sum4sq*(*a*,*b*,*c*,*d*) ∗ *sum4sq*(*p*,*q*,*r*,*s*) =
sum4sq(*a*∗*p*+*b*∗*q*+*c*∗*r*+*d*∗*s*, *a*∗*q*−*b*∗*p*−*c*∗*s*+*d*∗*r*,
a∗*r*+*b*∗*s*−*c*∗*p*−*d*∗*q*, *a*∗*s*−*b*∗*r*+*c*∗*q*−*d*∗*p*)
by (*unfold sum4sq-def*, *simp add*: *nat-number ring-simps*)

lemma *is-mult-sum4sq*: *is-sum4sq x* ⟹ *is-sum4sq y* ⟹ *is-sum4sq* (*x*∗*y*)
by (*unfold is-sum4sq-def*, *auto simp only*: *mult-sum4sq*, *blast*)

lemma *mult-oddprime-is-sum4sq*: ⟦ *zprime p*; *p* ∈ *zOdd* ⟧ ⟹
∃ *t*. 0<*t* ∧ *t*<*p* ∧ *is-sum4sq* (*p*∗*t*)
proof −
assume *p1*: *zprime p*
hence *p0*: *p* > *1* **by** (*simp add*: *zprime-def*)
assume *p2*: *p* ∈ *zOdd*
then obtain *n* **where** *n*: *p* = *2*∗*n*+*1* **by** (*auto simp add*: *zOdd-def*)
with *p1* **have** *n0*: *n* > *0* **by** (*auto simp add*: *zprime-def*)
let *?C* = {*y*. *0* ≤ *y* ∧ *y* < *p*}
let *?D* = {*y*. *0* ≤ *y* ∧ *y* ≤ *n*}
let *?f* = %*x*. *x*^2 *mod p*
let *?g* = %*x*. (−*1*−*x*^2) *mod p*

let *?A = ?f ‘ ?D*
let *?B = ?g ‘ ?D*
have *finC*: *finite ?C* **by** (*rule bdd-int-set-l-finite*)
have *finD*: *finite ?D* **by** (*rule bdd-int-set-le-finite*)
from *p0* **have** *AsubC*: *?A ⊆ ?C* **and** *BsubC*: *?B ⊆ ?C*
by (*auto simp add*: *pos-mod-conj*)
with *finC* **have** *finA*: *finite ?A* **and** *finB*: *finite ?B*
by (*auto simp add*: *finite-subset*)
from *AsubC BsubC* **have** *AunBsubC*: *?A ∪ ?B ⊆ ?C* **by** (*rule Un-least*)
from *p0* **have** *cardC*: *card ?C = nat p* **by** (*simp only*: *card-bdd-int-set-l*)
from *n0* **have** *cardD*: *card ?D = 1+ nat n* **by** (*simp only*: *card-bdd-int-set-le*)
have *cardA*: *card ?A = card ?D*
proof −
have *inj-on ?f ?D*
proof (*unfold inj-on-def*, *auto*)
fix *x* **fix** *y*
assume *x0*: *0 ≤ x* **and** *xn*: *x ≤ n* **and** *y0*: *0 ≤ y* **and** *yn*: *y ≤ n*
and *xyp*: *x^2 mod p = y^2 mod p*
with *p0* **have** [*x^2 = y^2*] (*mod p*) **by** (*simp only*: *zcong-zmod-eq*)
hence *p dvd x^2−y^2* **by** (*simp only*: *zcong-def*)
hence *p dvd* (*x+y*)∗(*x−y*) **by** (*simp only*: *zspecial-product*)
with *p1* **have** *p dvd x+y ∨ p dvd x−y* **by** (*simp only*: *zprime-zdvd-zmult-general*)
moreover
{ assume *p dvd x+y*
moreover from *xn yn n* **have** *x+y < p* **by** *auto*
ultimately have ¬ *x+y > 0* **by** (*auto simp add*: *zdvd-not-zless*)
with *x0 y0* **have** *x = y* **by** *auto* **}** — both are zero
moreover
{ assume *ass*: *p dvd x−y*
have *x = y*
proof (*rule ccontr*, *case-tac x−y ≥ 0*, *auto*)
assume *x−y ≥ 0* **and** *x ≠ y* **hence** *x−y > 0* **by** *auto*
with *ass* **have** ¬ *x−y < p* **by** (*auto simp add*: *zdvd-not-zless*)
with *xn y0 n p0* **show** *False* **by** *auto*
next
assume ¬ *0 ≤ x−y* **hence** *y−x > 0* **by** *auto*
moreover from *x0 yn n p0* **have** *y−x < p* **by** *auto*
ultimately have ¬ *p dvd y−x* **by** (*auto simp add*: *zdvd-not-zless*)
moreover from *ass* **have** *p dvd* −(*x−y*) **by** (*simp only*: *zdvd-zminus-iff*)
ultimately show *False* **by** *auto*
qed **}**
ultimately show *x=y* **by** *auto*
qed
with *finD* **show** *?thesis* **by** (*simp only*: *inj-on-iff-eq-card*)
qed
have *cardB*: *card ?B = card ?D*
proof −
have *inj-on ?g ?D*
proof (*unfold inj-on-def*, *auto*)
fix *x* **fix** *y*
assume *x0*: *0 ≤ x* **and** *xn*: *x ≤ n* **and** *y0*: *0 ≤ y* **and** *yn*: *y ≤ n*
and *xyp*: (*−1−x^2*) *mod p* = (*−1−y^2*) *mod p*

```
    with p0 have [−1−y^2 = −1−x^2] (mod p) by (simp only: zcong-zmod-eq)
    hence p dvd (−1−y^2) − (−1−x^2) by (simp only: zcong-def)
    moreover have −1−y^2 − (−1−x^2) = x^2 − y^2 by arith
    ultimately have p dvd x^2−y^2 by simp
    hence p dvd (x+y)*(x−y) by (simp only: zspecial-product)
    with p1 have p dvd x+y ∨ p dvd x−y by (simp only: zprime-zdvd-zmult-general)
    moreover
    { assume p dvd x+y
      moreover from xn yn n have x+y < p by auto
      ultimately have ¬ x+y > 0 by (auto simp add: zdvd-not-zless)
      with x0 y0 have x = y by auto } — both are zero
    moreover
    { assume ass: p dvd x−y
      have x = y
      proof (rule ccontr, case-tac x−y ≥ 0, auto)
        assume x−y ≥ 0 and x ≠ y hence x−y > 0 by auto
        with ass have ¬ x−y < p by (auto simp add: zdvd-not-zless)
        with xn y0 n p0 show False by auto
      next
        assume ¬ 0 ≤ x−y hence y−x > 0 by auto
        moreover from x0 yn n p0 have y−x < p by auto
        ultimately have ¬ p dvd y−x by (auto simp add: zdvd-not-zless)
        moreover from ass have p dvd −(x−y) by (simp only: zdvd-zminus-iff)
        ultimately show False by auto
      qed }
    ultimately show x=y by auto
  qed
  with finD show ?thesis by (simp only: inj-on-iff-eq-card)
qed
have ?A ∩ ?B ≠ {}
proof (rule ccontr, auto)
  assume ABdisj: ?A ∩ ?B = {}
  from cardA cardB cardD have 2 + 2*(nat n) = card ?A + card ?B by auto
  also with finA finB ABdisj have ... = card (?A ∪ ?B)
    by (simp only: card-Un-disjoint)
  also with finC AunBsubC have ... ≤ card ?C by (simp only: card-mono)
  also with cardC have ... = nat p by simp
  finally have 2 + 2*(nat n) ≤ nat p by simp
  with n show False by arith
qed
then obtain z where z ∈ ?A ∧ z ∈ ?B by auto
then obtain x y where xy: x ∈ ?D ∧ y ∈ ?D ∧ z = x^2 mod p ∧
  z = (−1−y^2) mod p by auto
with p0 have [x^2=−1−y^2](mod p) by (simp add: zcong-zmod-eq)
hence p dvd x^2−(−1−y^2) by (simp only: zcong-def)
moreover have x^2−(−1−y^2)=x^2+y^2+1 by arith
ultimately have p dvd sum4sq(x,y,1,0) by (auto simp add: sum4sq-def)
then obtain t where t: sum4sq(x,y,1,0) = p*t by (auto simp add: dvd-def)
hence is-sum4sq (p*t) by (unfold is-sum4sq-def, auto)
moreover have t > 0 ∧ t < p
proof
  have x^2 ≥ 0 ∧ y^2 ≥ 0 by (simp add: zero-le-power2)
```

```
    hence x^2+y^2+1 > 0 by arith
    with t have p*t > 0 by (unfold sum4sq-def, auto)
    moreover
    { assume t < 0 with p0 have p*t < p*0 by (simp only: zmult-zless-mono2)
      hence p*t < 0 by simp }
    moreover
    { assume t = 0 hence p*t = 0 by simp }
    ultimately have ¬ t < 0 ∧ t ≠ 0 by auto
    thus t > 0 by simp
    from xy have x^2 ≤ n^2 ∧ y^2 ≤ n^2 by (auto simp add: power-mono)
    hence x^2+y^2+1 ≤ 2*n^2 + 1 by auto
    with t have contr: p*t ≤ 2*n^2+1 by (simp add: sum4sq-def)
    moreover
    { assume t > n+1
      with p0 have p*(n+1) < p*t by (simp only: zmult-zless-mono2)
      with n have p*t > (2*n+1)*n + (2*n+1)*1 by (simp only: zadd-zmult-distrib2)
      hence p*t > 2*n*n + n + 2*n + 1 by (simp only: zadd-zmult-distrib zmult-1)
      with n0 have p*t > 2*n^2 + 1 by (simp add: power2-eq-square) }
    ultimately have ¬ t > n+1 by auto
    with n0 n show t < p by auto
  qed
  ultimately show ?thesis by blast
qed

lemma zprime-is-sum4sq: zprime p ⟹ is-sum4sq p
proof (cases)
  assume p2: p=2
  hence p = sum4sq(1,1,0,0) by (auto simp add: sum4sq-def)
  thus ?thesis by (auto simp add: is-sum4sq-def)
next
  assume ¬ p =2 and prp: zprime p
  hence 2 < p by (simp add: zprime-def)
  with prp have p ∈ zOdd by (simp only: zprime-zOdd-eq-grt-2)
  with prp have ∃ t. 0<t ∧ t<p ∧ is-sum4sq (p*t)
    by (rule mult-oddprime-is-sum4sq)
  then obtain a b c d t where pt-sol: 0<t ∧ t<p ∧ sum4sq(a,b,c,d)=p*t
    by (unfold is-sum4sq-def, blast)
  hence Qt: 0<t ∧ t<p ∧ (∃ a1 a2 a3 a4. sum4sq(a1,a2,a3,a4)=p*t)
    (is ?Q t) by blast
  have ?Q 1
  proof (rule ccontr)
    assume nQ1: ¬ ?Q 1
    have ¬ ?Q t
    proof (rule-tac x=t and V=λx. (nat x)- 1 in val-infinite-descent, clarify)
      fix x a b c d
      assume nat x - 1 = 0 and x > 0 and s: sum4sq(a,b,c,d)=p*x and x < p
      moreover hence x = 1 by arith
      ultimately have ?Q 1 by auto
      with nQ1 show False by auto
    next
      fix x
      assume 0 < nat x - 1 and ¬ ¬ ?Q x
```

```
then obtain a1 a2 a3 a4 where ass: 1<x ∧ x<p ∧ sum4sq(a1,a2,a3,a4)=p*x
  by auto
have ∃ y. nat y − 1 < nat x − 1 ∧ ?Q y
proof (cases)
  assume evx: x ∈ zEven
  hence x*p ∈ zEven by (rule even-times-either)
  with ass have ev1234: a1^2+a2^2 + a3^2+a4^2 ∈ zEven
    by (auto simp add: sum4sq-def mult-ac)
  have ∃ b1 b2 b3 b4. sum4sq(b1,b2,b3,b4)=p*x ∧
    b1+b2 ∈ zEven ∧ b3+b4 ∈ zEven
  proof (cases)
    assume ev12: a1^2+a2^2 ∈ zEven
    moreover have 2*a1*a2 ∈ zEven by (auto simp add: zEven-def)
    ultimately have a1^2+a2^2+2*a1*a2 ∈ zEven by (rule even-plus-even)
    hence (a1+a2)^2 ∈ zEven by (auto simp add: zadd-power2 add-ac)
    hence tmp: a1+a2 ∈ zEven by (auto simp add: power-preserves-even)
    from ev12 ev1234 have a1^2+a2^2+a3^2+a4^2−(a1^2+a2^2) ∈ zEven
      by (simp only: even-minus-even)
    hence a3^2+a4^2 ∈ zEven by auto
    moreover have 2*a3*a4 ∈ zEven by (auto simp add: zEven-def)
    ultimately have a3^2+a4^2+2*a3*a4 ∈ zEven by (rule even-plus-even)
    hence (a3+a4)^2 ∈ zEven by (auto simp add: zadd-power2 add-ac)
    hence a3+a4 ∈ zEven by (auto simp add: power-preserves-even)
    with tmp ass show ?thesis by blast
  next
    assume ¬ a1^2+a2^2 ∈ zEven
    hence odd12: a1^2+a2^2 ∈ zOdd by (simp add: odd-iff-not-even)
    with ev1234 have a1^2+a2^2+a3^2+a4^2−(a1^2+a2^2) ∈ zOdd
      by (simp only: even-minus-odd)
    hence odd34: a3^2+a4^2 ∈ zOdd by auto
    show ?thesis
    proof (cases)
      assume ev1: a1^2 ∈ zEven
      with odd12 have odd2: a2^2 ∈ zOdd by (rule even-plus-odd-prop2)
      show ?thesis
      proof (cases)
        assume ev3: a3^2 ∈ zEven
        with odd34 have a4^2 ∈ zOdd by (rule even-plus-odd-prop2)
        with odd2 have a2 ∈ zOdd ∧  a4 ∈ zOdd
          by (auto simp add: power-preserves-odd)
        hence tmp: a2+a4 ∈ zEven by (simp only: odd-plus-odd)
        from ev3 ev1 have a1 ∈ zEven ∧ a3 ∈ zEven
          by (auto simp add: power-preserves-even)
        hence tmp2: a1+a3 ∈ zEven by (simp only: even-plus-even)
        from ass have sum4sq(a1,a3,a2,a4)=p*x
          by (auto simp add: sum4sq-def)
        with tmp tmp2 show ?thesis by blast
      next
        assume ¬ a3^2 ∈ zEven
        hence odd3: a3^2 ∈ zOdd by (simp add: odd-iff-not-even)
        with odd34 have a4^2 ∈ zEven by (rule even-plus-odd-prop1)
        with ev1 have a1 ∈ zEven ∧  a4 ∈ zEven
```

```
      by (auto simp add: power-preserves-even)
    hence tmp: a1+a4 ∈ zEven by (simp only: even-plus-even)
    from odd2 odd3 have a2 ∈ zOdd ∧ a3 ∈ zOdd
      by (auto simp add: power-preserves-odd)
    hence tmp2: a2+a3 ∈ zEven by (simp only: odd-plus-odd)
    from ass have sum4sq(a1,a4,a2,a3)=p*x
      by (auto simp add: sum4sq-def)
    with tmp tmp2 show ?thesis by blast
  qed
 next
  assume ¬ a1^2 ∈ zEven
  hence odd1: a1^2 ∈ zOdd by (simp add: odd-iff-not-even)
  with odd12 have ev2: a2^2 ∈ zEven by (rule even-plus-odd-prop1)
  show ?thesis
  proof (cases)
    assume ev3: a3^2 ∈ zEven
    with odd34 have a4^2 ∈ zOdd by (rule even-plus-odd-prop2)
    with odd1 have a1 ∈ zOdd ∧  a4 ∈ zOdd
      by (auto simp add: power-preserves-odd)
    hence tmp: a1+a4 ∈ zEven by (simp only: odd-plus-odd)
    from ev3 ev2 have a2 ∈ zEven ∧ a3 ∈ zEven
      by (auto simp add: power-preserves-even)
    hence tmp2: a2+a3 ∈ zEven by (simp only: even-plus-even)
    from ass have sum4sq(a1,a4,a2,a3)=p*x
      by (auto simp add: sum4sq-def)
    with tmp tmp2 show ?thesis by blast
  next
    assume ¬ a3^2 ∈ zEven
    hence odd3: a3^2 ∈ zOdd by (simp add: odd-iff-not-even)
    with odd34 have a4^2 ∈ zEven by (rule even-plus-odd-prop1)
    with ev2 have a2 ∈ zEven ∧ a4 ∈ zEven
      by (auto simp add: power-preserves-even)
    hence tmp: a2+a4 ∈ zEven by (simp only: even-plus-even)
    from odd1 odd3 have a1 ∈ zOdd ∧ a3 ∈ zOdd
      by (auto simp add: power-preserves-odd)
    hence tmp2: a1+a3 ∈ zEven by (simp only: odd-plus-odd)
    from ass have sum4sq(a1,a3,a2,a4)=p*x
      by (auto simp add: sum4sq-def)
    with tmp tmp2 show ?thesis by blast
  qed
 qed
qed
then obtain b1 b2 b3 b4 where b: sum4sq(b1,b2,b3,b4)=p*x ∧
  b1+b2 ∈ zEven ∧ b3+b4 ∈ zEven by auto
then obtain c1 c3 where c13: b1+b2 = 2*c1 ∧ b3+b4 = 2*c3
  by (auto simp add: zEven-def)
have 2*b2 ∈ zEven ∧ 2*b4 ∈ zEven by (auto simp add: zEven-def)
with b have b1+b2 − 2*b2 ∈ zEven ∧ b3+b4 − 2*b4 ∈ zEven
  by (auto simp only: even-minus-even)
moreover have b1+b2 − 2*b2 = b1−b2 ∧ b3+b4 − 2*b4 = b3−b4 by auto
ultimately have b1−b2 ∈ zEven ∧ b3−b4 ∈ zEven by simp
then obtain c2 c4 where c24: b1−b2 = 2*c2 ∧ b3−b4 = 2*c4
```

```
    by (auto simp add: zEven-def)
  from evx obtain y where y: x = 2*y by (auto simp add: zEven-def)
  hence 4*(p*y) = 2*(p*x) by (simp add: mult-ac)
  also from b have ... = 2*b1^2 + 2*b2^2 + 2*b3^2 + 2*b4^2
    by (auto simp only: sum4sq-def)
  also have ... = (b1 + b2)^2 + (b1 - b2)^2 + (b3 + b4)^2 + (b3 - b4)^2
    by (auto simp add: zadd-power2 zdiff-power2)
  also with c13 c24 have ... = 4*(c1^2 + c2^2 + c3^2 + c4^2)
    by (auto simp add: power-mult-distrib)
  finally have sum4sq(c1,c2,c3,c4) = p*y by (auto simp add: sum4sq-def)
 moreover from y ass have 0 < y ∧ y < p ∧ (nat y) - 1 < (nat x) - 1 by arith
  ultimately show ?thesis by blast
next
  assume ¬ x ∈ zEven
  hence xodd: x ∈ zOdd by (simp add: odd-iff-not-even)
  with ass have ∃ c1 c2 c3 c4. 2*|a1-c1*x|<x ∧ 2*|a2-c2*x|<x
   ∧ 2*|a3-c3*x|<x ∧ 2*|a4-c4*x|<x
    by (simp add: best-odd-division-abs)
  then obtain b1 c1 b2 c2 b3 c3 b4 c4 where
    bc-def: b1 = a1-c1*x ∧ b2 = a2-c2*x ∧ b3 = a3-c3*x ∧ b4 = a4-c4*x
    and bc-abs: 2*|b1|<x ∧ 2*|b2|<x ∧ 2*|b3|<x ∧ 2*|b4|<x
    by blast
  let ?B = b1^2 + b2^2 + b3^2 + b4^2
  let ?C = c1^2 + c2^2 + c3^2 + c4^2
  have x dvd ?B
  proof
    from bc-def ass have
      ?B = p*x - 2*(a1*c1+a2*c2+a3*c3+a4*c4)*x + ?C*x^2
      by (auto simp add: zdiff-power2 sum4sq-def
        zadd-zmult-distrib power-mult-distrib)
    thus ?B = x*(p - 2*(a1*c1+a2*c2+a3*c3+a4*c4) + ?C*x)
      by (auto simp add: mult-ac power2-eq-square
        zadd-zmult-distrib2 zdiff-zmult-distrib2)
  qed
  then obtain y where y: ?B = x * y by (auto simp add: dvd-def)
  let ?A1 = a1*b1 + a2*b2 + a3*b3 + a4*b4
  let ?A2 = a1*b2 - a2*b1 - a3*b4 + a4*b3
  let ?A3 = a1*b3 + a2*b4 - a3*b1 - a4*b2
  let ?A4 = a1*b4 - a2*b3 + a3*b2 - a4*b1
  let ?A = sum4sq(?A1,?A2,?A3,?A4)
  have x dvd ?A1 ∧ x dvd ?A2 ∧ x dvd ?A3 ∧ x dvd ?A4
  proof (safe)
    from bc-def have
      ?A1 = (b1+c1*x)*b1 + (b2+c2*x)*b2 + (b3+c3*x)*b3 + (b4+c4*x)*b4
      by simp
    also with y have ... = x*(y + c1*b1 + c2*b2 + c3*b3 + c4*b4)
      by (auto simp add: zadd-zmult-distrib2 power2-eq-square mult-ac)
    finally show x dvd ?A1 by auto
    from bc-def have
      ?A2 = (b1+c1*x)*b2 - (b2+c2*x)*b1 - (b3+c3*x)*b4 + (b4+c4*x)*b3
      by simp
    also have ... = x*(c1*b2 - c2*b1 - c3*b4 + c4*b3)
```

```
    by (auto simp add: zadd-zmult-distrib2 zdiff-zmult-distrib2 mult-ac)
  finally show x dvd ?A2 by auto
  from bc-def have
    ?A3 = (b1+c1*x)*b3 + (b2+c2*x)*b4 - (b3+c3*x)*b1 - (b4+c4*x)*b2
    by simp
  also have ... = x*(c1*b3 + c2*b4 - c3*b1 - c4*b2)
    by (auto simp add: zadd-zmult-distrib2 zdiff-zmult-distrib2 mult-ac)
  finally show x dvd ?A3 by auto
  from bc-def have
    ?A4 = (b1+c1*x)*b4 - (b2+c2*x)*b3 + (b3+c3*x)*b2 - (b4+c4*x)*b1
    by simp
  also have ... = x*(c1*b4 - c2*b3 + c3*b2 - c4*b1)
    by (auto simp add: zadd-zmult-distrib2 zdiff-zmult-distrib2 mult-ac)
  finally show x dvd ?A4 by auto
qed
then obtain d1 d2 d3 d4 where d:
  ?A1=x*d1 ∧ ?A2=x*d2 ∧ ?A3=x*d3 ∧ ?A4=x*d4
  by (auto simp add: dvd-def)
let ?D = sum4sq(d1,d2,d3,d4)
from d have x^2*?D = ?A
  by (auto simp only: sum4sq-def power-mult-distrib zadd-zmult-distrib2)
also have ... = sum4sq(a1,a2,a3,a4)*sum4sq(b1,b2,b3,b4)
  by (simp only: mult-sum4sq)
also with y ass have ... = (p*x)*(x*y) by (auto simp add: sum4sq-def)
also have ... = x^2*(p*y) by (simp only: power2-eq-square mult-ac)
finally have x^2*(?D - p*y) = 0 by (auto simp add: zdiff-zmult-distrib2)
with ass have ?D = p*y by auto
moreover have y-l-x: y < x
proof -
  have 4*b1^2 = (2*|b1|)^2 ∧ 4*b2^2 = (2*|b2|)^2 ∧
    4*b3^2 = (2*|b3|)^2 ∧ 4*b4^2 = (2*|b4|)^2
    by (auto simp add: power-mult-distrib abs-mult power2-abs)
  with bc-abs have 4*b1^2<x^2 ∧ 4*b2^2<x^2 ∧ 4*b3^2<x^2 ∧ 4*b4^2<x^2
    by (auto simp add: power-strict-mono)
  hence ?B < x^2 by auto
  with y have x*(x-y) > 0
    by (auto simp add: power2-eq-square zdiff-zmult-distrib2)
  moreover from ass have x > 0 by simp
  ultimately show ?thesis by (auto dest: pos-zmult-pos)
qed
moreover have y > 0
proof -
  have b2pos: b1^2 ≥ 0 ∧ b2^2 ≥ 0 ∧ b3^2 ≥ 0 ∧ b4^2 ≥ 0
    by (auto simp add: zero-le-power2)
  hence ?B = 0 ∨ ?B > 0 by arith
  moreover
  { assume ?B = 0
    moreover from b2pos have
      ?B-b1^2 ≥ 0 ∧ ?B-b2^2 ≥ 0 ∧ ?B-b3^2 ≥ 0 ∧ ?B-b4^2 ≥ 0 by arith
    ultimately have b1^2 ≤ 0 ∧ b2^2 ≤ 0 ∧ b3^2 ≤ 0 ∧ b4^2 ≤ 0 by auto
    with b2pos have  b1^2 = 0 ∧ b2^2 = 0 ∧ b3^2 = 0 ∧ b4^2 = 0 by arith
    hence b1 = 0 ∧ b2 = 0 ∧ b3 = 0 ∧ b4 = 0 by auto
```

```
        with bc-def have x dvd a1 ∧ x dvd a2 ∧ x dvd a3 ∧ x dvd a4
          by (auto simp add: zdvd-triv-right)
        hence x^2 dvd a1^2 ∧ x^2 dvd a2^2 ∧ x^2 dvd a3^2 ∧ x^2 dvd a4^2
          by (auto simp only: zpower-zdvd-mono)
        hence x^2 dvd a1^2+a2^2+a3^2+a4^2 by (simp only: zdvd-zadd)
        with ass have x^2 dvd p*x by (auto simp only: sum4sq-def)
        hence x*x dvd x*p by (simp only: power2-eq-square mult-ac)
        with ass have x dvd p by (auto dest: zdvd-mult-cancel)
        moreover from ass prp have x ≥ 0 ∧ x ≠ 1 ∧ x ≠ p ∧ zprime p by simp
        ultimately have False by (unfold zprime-def, auto) }
      moreover
      { assume ?B > 0
        with y have x*y > 0 by simp
        moreover from ass have x > 0 by simp
        ultimately have ?thesis by (auto dest: pos-zmult-pos) }
      ultimately show ?thesis by auto
    qed
    moreover with y-l-x have (nat y) − 1 < (nat x) − 1 by arith
    moreover from y-l-x ass have y < p by auto
    ultimately show ?thesis by blast
  qed
  thus ∃ y. nat y − 1 < nat x − 1 ∧ ¬ ¬ ?Q y by blast
 qed
 with Qt show False by simp
qed
thus is-sum4sq p by (auto simp add: is-sum4sq-def)
qed

theorem four-squares: (n::int) ≥ 0 ⟹ ∃ a b c d. a^2 + b^2 + c^2 + d^2 = n
proof −
 assume n ≥ 0
 hence n = 0 ∨ n > 0 by auto
 moreover
 { assume n = 0
  hence n = sum4sq(0,0,0,0) by (auto simp add: sum4sq-def)
  hence is-sum4sq n by (auto simp add: is-sum4sq-def) }
 moreover
 { assume npos: n > 0
  hence nat n ≠ 0 by simp
  then obtain ps where ps: primel ps ∧ prod ps = nat n
    by (frule-tac a=nat n in factor-exists-general, auto)
  have primel ps ⟹ is-sum4sq (int (prod ps))
  proof (induct ps, auto)
    have sum4sq(1,0,0,0) = 1 by (auto simp add: sum4sq-def)
    thus is-sum4sq 1 by (auto simp add: is-sum4sq-def)
  next
    fix p ps
    let ?X = int (prod ps)
    assume primel ps ⟹ is-sum4sq ?X and primel (p#ps)
    hence prime p and x: is-sum4sq ?X by (auto simp add: primel-hd-tl)
    hence zprime (int p) by (simp only: prime-impl-zprime-int)
    hence is-sum4sq (int p) by (rule zprime-is-sum4sq)
```

```
    with x have is-sum4sq((int p)*?X) by (simp add: is-mult-sum4sq)
    thus is-sum4sq (int (p*prod ps)) by (auto simp only: int-mult)
  qed
  with ps npos have is-sum4sq n by auto }
 ultimately have is-sum4sq n by auto
 thus ?thesis by (auto simp only: is-sum4sq-def sum4sq-def)
qed

end
```