An Axiomatic Formalisation of Trigonometric Functions in Isabelle

Imogen I. Morris

Year 4 Project
School of Mathematics
University of Edinburgh
10th March 2017

Abstract

We provide a formalisation of trigonometric functions in the computer-based proof assistant Isabelle using only elementary limit theorems. Non-geometric definitions of sine and cosine usually rely on power series, however we avoid this by basing our formalisation on a paper by Gerson Robison [28] that starts from three simple axioms. Proofs in this paper make no explicit appeals to the theorems of analysis that are being used: we find that it is not always obvious which theorems are intended. It becomes clear, after formalisation, that some statements which seem mathematical are imprecise and some subtlety is required to give them precision. This broader aim of this work is to prove the power series characterisation for sine and cosine in Isabelle by formalising the arguments used by Euler in his famous 'Introduction to the Analysis of the Infinite' [10]. As the formalisation is intended to be readable by mathematicians, we use the structured proof language Isar.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Imogen I. Morris*)

Contents

Chapter 1

Introduction

Mathematicians of the 18$^{\text{th}}$ century and earlier are usually portrayed as exceptionally unrigorous. Even great mathematicians such as Euler, Fourier and Newton are considered to have found their results using a series of ingenious tricks. Formal proof verification is a recent innovation in which proofs are mechanically checked by a computer. The software can be reduced to a kernel of code which can then itself be verified by other verification systems or by humans [26]. Besides this, type checking allows theorems to have a special status and guarantees that they can only be derived from axioms and inference rules [14].

These seem to take opposite ends in the spectrum of mathematical rigour. Yet perhaps the 18th century mathematicians instinctively followed consistent rules, without making them explicit. We can try to formalise their reasoning faithfully in a proof verification system. One example is Jacques Fleuriot's formalisation of Newton's *Principia Mathematica* [12]. He uses nonstandard analysis [27] to provide a rigorous foundation for infinitesimals, those infinitely small numbers that were called 'the ghosts of departed quantities' by Bishop Berkeley [2].

This project began by aiming to formalise the proof of the power series characterisation for sine and cosine, that was given by Euler in his *Introductio in Analysin Infinitorum*. A paper by Mark McKinzie and Curtis Tuckey [21] uses nonstandard analysis to give Euler's reasoning a modern standard of rigour. This paper could therefore serve as a guide to formalising Euler's reasoning in Isabelle. However, before we can even formalise this argument, we need definitions of sine and cosine. The proof verification system that we are using, Isabelle [25], currently defines sine and cosine using their power series. To avoid a circular argument, we looked for an alternative definition.

1.1 Standards of Rigour

A mathematician's work is mostly a tangle of guesswork, analogy, wishful thinking and frustration, and proof, far from being the core of discovery, is more often than not a way of making sure that our minds are not playing tricks.

- Gian-Carlo Rota, Introduction to *The Mathematical Experience* [5]

Before the Ancient Greeks, reading mathematics was much like reading a recipe book. It was presented in the form of algorithm that was not necessarily even given the grace of being stated as a general procedure, or of being justified [1]. The Ancient Greeks are regarded to have brought in a new age of rigour in mathematics. Euclid's *Elements* [9] is the archetypal example. It is considered to be the first explicitly axiomatic system, and carefully builds up each result by attempting to use only previously proved lemmas. However, Euclid thought of his propositions as synthetic a priori truths rather than as well-chosen assumptions. That is why the discovery of non-Euclidean geometry was so controversial and significant. The influence of the Greeks remained until Euler's time, when geometry was still considered to be more rigorous and sound than any other area of mathematics, especially the doubtfully justified calculus, which involved adding up infinitely many infinitely small quantities, that seemed to disappear and appear whenever Newton and his comrades desired them to [5, p. 238].

Cauchy brought rigour to analysis by replacing these sorts of arguments with arguments concerning limits, and avoiding infinite numbers. Later mathematicians (such as Riemann and Weierstrass) built on his work resulting in the analysis we are familiar with today. However, mathematicians still routinely made inferences that were only justified by intuition, or by the belief that the step could be easily verified.

In the late 19th century, Bertrand Russell and Alfred North Whitehead embarked upon a project to build mathematics up from logical foundations. The result of their labour *Principia Mathematica* [31] is simultaneously one of the most influential books ever written and, due to its forbidding formalisms, rarely ever read. The logical system they developed gave a framework for Gödel's Incompleteness Theorems, which demonstrate that there will always exist true statements (Gödel statements) that are unprovable within the formal system. The discovery of the Incompleteness Theorems disheartened the project. However, the theorems should actually strengthen the importance of formal proof, because without it we may not recognise the branching points where we could take either the 'true' Gödel statement or its negation to continue the theory.

Mechanical proof verification has brought in a new way of approaching formal proof. When formal proofs are written by humans, it is very easy to make a mistake, especially since perfect precision is required. The machine can spot these errors, and in the case of an interactive theorem prover, it can provide hints on how to fix them and proceed.

This seems especially important in an era when proofs are becoming unmanageably long. Thomas Hales' proof of the Kepler conjecture was too long and complicated for the referees to verify satisfactorily. The only way for him to remove the doubt over his proof was to formalise it in an interactive proof system, although this took several people over a decade to complete. Hales says

> *In truth, my motivations for the project are far more complex than a simple hope of removing residual doubt from the minds of few referees. Indeed, I see formal methods as fundamental to the long-term growth of mathematics. [16, p. 1]*

Proofs will only become longer, and mathematicians will only become more specialised, with fewer colleagues who can verify their work. Eventually, mechanical proof verification will become an indispensable aid.

1.2 A Motivating Problem: Euler's Analysis of the Infinite

Considered to be one of the most influential textbooks ever written, Euler's *Introductio in Analysin Infinitorum*, places functions, a new concept at the time, as the focus of study in analysis [6, p. 179], as well as introducing and proving significant and beautiful results such as the Euler identity

$$e^{i\theta} = \cos\theta + i\sin\theta$$

his series characterisation of π which demonstrates a surprising connection to number theory

$$\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$$

the power series of sine and cosine

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1!} x^{2n+1}$$
$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n!} x^{2n}$$

and he accomplishes all this without calculus, using merely 'ordinary algebra'. However, a cursory examination of his arguments will show that 'ordinary algebra', for Euler, includes all sorts of surprising and ingenious arguments concerning infinity. In fact, his arguments using infinities are so inventive and marvellous that many historians have criticised his methods, saying, for example

> *He was incautious in his use of infinite series, often applying to them laws valid only for finite sums. ... by such careless approaches he luckily obtained truly profound results. [11, p. 435]*

> *Today, we recognise that Euler was not so precise in his use of the infinite as he should have been. His belief that finitely generated patterns and formulas automatically extend to the infinite case was more a matter of faith than science, and subsequent mathematicians would provide scores of examples showing the folly of such hasty generalizations. [7, p. 222]*

It is true that many people in Euler's time did not understand how to manipulate infinities correctly: as Euler notes in his preface to the *Introductio*, 'they entertain strange ideas about the concept of the infinite' [10, p. v]. However, Euler realised that dealing with infinities required careful reasoning, calling it a 'subtle art' [10, p. v]. Perhaps this is one reason he was able to surpass his contemporaries.

Mark McKinzie and Curtis Tuckey [21] have used nonstandard analysis (see Section 1.2.1) in order to treat the arguments that Euler made in the *Introductio* with a modern standard of rigour, but at the same time remaining faithful to his arguments using infinities. In fact, McKinzie and Tuckey say 'There is the occasional misstep, but on the whole, Euler's use of the infinite and infinitesimal is consistent and clear.' [21, p. 340] With the aid of a proof verification system, we find that even modern mathematicians make an 'occasional misstep'. The major difference between Euler's intuitive consistency and modern rigorous mathematics is simply in defining the rules explicitly, and in saying which rule we are using and how we are using it. What McKinzie and Tuckey have shown, and that formal verification of their paper can reinforce, is that having high standards of rigour does not mean compromising elegant and beautiful proofs such as those that Euler gave.

1.2.1 Nonstandard Analysis

In Section 1.1, we saw that the concept of infinitely large and small numbers was discarded in favour of the rigorous theory of limits. However, there is another way to make analysis rigorous which is closer to the way people treated analysis before and during Euler's time.

Infinite and infinitesimal numbers

Definition 1.2.1 (Infinite number)**.** *We define an infinite number to be a number X such that for any $a \in \mathbb{R}$ and $n \in \mathbb{N}$ we have $|X| > na$.*

Definition 1.2.2 (Infinitesimal number)**.** *We define an infinitesimal number to be a number ϵ such that for any $a \in \mathbb{R}$ and $n \in \mathbb{N}$ we have $n|\epsilon| < a$.*

If we consider $\mathbb{R}$, we see that the only infinitesimal is 0, and all numbers satisfy the Archimedean property, thus there are no infinite numbers. However, we can extend $\mathbb{R}$ to ${}^*\mathbb{R}$, the field of hyperreals, which contains infinite numbers and nonzero infinitesimals as well as the ordinary real numbers. This is possible either by axiomatisation or by construction, so the hyperreals are consistent if and only if the reals are consistent (assuming the Axiom of Choice) [13].

The infinitely-close relation

We can see that for every real number r and infinitesimal ϵ there is the corresponding hyperreal $r + \epsilon$. In fact, given any hyperreal h, there is a unique real number r_h and unique infinitesimal ϵ_h such that $h = r_h + \epsilon_h$. We call this unique r_h the standard part of h. When h is a real number, it is its own standard part, and the corresponding infinitesimal is zero. We would like to say 'h is infinitely close to r_h'. This motivates the following definition

Definition 1.2.3 (Infinitely-close relation)**.** *We say that two hyperreals h_1 and h_2 are infinitely-close, written as $h_1 \simeq h_2$, if their difference, $h_1 - h_2$, is infinitesimal.*

We can verify that $\simeq$ is an equivalence relation using properties of infinitesimals. The object of this project is to show that $\sin \epsilon \simeq \epsilon$ for infinitesimal ϵ, as we will see in Section 1.2.2.

The Transfer Principle

The reals satisfy the Archimedean property

$$\forall x \in \mathbb{R}, \forall a \in \mathbb{R}, \exists n \in \mathbb{N}, x < na. \tag{1.1}$$

Notice from the definition of infinite numbers that the Archimedean property is false for hyperreal x. It is not the case that

$$\forall x \in {}^*\mathbb{R}, \forall a \in \mathbb{R}, \exists n \in \mathbb{N}, x < na. \tag{1.2}$$

However, we can see why it is not true. We have not really extended the Archimedean property correctly, because we allowed x to take infinite values, whereas a and n were still required to take finite values. Thus x had an 'unfair advantage'. We can change it to

$$\forall x \in {}^*\mathbb{R}, \forall a \in {}^*\mathbb{R}, \exists n \in {}^*\mathbb{N}, x < na. \tag{1.3}$$

which *is* true for the hyperreals (here ${}^*\mathbb{N}$ is the set of hypernaturals which we can informally think of as natural numbers that may take infinite values).

There is an underlying principle at work here. The Archimedean property for the reals 1.1, which is true, has a nonstandard version 1.3, which is also true. However, from 1.2, we can see that this principle is more subtle than simply replacing any chosen set with its nonstandard extension. Further, not every statement about the reals has a nonstandard version with the same truth value, e.g. 'Every infinitesimal is zero' is true for the reals and false for the hyperreals.

What we have seen here is an example of the Transfer Principle, which, applied to a certain well-formed statement about the reals, will give a statement concerning the hyperreals which is true if and only the original statement was true. This also works the other way round: if we have a well-formed statement concerning the hyperreals, the Transfer Principle will give us a corresponding statement concerning the reals. Thus given a statement in standard analysis, we can choose to prove either that statement or its transfer, and the proof may happen to be more elegant using nonstandard analysis or vice-versa. There is no dilemma between using standard analysis, and using nonstandard analysis. Rather, the choice is between using half of the proof methods available to us, and all of them.

1.2.2 Diagrammatic Justification

McKinzie and Tuckey do not define sine and cosine. They are following Euler's reasoning and he did not define these functions either. However, McKinzie and Tuckey do assume certain basic properties of sine and cosine, such as the double-angle formula and the symmetry of sine around $x = 0$. Besides these, their

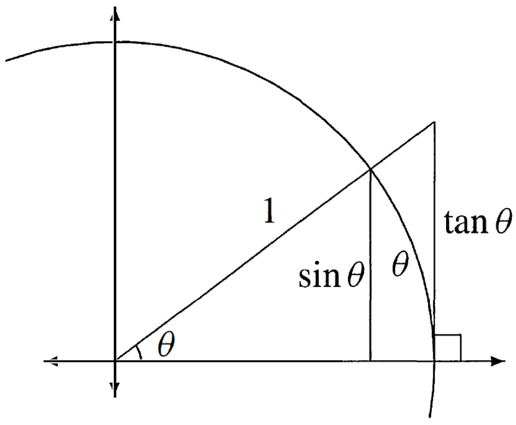

Figure 1.1: $0 < \sin\theta < \theta < \tan\theta$ in the first quadrant *(Taken from McKinzie and Tuckey [21])*

argument requires $\sin\epsilon \simeq \epsilon$ for infinitesimal ϵ, which is a nontrivial property: we will later discover how much work is required to formally prove its standard analysis counterpart[1]. They give a proof of this property. This proof in turn requires a property, namely that $0 < \sin\theta < \theta < \tan\theta$ for $0 < \theta < \pi/2$. Its standard analysis counterpart seems to require continuity of sine to prove analytically[2], so it is not as basic a property as the double-angle formula, for example. McKinzie and Tuckey justify the use of this property by referring us to the diagram given in Figure 1.1.

We might argue that it is right to use a geometric argument in this case, since Euler used a geometric definition of sine and cosine, as we will see in Section 1.3.2. However for several reasons this makes their argument very difficult to formalise:

1. A diagram is not a formal geometric argument. Perhaps we are able to 'see' that the line representing $\sin\theta$ is shorter than the arc representing θ, and we are able to 'see' that for any other θ in this quadrant, the diagram would be essentially the same. But the mechanism by which we are able to 'see' in this way is unclear, and may even be untrustworthy[3].

2. Our proof verification system, Isabelle, cannot interpret diagrams.

3. Even if a formal geometric argument was given, e.g. of the kind that Euclid gave in the *Elements* [9], axiomatised geometry in Isabelle is not at the point where sine and cosine could be defined (see Section 1.3.2).

So it is left to us to work out how we can define sine and cosine and formally prove these properties.

[1]In actual fact they prove $\sin\epsilon \simeq \epsilon(\text{mod}\epsilon)$ which is stronger and allows substitution even when one side is taken to an infinite exponent.

[2]Intuitively, since $\sin\epsilon \simeq \epsilon$ for infinitesimal ϵ is true, the inequality $\sin\theta < \theta$ must be tight and thus difficult to prove.

[3]See *Philosophy of Mathematics: An Introduction to the World of Proofs and Pictures* [3] for an interesting and controversial discussion on whether pictures can be mathematical proofs.

1.3 Defining Sine and Cosine in Isabelle

1.3.1 Current Definition

The current definitions of sine and cosine in Isabelle are in terms of their power series. They are defined in a very general setting: originally they were formalised by Jacques Fleuriot only for the reals. But many theories in Isabelle which originally only concerned the real or complex numbers have now been extended to their most general setting (e.g. a metric space rather than $\mathbb{R}$, or an algebraically closed field rather than $\mathbb{C}$), since that way the theories are more unified and encompass more of mathematics [18]. This also eliminates the need to prove the same proposition separately for each type of space.

Below is their expression in Isabelle

```
definition sin_coeff :: "nat ⇒ real"
  where "sin_coeff=(λn. if even n
         then 0 else (- 1)^((n-Suc 0) div 2)/(fact n))"

definition cos_coeff :: "nat ⇒ real"
  where "cos_coeff=(λn. if even n
         then ((- 1)^(n div 2))/(fact n) else 0)"

definition
sin :: "'a ⇒ 'a::{real_normed_algebra_1,banach}"
  where "sin = (λ x. ∑ n. sin_coeff n *R x^n)"

definition
cos :: "'a ⇒ 'a::{real_normed_algebra_1,banach}"
  where "cos = (λ x. ∑ n. cos_coeff n *R x^n)"
```

Here `{real_normed_algebra_1,banach}` means that `'a` is an element of a space that is both a real normed algebra and a Banach space and $*_R$ is the scalar multiplication defined on that space. The `fact` refers to the factorial function. The λ indicates that `sin_coeff` takes `n` as an argument. These definitions are unsuitable for our purpose: we wish to *prove* the power series definition, and we cannot do that by assuming it.

The power series seems to be the most common definition given in university-level courses[4] however, we can dispute whether it should be taken as the definition at all. We might argue that the definition should be the characterising property, it should be the *meaning* of whatever we are discussing, not simply the most convenient way of specifying it.

Take, for example, the circle. A circle certainly *can* be uniquely determined by its equation $x^2 + y^2 = r^2$, but we wouldn't usually say that that is what a circle 'is'. The equation is incidental, arbitrary. A circle is the shape traced out by a point that moves at a constant distance from the centre. The meaning of a circle does depend on context, of course. Perhaps we are struck by the beautiful

[4] Although any kind of rigorous definition at all is actually uncommon: usually the properties of these ubiquitous functions are simply assumed.

symmetry of the equation, and decide that we want to give it a name for its own sake, and only later do we realise that it also has a geometric meaning.

In this case, with sine and cosine, our context is Euler, and the mathematics of his time was still deeply under the influence of the Greeks and their geometry.

1.3.2 Geometric Definition

In schools, sine and cosine are usually defined using the ratios of sides of a right-angled triangle. This definition is not the original one, however. The Ancient Greeks were the first to define sine and cosine, although their definition was stated purely geometrically and not in terms of functions [17]. They defined sine as 'the chord of a circle subtended by an angle': as we are no longer so familiar with the terminology of Euclid's geometry, this is illustrated in Figure 1.2. Hipparchus, who was the first person to compile a table of trigonometric values, proved theorems concerning the ratios of sides of a right-angled triangle, so the Greeks were aware of this alternative characterisation of sine and cosine [19, p. 68].

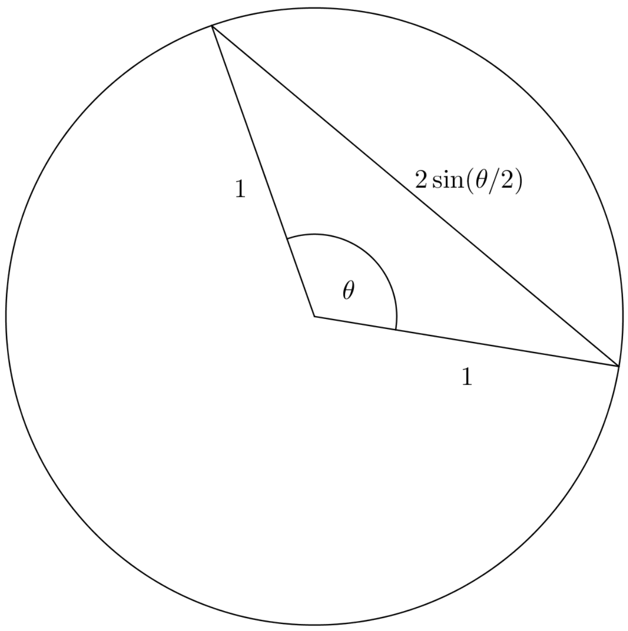

Figure 1.2: Sine as a chord

Their definition lived on well into the 19th century, and certainly during Euler's day. In the *Introductio*, Euler does not explicitly state what definition he is using, but since he refers to the radius of a circle as the 'total sine', it is evident he is using the standard Greek definition in terms of chords. By Euler's time, this definition had changed slightly to the 'half-sine': in this way the greatest value of sine corresponds to the radius rather than the diameter of the circle [24].

In the *Introductio*, Euler assumes the radius of a circle to be 1. The length of the chord corresponding to an angle of π, in this case the diameter, is 2 and as Euler takes sine to be half the chord, we have $\sin \pi/2 = 1$. This explains why Euler calls the radius the 'total sine': firstly, it is the maximum value that sine may take, secondly, as per the Greek definition, the half-chord and radius really are the same line for this value of the arc. Euler then goes on to list some properties of sine and cosine 'all of which follow from trigonometry' and derives many others [10].

Even though Euler used the Ancient Greek definition, he had begun to consider sine and cosine as functions, although 'of the arc' rather than 'of the angle' as we usually think of them. In fact, he begins the *Introductio* by introducing his readers to functions, which suggests that he considered them a central concept of his work. Although functions had been previously considered, for example, by Newton in his *Principia*, Euler's treatment of them was original and he was the first to introduce the $f(x)$ notation [4, p. 268].

Geometry has been axiomatised in Isabelle [20][29][22], but not up to the point where sine and cosine could be defined purely in terms of geometry without using their power series[5]. So we must search for another way to define them.

1.3.3 Analytic Definition

Sine is transcendental and thus cannot be defined by a finite number of algebraic operations. So, to define it analytically, we must use some argument that appeals to infinities (whether actual infinities or limits).

There are some analytic definitions which carefully maintain connections to the geometric meaning of sine and cosine. One, which is considered in a blogpost by Timothy Gowers [15], is to define the angle in terms of the arclength of a circle, and then to obtain the cosine and sine of the angle from the coordinates of the circle where it is cut by a line making that angle with the x-axis.

W. F. Eberlein defines $\cos\theta$ and $\sin\theta$ as the real and imaginary parts of cis θ respectively, and obtains cis θ essentially by the same method as Gowers, except in the complex plane rather than $\mathbb{R}^2$ [8]. Both consider the geometric meanings of their derivations. Eberlein begins by giving a geometrically justified derivation before treating it in an analogous but completely analytic way. Gowers makes reference to circles and infinitely-small triangles[6], which shows he is thinking about it in a geometric way, even if his derivation ends up being purely analytic.

Euler did know about the calculus, both integration and differentiation, but he believed it was unsound [19, p. 232]. The *Introductio* was meant as a pre-calculus book: at several points in the text Euler mentions that a certain derivation might be done more easily using calculus, but he avoids it [10]. In order to remain faithful to his methods, we should also try to avoid it.

1.3.4 Axiomatic Definition

In 1966 Gerson Robison read Eberlein's definition of sine and cosine, and was inspired to present his own, writing that 'There is another elementary approach

[5]Defining angles completely is the greatest obstacle here

[6]This is the classic example of an argument which has an elegant treatment in nonstandard analysis.

which seems not to be generally known, that by way of the functional equation'. Three functional equations, or axioms, which sine and cosine are known to satisfy, are given

$$\begin{aligned}
&C(x-y) = C(x)C(y) + S(x)S(y), \forall x, y \in \mathbb{R} \\
&S(p) = 1 \\
&S(x) \geq 0, \forall x \in [0, p]
\end{aligned}$$

where S and C are real-valued functions of a real variable and p is a positive real number. All further properties are built up from these, including continuity, a definition of π and $\lim_{x\to 0} \sin x/x = 1$. Sine and cosine are defined as the functions satisfying these equations when $p = \frac{\pi}{2}$. For most of the paper, their more general counterparts S and C are considered. At the end of the paper, π is defined as $2\lim_{x\to 0}(\sin x)/x$. He also proves that these three axioms guarantee uniqueness, up to p.

Robison himself seems to consider these functional equations as a definition. In a sense, they are, just as we can 'define' the reals by giving a set of axioms which they satisfy. But usually, we see definitions as conservative extensions to a theory, because adding a definition will not affect the consistency or power of a theory, whereas adding an axiom can, and so is not conservative extension. In this sense, the functional equations do not provide a definition, unless we can prove that the functions satisfying the axioms are 'well-defined'. The clearest way to do this, is to provide a model for the axioms, i.e. to find some functions that do satisfy the axioms. We already know one: sine and cosine as defined by the power series definition. We can instantiate our locale to these functions. This means we have no doubt about the consistency of our axioms. Unfortunately, as soon as we instantiate the locale to the power series definitions, we cannot prove the power series characterisation of sine and cosine without making a circular argument. We would like most of all to prove the consistency of the axioms and then to claim that the functions given by them must be well-defined. However, consistency is a metatheoretical result and in Isabelle we cannot use it at the theory level. As we don't even distinguish between metatheoretical and non-metatheoretical results in ordinary mathematics, we may wonder if there are some results we would be unable to prove if we could not reason metatheoretically. We will revisit this in Section 2.3.

Here we notice that the real power of our axiomatic 'definition' is its flexibility: whatever we choose to model the axioms, whichever of the other options that we have discussed above, it will be compatible with the properties we have derived from the axioms. As Euler knew these properties of sine and cosine, these axioms are consistent with the way he thought of the functions. In fact, Euler was very interested in the determination of functions using equations: in the 18th century this became the focus of analysis [6, pp. 179-180].

It is also very interesting that we can derive more complex properties of sine and cosine, such as continuity, from these three simple axioms. Once they are formalised in Isabelle, we can be doubly sure that Robison is not assuming any properties other than those specified.

1.4 Introduction to Isabelle/Isar

In an Isabelle theory, we may write axioms, definitions and theorems. Theorems can be indicated by either `lemma` or `theorem`. For example

```
lemma S2x: "S (2*x) = 2*S x * C x"
```

Here `S2x` is the name of the lemma. It is best to give it a name if we want to use it to prove something else later on. We can also refer to it by writing the statement in backticks, e.g. `` `S(2*x)= 2*S(x)*C(x)` ``. In the lemma, `S` and `C` are functions. In Isabelle, function application is indicated by a space[7]. It is only to specify a complex argument such as `2*x` that we need brackets. Since the argument is not enclosed in brackets, `S 2*x` would mean `(S 2)` multiplied by `x`. We might need to specify that `x` is a real number by adding type information

```
lemma S2x: "S (2*(x::real)) = 2*S x * C x"
```

However, if we have already defined `S` and `C` to be functions from $\mathbb{R}$ to $\mathbb{R}$, i.e. of type `real ⇒ real`, this is unnecessary since Isabelle can deduce that `x` must be a real number. We can also write proofs in our theory. Isabelle will only accept a correct proof. A one-line proof is written as `by` (method).

```
lemma frac_mult_by_denom: "b ≠ 0 ⟹ (a/b)*b=(a::real)"
  by simp
```

We used `simp` here which is the most common automatic proof method, especially for equational reasoning. It performs rewriting of the statement. We can also use previous theorems to show new statements. Our theorems can only be used if they have been proven beforehand.

```
 lemma "(2/(1::real))*1 = 2"
   by (rule frac_mult_by_denom, rule one_neq_zero)
```

Separating the methods by a comma means 'apply method 1 first, then apply method 2 to what remains to be proven. In this case, once we apply apply `rule frac_mult_by_denom`, we still need to prove `1≠0` which becomes our 'current goal'. Luckily, there is a lemma `one_neq_zero` which is exactly what we need to prove. Most proofs take more than one line. For that, we use a structured proof language called Isar [23], which is designed to be readable by humans. We indicate an Isar proof by writing `proof` (method). Usually we just use the `-` method here, which does nothing except insert any assumptions that the lemma might have. A proof consists of a collection of sublemmas which are each prefixed

[7]The case of a function with two or more arguments is slightly more complex than this example, where there is only one argument. Function application is again indicated by spaces between arguments, e.g. if `S` is a real-valued function of two real arguments we could obtain a real number by writing `S a b` for real a,b. However, we can also obtain a real-valued function of one real argument by applying `S` to only one argument, e.g. `S a` is a function from the reals to the reals.

by `have`. We often write `then have` which means that we are using the previous `have` statement to help us prove the current one. Each `have` statement behaves almost exactly like a `lemma`, so we can even begin a second Isar proof inside the original one in order to prove a particularly tricky `have` statement.

```
lemma Srecurrence:
    "4*((S(x/2))^2)*(1-(S(x/2))^2) = (S x)^2"
proof -
   have "2*S(x/2)*C(x/2) = S (2*(x/2))"
     by (rule S2x[symmetric])
   then have "2*S(x/2)*C(x/2) = S x"
     by simp
   then have "(2*S(x/2)*C(x/2))^2 = (S x)^2"
     by simp
   then have 1:"4*(S(x/2))^2*(C(x/2))^2 = (S x)^2"
     by (auto simp:power_mult_distrib)
   have 2: "C(x/2)^2 = 1 - S(x/2)^2"
   proof -
      have "(C (x / 2))^2 + (S (x / 2))^2 = 1"
         using Cx_sq_plus_Sx_sq_eq_1 by simp
      then show ?thesis
         by algebra
   qed
   from 1 show ?thesis
     by (subst 2[symmetric]) simp
qed
```

We use `show` instead of `have` for the statement that matches the goal of the current proof. The goal of the current proof is usually, but not always, the statement that we wrote before `proof`. If it is, we can just write `?thesis` as an abbreviation. It doesn't matter how we format the proof, except that it makes it more readable if we put each `have` statement on a separate line. The proofs in the next chapter are all written in Isar.

Chapter 2

The Mechanisation

2.1 Locales

Infallibility is never attainable, and therefore some element of doubt should always attach to every axiom and to all its consequences.

- Bertrand Russell and Alfred North Whitehead,
Principia Mathematica (2nd ed.), p. 59

We have formalised much of Robison's paper on the axiomatic definition of sine and cosine in an Isabelle theory `Circular_Functions`. In Isabelle, we could treat the functional equations for S and C as axioms, however, this would mean we could never prove that there really are functions satisfying these equations at a later stage without reformulating the whole theory. Thus we use a `locale` environment which allows us to treat the functional equations as assumptions which may be discharged and thus ensures the axioms are not added in an unsafe way. Informally, this mean that all theorems proved within the locale implicitly carry these assumptions as extra premises.

```
locale circular_functions =
  fixes S :: "real ⇒ real"
  fixes C :: "real ⇒ real"
  fixes p :: "real"
  assumes Cxminusy : "C(x)*C(y) + S(x)*S(y) = C(x-y)"
    and   Sp_eq_1 [simp]: "S p = 1"
    and   Sx_greq_0  [intro]:
    " ⟦  p ≥ x ; x ≥ 0 ⟧ ⟹ S x ≥ 0"
    and   p_gr_0: "p > 0"
```

The assumptions of the locale may be instantiated to any functions S' and C' satisfying them. Of course, we know these functions are unique up to p.

2.2 Continuity of S and C

The lemmas in Robison's paper that describe the basic properties of S and C were given without proof, and most had short proofs in Isabelle. These are given in Appendix .2. The first interesting proofs were of the lemmas leading up to the proof that S and C are continuous. We discuss these lemmas below.

2.2.1 S and C are continuous at zero

Continuity of C at zero

In Robison we have the following

Lemma 2.2.1. *For any constant c, if $C(x) \geq 1 - c$, then $C(x/2) \geq 1 - c/2$.*

Proof. $C(x/2) \geq (C(x/2))^2 = (1 + C(x))/2 \geq (1 + (1 - c))/2 = 1 - c/2$.

In Isabelle we formalise it as the following

```
lemma Cseq_goes_to_1: assumes p_greq_x: "p ≥ x "
                      and x_greq_0:" x ≥ 0"
                      and C_greq: "C x ≥ 1 - c"
                      shows "C (x/2) ≥ 1 - c/2"
```

We assumed $0 \leq x \leq p$ since Robison previously restricted the domain of S and C to $[0, p]$ for this lemma. The proof is straightforward and follows Robison's

```
  proof -
    have "C (x/2) ≥ (C (x/2))² "
    proof -
      have "C(x/2) ≤ 1" using p_greq_x and x_greq_0
        by (simp add: Cx_bounds)
      moreover have "0 ≤ C(x/2)"
        using p_greq_x and x_greq_0
        by (simp add: Cx_bounds)
      ultimately have "C (x/2)*C (x/2) ≤ C (x/2)"
        by (rule mult_left_le)
      then have "(C (x/2))² ≤ C (x/2)"
        by (subst power2_eq_square)
      then show ?thesis
        by simp
    qed
    moreover have "(C (x/2))² = (C x + 1)/2"
      by (subst Crecurrence) simp
    moreover have "...  ≥ (1+(1-c))/2 using C_greq
      by simp
    ultimately show ?thesis
      by simp
  qed
```

This lemma is then used along with $C(x) \leq 1$ and the Squeeze Theorem:

```
lemma LIMSEQ_squeeze:
  assumes squeeze: "∀n. a n ≤ b n ∧ b n ≤ c n" and
         lima: "a ⟶ (L::real)" and
         limc: "c ⟶ L"
   shows limb: "b ⟶ L"
```

to show that $\lim_{n\to\infty} C(p/2^n) = 1$. From this, and that C is nonincreasing on $[0, p]$ we intuitively see that for small values of x we ought to get $\lim_{x\to 0} C(x) = 1$ since limits should not depend on values of x far from the limit.

As $C(0) = 1$, we were then able to prove that C is continuous at zero. The proof is included in the appendices. The proof that S is continuous at zero was more complex and it was not provided by Robison. Since S and C are shifts of each other, a first thought might be to obtain this in some way from the previous statement. However, the point at which sine is equal to zero is quite different from the point at which cosine is equal to zero (the first is a point of inflexion, the second is a maximum).

Continuity of S at zero

We decided to prove S is continuous at zero similarly to the way C was shown continuous at zero. We know that $Sx \geq 0$ for $x \in [0, p]$. Thus we could use the Squeeze Theorem to show $\lim_{n\to\infty} S(p/2^n) = 0$ if we had a suitable sequence providing an upper bound on $S(p/2^n)$ where $n \in \mathbb{N}$. It is well known that $\sin x \leq x$ for small x, so the obvious choice is $S(p/2^n) \leq 1/2^n = S(p)/2^n$. It was unclear how to prove this without using continuity. For C, it was easier to find a sequence to form a bound to use with the squeeze theorem. We had the following lemma

```
lemma Crecurrence: "(C x + 1)/2 = (C(x/2))²"
```

which allowed us to prove if $C(x) \geq 1 - c$, then $C(x/2) \geq 1 - c/2$. Using a similar lemma for S,

```
lemma Srecurrence: "4*((S(x/2))²)*(1-(S(x/2))²)=(S x)²"
```

we are able to show

```
lemma Sseq_goes_to_0: assumes "p ≥  x "
                      and "x ≥ 0" and "c ≤ (sqrt 39)/10"
                      and "c ≥ 0" and "S x ≤ c"
                      shows "S (x/2) ≤ (4/5)*c"
```

This should still be true if we had `S (x/2)≤(1/2)*c` rather than `S (x/2)≤ (4/5)*c`. However, it would be much more difficult to prove: all proofs that came to mind used continuity. If we choose some fraction[1] that is greater than 1/2[2], it means we can throw away some terms in the chain of inequalities that will be required to prove it. Originally we chose 2/3 as our scaling fraction which made

[1]We only choose a rational number because it is easier to calculate with. Positive irrational numbers greater than 1/2 are equally suitable otherwise.

[2]Greater than 1/2, but also less than one, otherwise this lemma will be of no use to us in trying to find the limit of $S(x)$ as x goes to zero.

the lemma slightly stronger. Unfortunately that meant that the next lemma, which was proved by induction, had a stronger base case to prove:

```
lemma Spseq: shows "S(p/(2^n)) ≤ (2/3)^n"
```

So, in order to simplify the proofs, 2/3 was changed to 4/5 in both lemmas.[3] The base case was the largest part of the proof of `Spseq`, and required that the lemma be changed to

```
lemma Spseq: shows "S(p/(2^(n+3))) ≤ (4/5)^(n+3)"
```

so that it could be assumed that $n + 3 \geq 3$ and also needed the following lemma

```
lemma Sprecurr: "S(p/(2^n)) ≥ 2*S(p/2^(n+1))*(1 - 1/(2^(n+1)))"
```

After this lemma, the proof for S was exactly the same as for C.

2.2.2 S and C are continuous

Robison points us towards a proof that S and C are continuous only by indicating which lemmas should be used. We quote exactly:

> Lemma 19. S and C are continuous. *Proof.* 18, 6a and 12a, with $y = \Delta x$.

Lemmas 18, 6a and 12a are respectively

> S and C are continuous at zero.
>
> $S(x + y) = S(x)C(y) + C(x)S(y)$
>
> $C(x + y) = C(x)C(y) - S(x)S(y)$

His Δx is rather suggestive notation since Δ is often used to mean the change in a quantity. This, along with Lemma 6a, leads us to consider $\lim_{\Delta x \to 0} S(x+\Delta x) = \lim_{\Delta x \to 0}(S(x)C(\Delta x)+C(x)S(\Delta x)) = S(x)\lim_{\Delta x \to 0} C(\Delta x)+C(x)\lim_{\Delta x \to 0} S(\Delta x))$ and then, considering the definition of continuity in terms of limits, we can use Lemma 18 to get $\lim_{\Delta x \to 0} S(x + \Delta x) = S(x)C(0) + C(x)S(0) = S(x)$. This is equivalent to $\lim_{\Delta x \to x} S(\Delta x) = S(x)$, which by definition means that $S(x)$ is continuous[4].

This requires translation from the usual way of expressing limits, $\lim_{x \to a} f(x) = L$, to Isabelle's notation, `f -a→L` or equivalently `(λx. f x)-a→L`. In the latter, the λ indicates explicitly that `x` is the argument of `f x`.

[3]Besides the arbitrary choice of 4/5, the assumption that $c \leq \sqrt{39}/10$ may be puzzling. The key idea of this proof is that we want to show $(S(x/2))^2$ is less than some multiple of c^2. Unfortunately, using the `S_recurrence` lemma gives us that $(S(x/2))^2$ is less than some multiple of c^2 plus some multiple of c^4. But $c^4 < c^2$ for $0 < c < 1$, so we just need to make sure that c^4 is far enough below c^2. We find $c < \sqrt{39}/10$ makes c^4 far enough below c^2, by doing some arithmetic, which we initially worked out with pen and paper.

[4]Paradoxically, although the Δx notation is suggestive, it is also misleading because it implies that Δx is a function of x, which would make the next step invalid. This makes it even more incomprehensible why our intuition works the way it does and would add difficulties to the task of mechanical interpretation of human proof

```
lemma S_cont: "isCont S a"
  proof -
    from C_continuous_0 have "C -0→ (C 0)"
      unfolding isCont_def
      by simp
    then have 1:"C -0→ 1"
      by (subst C0_eq_1[symmetric]) simp
    have 2:"(λ(x::real). S a) -(0::real)→ S a"
      by simp
    from S_continuous_0 have "S -0→ (S 0)"
      unfolding isCont_def
      by simp
    then have 3:"S -0→ 0"
      by (subst S0_eq_0[symmetric]) simp
    have 4:"(λx. C a) -(0::real)→ C a"
      by simp
    from 1 2 have "(λ(x::real). (C x)*(S a)) -0→ S a"
      using  tendsto_mult
      by fastforce
    moreover from 3 4 have 6:"(λx. (S x)*(C a)) -0→ 0"
      using  tendsto_mult
      by fastforce
    ultimately have
      "(λx. (S x)*(C a)+(C x)*(S a)) -0→ 0+S a"
      using tendsto_add
      by fastforce
    moreover have
      "(λx. (S x)*(C a)+(C x)*(S a)) = (λx. S(x+a))"
      using Sxplusy
      by simp
    ultimately have "(λx. S(x+a)) -0→ S a"
      by simp
    then have "S -a→ S a"
      by (simp add: limit_subst)
    then show ?thesis unfolding isCont_def
      by simp
  qed
```

As C is just a shift of S, we can prove it is continuous using the continuity of S:

```
lemma C_cont: "isCont C a"
  proof (simp add: isCont_def)
  from S_cont have "S -(p+a)→ S(p+a)" unfolding
   isCont_def
    by simp
  then have "(λx. S(x+p)) -(p+a)-p→ S(p+a)"
    by (rule LIM_offset)
  moreover have "(λx. S(x+p)) = (λx. S(p+x))"
  proof
    fix x
    have 1:"(x + p) = (p + x)"
      by simp
    then show "S (x + p) = S (p + x)"
      by (subst 1) simp
  qed
  ultimately have "(λx. S(p+x)) -a→ S(p+a)"
    by auto
  then show "C -a→ C a" using Spplusx_eq_Cx
    by simp
qed
```

The proofs that were omitted (those concerning S and particularly finding a sequence that was an upper bound on S) turned out to be more intricate and messy, although they did follow similar lines to those of C. Since S is so similar to C (they are just shifts of the same curve) this is rather surprising. But it may be that the part of the curve at $x = 0$ is more complex in some sense for S than it is for C, and Robison's proof uses the unique properties of $x = 0$. It is possible that the full extent of their messiness was hidden before this formalisation, since Robison could have relied on intuition about what was provable. Or it may be that the messiness was known, and that is why the neater proof for C was given.

2.3 Uniqueness of S and C

Robison next proves uniqueness of S and C, which should mean that the axioms are sufficient to define such functions. However, we have no way of using of the axioms to prove existence[5]. It is impossible to remove the assumptions of the locale from facts proved using these assumptions, just as attempting to prove P by assuming P will really lead to a proof of $P \to P$. Yet in order to instantiate the locale, we must choose a uniquely determined function, which we do not have.

To make this clearer, we can express Robison's theorem in predicate logic. Let P be the property that S satisfies (i.e. the axioms we specified above). Then Robison's theorem states

$$\forall s, \forall f, (Ps \wedge Pf) \to f = s.$$

[5]At least in Isabelle where metamathematical arguments are separated from the standard mathematical arguments, and require deep embedding inside the HOL logic.

Now we choose a constant sin in place of the variable S, which is analogous to instantiating the locale

$$\forall F, (P \sin \wedge P F) \rightarrow F = \sin .$$

If $P \sin$ evaluates to 'true', we have

$$\forall F, P F \rightarrow F = \sin .$$

The problem is we do not have such a sin for which $P \sin$ is true. Even if we did, then firstly, we would have no need of defining sine using these axioms, and secondly, the theorem would only say that 'these axioms are sufficient to determine sine', which is interesting metamathematically, but otherwise useless. Thus there seems little point in formalising this theorem, and we move on to properties of $S(x)/x$.

Robison also states that T, a generalisation of the tangent function, can be uniquely specified by similar axioms. He leaves the proof as an exercise to the reader. Of course, once we have S and C, it is enough to define $T(x) = S(x)/C(x)$, so we omitted the proof of this theorem in our formalisation.

2.4 Properties of $S(x)/x$

The culmination of Robison's paper is to define π, and thus sine and cosine, and then to show that $\lim_{x\to 0}(\sin x)/x = 1$. These will be proved in terms of $S(x)/x$ so first we must establish some key properties of this function. Robison takes $p = 1$ for this section of the paper and defines $F(x) = S(x)/x$ for $0 < x \leq 1$. However, Isabelle/HOL[6] only has a notion of total functions, so we simply define

```
definition F :: "real ⇒ real" where "F = (λx. S x/x)"
```

and wherever we prove a lemma concerning F, we put $0 < x \leq 1$ as an assumption.

2.4.1 $S(nx)/nx$ is a decreasing sequence

First Robison proves

Lemma 2.4.1. *For any x and positive integer n such that $0 < x < nx \leq 1$, the sequence $F(x)$, $F(2x)$, ..., $F(nx)$ decreases.*

We formalise this as

```
lemma F_seq_dec: assumes p_one: "p=1" and "0<x"
  shows "⟦x<(n::nat)*(x::real);n*x≤1⟧
              ⟹ F(n*x)<F((n-(1::real))*x)"
```

which is an unusual mix of Isar style `assumes "..."and "..."shows "..."` and older style `"⟦ ... ⟧ ⟹..."`. Usually the two styles would have an equivalent

[6]This refers to Isabelle with higher-order logic. Other logics can also be used, but currently most of the mathematics formalised in Isabelle is formalised in HOL.

effect. Here the mix is required because we wish to do an inductive proof which also inducts on some of the assumptions, and we must put those assumptions into the `shows "..."` part so that Isabelle knows to induct on them.

Robison's proof is by induction, beginning with the base case $n = 2$, and then moving on to prove the inductive step. However, there are two significant problems with his proof. The first is the way he proves the base case:

Let $n = 2$.

$$F(2x) = \frac{S(2x)}{2x} = [F(x)]C(x) < F(x).$$

In the last step we notice that we need $C(x) < 1$ for $0 < x \leq 1$. As $p = 1$, this is true, but it has not yet been proven, and as we shall find, the proof is nontrivial, at least by the standards of Robison's paper. Slightly less obviously, the last step also requires $F(x) > 0$ since $0 \cdot C(x)) \not< 0 \cdot 1$. This is also nontrivial to prove.

None of Robison's lemmas can possibly give a very straightforward proof of $C(x) < 1$ since he has only dealt with non-strict inequalities so far. However, we can emulate the method he used for proving that $lim_{n\to\infty} C(p/2^n) = 1$. There he showed that if $C(x) \geq 1 - c$ then $C(x/2) \geq c/2$. Here we show that $C(x) < 1$ implies $C(x/2) < 1$. We induct this to get

```
lemma C_strict_less_one_inducted:
assumes x_greater_0: "0<x"
    and x_less_p:"x≤ p" and Cx_less_1:"C x < 1"
    shows "C (x/(2^n)) < 1"
```

We also show that every real number greater than one is 'sandwiched' between two consecutive powers of 2.

```
lemma twopowers_sandwich: assumes "1≤(a::real)"
       shows "∃(n::nat). (2^n ≤ a) ∧ (a < 2^(n+1))"
```

These lemmas, along with the theorem that C is nondecreasing on $(0, p]$, are enough to prove $C(x) < 1$ for $0 < x \leq 1$:

```
lemma C_strict_less_one: assumes x_gr_0:"0<x"
                         and "x≤ p"
                         shows "C x<1"
proof -
  consider (a) "x<p/2" | (b) "p/2≤ x"
    by linarith
  then show ?thesis
  proof cases
    case a
    assume "x<p/2"
    then have "p/x ≥ 1"
      using x_gr_0 by auto
    then obtain n where "2^(n+1) > p/x" and "p/x≥2^n"
      using twopowers_sandwich by blast
    from `p/x≥2^n` have "p≥2^n*x"
      using x_gr_0 by (simp only: pos_le_divide_eq)
    moreover from `2^(n+1) > p/x` have "2^n*x≥p/2"
      using x_gr_0 by (auto simp: field_simps)
    ultimately have "C(p/2 + (2^n*x-p/2)) ≤C (p/2)"
      by (subst C_nonincreasing_on_0_p) auto
    then have "C(2^n*x) ≤C (p/2)"
      by simp
    also have "... = sqrt 2 / 2"
      by (rule Spdiv2_Cpdiv2(2))
    finally have "C(2^n*x) < 1"
      using sqrt2_less_2 by linarith
    from `2^n*x≥p/2` p_gr_0 `p≥2^n*x` this have "C((2^n*x)
   / 2 ^ n) < 1"
      by (subst C_strict_less_one_inducted) auto
    then show "C(x) < 1"
      by auto
  next
    case b
    assume "p/2≤x"
    from p_gr_0 assms(2) this have "C(p/2 + (x-p/2)) ≤C
    (p/2)"
      by (subst C_nonincreasing_on_0_p) auto
    then have "C(x) ≤C (p/2)"
      by simp
    also have "... = sqrt 2 / 2"
      by (rule Spdiv2_Cpdiv2(2))
    finally show "C(x) < 1"
      using sqrt2_less_2 by linarith
  qed
qed
```

We can then prove $S(x) > 0$ for $0 < x \leq 1$ in an analogous way, and since

$0 < x$ this easily gives us $F(x) > 0$. So we are able to formalise Robison's proof of the $n = 2$ base case of Lemma 2.4.1. It seems clear that Robison must have simply forgotten to prove that $C(x) < 1$ for $0 < x \leq 1$ as he proved all other lemmas of similar difficulty. It is also relatively intuitive, and since he proved many lemmas, it would be easy to lose track of what he had and hadn't proven. Robinson must at least have been aware that he was using $C(x) < 1$ as a fact, otherwise he would not have thought of this proof. But we may doubt that he noticed that $F(x) > 0$ was needed as well.

The second problem with Robison's proof of Lemma 2.4.1 is subtle. Robinson is inducting a conditional. That means his induction step should be proving $0 < x < nx \leq 1 \rightarrow F(nx) < F((n-1)x)$ implies $0 < x < (n+1)x \leq 1 \rightarrow F((n+1)x) < F(nx)$. But instead Robison only explicitly proves $F(nx) < F((n-1)x)$ implies $F((n+1)x) < F(nx)$. This leads him to overlook one detail. We will carefully go through how the induction step ought to be proved.

As this is an inductive step, we are allowed to assume $0 < x < nx \leq 1 \rightarrow F(nx) < F((n-1)x)$, and we must show $0 < x < (n+1)x \leq 1 \rightarrow F((n+1)x) < F(nx)$. To show $0 < x < (n+1)x \leq 1 \rightarrow F((n+1)x) < F(nx)$, we should first assume $0 < x < (n+1)x \leq 1$ and try to prove $F((n+1)x) < F(nx)$. So we end up with two assumptions

1. $0 < x < nx \leq 1 \rightarrow F(nx) < F((n-1)x)$
2. $0 < x < (n+1)x \leq 1$.

Now, Robison has proved that $F(nx) < F((n-1)x)$ implies $F((n+1)x) < F(nx)$. So we should try to get $F(nx) < F((n-1)x)$ and then we will be done. That means we should prove $0 < x < nx \leq 1$, since then modus ponens with the first assumption gives us what we want. Hence, we are left with proving $0 < x < (n+1)x \leq 1 \rightarrow 0 < x < nx \leq 1$. As $0 < x$ gives us $0 < x$, and $(n+1)x \leq 1$ gives us $nx \leq 1$, we must show $x < (n+1)x$ implies $x < nx$. This is true when $n > 2$, and not true when $n = 2$. It is also true when $n = 0$ and when $n = 1$ because $x < nx$ is false. So we need to assume $n \neq 2$ in the induction step. It is best to assume $n > 2$ and prove the cases $n = 0$ and $n = 1$ separately, since it looks odd to prove them in the induction step.

As we wish to assume $n > 2$ in the inductive step, we must prove the base case $n = 3$ as well as the base case $n = 2$ that Robison already proved. Fortunately, the method that Robison used for the inductive step also works for the case $n = 3$. It is quite probable that Robison did not notice that his proof requires the $n = 3$ base case to be valid, since the reasoning required to discover it is quite convoluted. It is possible, however, that Robison did notice that his proof required the extra base case, but decided to omit it since it was provable by the same method as the inductive step.

2.4.2 $S(x)/x$ is a decreasing function

Now that Robison has proved Lemma 2.4.1 he uses it to show the stronger statement

Theorem 2.4.2. *$F(x)$ is a decreasing function on (0,1].*

Formalising the proof in Isabelle mainly required rewriting such as showing that $(1/2^j)(1/2^k) = 1/2^{j+k}$ and proving various quantities to be nonzero. This lengthened the proof considerably. It would be easy to prove all these sorts of steps as lemmas and so make the proof more readable and closer to Robison's.

One step was interesting. Robison says

> ... Then for some positive integer n, $y - \delta < x_1 + n(2^{-(k+j)})x_0 < y$, by the Archimedean property.

The question is, which version of the Archimedean property is he using? The most familiar one is 'for all $a, b \in \mathbb{R}$ with $a > 0$, there is an $n \in \mathbb{N}$ such that $an > b$'. This does not suffice here, however, because it only gives an upper bound. None of the Archimedean properties that we could find already in Isabelle seemed appropriate. Instead we proved

```
lemma exists_n_strict_interval_length_gr_1:
  assumes "(b::real)>1" and "c≥0"
  shows "∃(n::nat). c< n ∧ n<c+b"
```

then by translating and scaling appropriately, this gives us the desired result. Robison was right to refer to this as the Archimedean property since its proof requires using the ceiling function, which is defined using another version of the Archimedean property.

We also found that Lemma 2.4.1 was not strong enough. Robison invoked the lemma twice, but the second time he actually needed an inducted version of it. The induction is straightforward, however it may have made his proof more readable if he had mentioned this explicitly, rather than leaving it for the reader to figure out. As Jean-Pierre Serre says in his famous lecture 'How to write mathematics badly' [30], badly-written mathematics makes the job as easy as possible for the writer whilst leaving it for the reader to do the heavy-lifting. We might infer that well-written mathematics explains what is going on as fully as possible for the reader. This is not meant as a criticism of Robison, whose proofs seem generally precise and well-presented, but only to support the idea that formal proof has, in its own way, some advantages in readability.

2.4.3 $S(x)/x$ has a limit as x goes to zero

Next Robison proves that $\lim_{x\to 0} F(x)$ exists. Robison uses two lemmas to aid in the proof of this theorem, the first is

Lemma 2.4.3. *For any nonnegative integer n*

$$\frac{2}{1 + C(2^{-n})} \leq \frac{2}{2 - 2^{-n}}.$$

He proves this by induction. However, since we had previously found it necessary to prove $C(p/2^n) \geq 1 - 1/2^n$ for all positive integer n, we used this to give a simpler proof of Lemma 2.4.3 (remember that we are now assuming $p = 1$).

Lemma 2.4.3 was only used to prove the second lemma:

Lemma 2.4.4. *The sequence $\{F(2^{-i})\}$, $i = 1, 2, ...$ has a limit.*

Robison gives a summary of his strategy

> *We shall use induction on $F^2(2^{-i})(= [F(2^{-i}]^2)$ to show that the sequence is bounded. Specifically, we show that $F^2(2^{-i}) \leq (2^{n+1})/(2^{n+1})/(2^{n-1} + 1) < 4$. By Theorem III (F is a decreasing function on $(0, 1]$), the sequence is increasing, so it will therefore have a limit.*

We follow his proof, with the only difference that we use induction on $F(2^{-i-1})$ rather than $F(2^{-i})$ since Isabelle's `induct i` method takes $i = 0$ as the base case, whereas Robison takes $i = 1$ as his base case.

His proof of this also uses the following property of F

$$F^2(x/2) = 2/(1 + C(x))F^2(x)$$

which he does not prove, saying it follows easily from previously proved lemmas. We prove it using the lemmas `Srecurrence` and `Crecurrence` (see Appendix .1.1) which were useful in the proofs that S and C were continuous.

Once these are proven, he simply instructs the reader to use Lemma 2.4.4 and that F is a decreasing function on $(0, 1]$ as his proof that $\lim_{x\to 0} F(x)$ exists. However, formalising this proof involved more than just referencing the two previous lemmas. Before we go through our formalisation, let us look at this proof from a higher-level perspective.

General strategy for showing existence of a limit of a function

The structure of this proof is very similar to the way Robison showed that C was continuous at 0: there he showed that a sequence of C values converged to a $C(0)$, and used the property that C is nonincreasing to find that $\lim_{x\to 0} C(x) = C(0)$, here he again shows that a sequence of F values converge to some L, and uses the property that F is decreasing to show that $\lim_{x\to 0} F(x)$ exists.

We can extend this to a general strategy to show that the limit of some f exists at a, from the right or left or both. First find a sequence X_n which converges to a as n tends to infinity (it does not need to be monotonic since there always exists a monotonic subsequence, but if we wish to find the limit from the right (or left), the sequence must have a monotonic subsequence that converges to a from the right (or left)). Robison chose a constant multiple of 2^n for both his proofs. Then show that $f(X_n)$ converges to some L (where $L = f(a)$ if we are proving continuity). Now, if f is monotonic close to a (on the right or the left or both), we can use this to show that, for an appropriate choice of $c = a^+$, a^- or a, we have $f(x) \to L$ as $x \to c$. Since such a general strategy is used three times in our formalisation of this paper (twice for continuity of S and C at 0, and once for the existence of a limit of F at 0), we could have proved a lemma, or several lemmas, that were equivalent to this strategy and used these same lemmas in each of the three cases. That might have shortened the proofs, but more importantly, it would have demonstrated the existence of the general strategy.

However, this only became clear with hindsight. Therefore in the rest of this section we are concerned with the formalisation of a specific instance of this strategy: proving that $\lim_{x\to 0} F(x)$ exists given that for any nonnegative integer n, we have $2/(1 + C(2^{-n})) \leq 2/(2 - 2^{-n})$ and that F is a decreasing function on $(0, 1]$.

What are we trying to prove?

We know (or think we know) that we have two theorems that can be brought together to show a third theorem. What do we do next? We go back to definitions, strip away notation until we are sure of what it is we want to prove and what we can use to prove it. If it is still nonobvious, we can think about what each theorem means, and why they might entail the third. We can imagine the first or second theorem wasn't true after all and consider what might go wrong with the third theorem.

First, we begin by understanding what we need to prove.

The sequence has the same limit as the function

Although Robison only proves that $\lim_{x\to 0} F(x)$ exists, we realised that $\lim_{x\to 0} F(x) = \lim_{n\to\infty} F(2^{-n})$ and we decided to prove this stronger statement. From now on we will call this limit `F_lim` as it is called in our Isabelle theory. We already know that $\lim_{n\to\infty} F(2^{-n})$ exists but for it to be well-defined it must also be unique. Mathematicians are so used to the idea that limits are unique that usually it is not even mentioned. However, we prove this explicitly.

```
lemma F_limit_unique: assumes "p=1"
        shows " ∃!L.(λn. F(1/2ⁿ⁺¹)) ⟶ L"
  using LIMSEQ_unique F_seq_has_limit assms by blast
```

Now that we know it exists and is unique, we can define `F_lim` to be the limit that $F(1/2^{n+1})$ tends to as n goes to infinity. As limits are only defined implicitly in Isabelle, we need to use a choice operator `THE` to specify it.

```
definition F_lim :: "real"  where
"F_lim = (THE L. (λ n. F(1/2ⁿ⁺¹)) ⟶ L)"
```

We have to explicitly show that `F_lim` satisfies what we would really think of as its definition. This is just a matter of putting together the facts that we already showed concerning existence and uniqueness

```
lemma F_lim_bestdef: assumes "p=1"
                       shows "(λ n. F(1/2ⁿ⁺¹)) ⟶ F_lim"
 by (subst F_lim_def,rule theI',rule F_limit_unique,
   rule assms)
```

Note that the lemma carries the assumption $p = 1$. Nearly all of the following lemmas carry this assumption. If we want to avoid writing this assumption before every lemma, we could begin another locale inside the original one. However, it is not necessary for every lemma.

Right-hand limits

Recall that Robison's definition of F restricts its domain to $(0, 1]$. When defining F in Isabelle however, we did not make this restriction and instead defined F on the whole real line. Now this means that, where Robison can accurately refer to $\lim_{x\to 0} F(x)$ as though it is not a one-sided limit, since the restriction to positive x was made when defining the domain, we, however, are obliged to define a one-sided limit[7].

There is a notion of one-sided limits in Isabelle, e.g. for the limit of $F(x)$ as x goes to 0 from the right we can write

```
THE L. (F ⟶ L) (at 0 within {0..}).
```

However, `at 0 within {0..}` is defined in terms of general notions in a topological space, which potentially do not yet have their nonstandard analysis counterparts defined. In order to use the Transfer Principle on them, it would require unfolding until we reached more basic concepts to which the Transfer Principle could be applied.

Instead we will use what is an equivalent definition for the reals

```
THE L. ∀r>0. ⟶
         (∃s>0. ∀x. x>0 ∧ dist x 0<s ⟶ dist (F x) L<r)
  .
```

We can always prove the definitions are equivalent later. We might be worried that this will not give us what we want since $\sin\epsilon \simeq \epsilon$ for infinitesimal ϵ isn't 'one sided'. But we can use symmetry of sine around 0 and prove it for 0 separately.

Planning a proof

It is generally useless to carry out details without having seen the main connection, or having made a sort of plan.

- George Pólya, *How to Solve It, p. 6*

Consider for now what we need to prove and what we have, without the quantifiers or assumptions, which make it more obscure to interpret. From `F_lim_bestdef` we have

```
¦ F (1 / 2 ^ (n + 1)) - F_lim¦ < r
```

for any natural number `n` and `r`. From `F_decreasing_on_0_1` we have

```
"F y < F x"
```

for any `x` and `y` with `x<y`. Without yet knowing whether we are making a valid substitution, we could let `y = 1 / 2 ^ (n + 1)` in this

```
"F (1 / 2 ^ (n + 1)) < F x"
```

[7]Actually, it would be possible to define an ordinary two-sided limit for F and, although it would require reproving some lemmas, this could easily be done using the symmetry of sine. In fact, we took this route when we proved continuity of sine and cosine at 0 (see Section 2.2.1), where again Robison had restricted their domains. However, a one-sided limit follows Robison's reasoning more closely here.

Now we see that, since `F (1 / 2 ^ (n + 1))` is greater than zero, if we have `F x ≤F_lim` then we can put these together to get

```
¦ F x - F_lim¦ < r
```

which is what we wanted to prove. Since F is decreasing, we should be able to prove `F(x) ≤ F_lim` for any x in the domain of F. But since this is behaviour that occurs at the limit, it will require more than straightforward application of the previously proved lemma that F is decreasing. We formalise and prove this lemma before proceeding with our proof that $\lim_{x\to 0} F(x) =$ `F_lim`:

```
lemma F_lim_bound: assumes "x>(0::real)" and "x≤1"
                   and "p=1"
                   shows "F x ≤ F_lim"
proof -
  obtain n::nat where "n>1/x" using `x>0`
   ex_less_of_nat
    by auto
  have "2^(n+1) ≥ (1::real)"
    by (rule two_realpow_ge_one)
  then have "1/2^(n+1) > (0::real)"
    by auto
  from `x>0` `2^(n+1) ≥ (1::real)`
    have "0<( 2^(n+1))*(1/x)"
    by simp
  have "(2::real)^(n+1) > (n+1)"
    by (rule of_nat_less_two_power)
  from `n>1/x` this have "2^(n+1) > 1/x"
    by linarith
  from this zero_less_one `0<( 2^(n+1))*(1/x)`
    have "1 / 2^(n+1) < 1/(1/x)"
    by (rule divide_strict_left_mono)
  from `x>0` have "x=1/(1/x)" by simp
  from `1 / 2^(n+1) < 1/(1/x)`
    have "1 / 2^(n+1) < x"
    by (subst `x=1/(1/x)`)
  from `0<1/2^(n+1)` this `x ≤ 1` `p=1`
    have that: " F x < F (1 / 2^(n+1))"
    by (rule F_decreasing_on_0_1)
  from assms have "F (1 / 2^(n+1)) ≤ F_lim"
    by (rule F_limseq_bound)
  from this that show "F x ≤ F_lim"
    by linarith
qed
```

Carrying out a proof

Now that we have everything we need, we can prove $\lim_{x\to 0} F(x) =$ `F_lim`. In Isabelle, in order to strip away the notation and quantifiers in whatever we are trying to prove, we can begin our proof by applying several methods to the statement, e.g.

```
lemma F_limit: assumes "p=1"
  shows "∀ r>0. ∃ s>0. ∀ x. x>0 ∧ dist x 0<s
                                        ⟶ dist (F x) F_lim<r"
proof (safe,subst dist_real_def, subst dist_real_def)
```

Each method is separated from the others by a comma. First we use `safe`. This method breaks down any classical logic terms by using safe rules, i.e. rules that cannot turn a provable goal into an unprovable one. It will not apply `rule exI` for example, which would remove the existential quantifier, and replaces any variables bound by it with a schematic variable. Whenever we use a `rule` *lemma* method, it first checks if the current goal matches the conclusion of the previously proved *lemma*. If it does (with suitable variable instantiations) then it replaces the current goal with the assumptions of *lemma*. After applying `safe`, the current goal becomes

```
 "⋀r. 0<r ⟹ ∃s>0. ∀x. 0<x ∧ dist x 0<s
                                       ⟶ dist (F x) F_lim<r".
```

Here $\bigwedge$ is the meta-level version of the universal quantifier. Informally, we can think of it as the same as a universal quantifier but easier to instantiate. Similarly $\Longrightarrow$ is meta-level implication. Next, `dist_real_def` is a lemma referring to the definition of the standard metric on the reals

```
lemma dist_real_def: dist x y = ¦ x - y ¦
```

so the two applications of `subst dist_real_def` replace `dist x y` with its specialisation to the reals. Now our current goal is

```
 "⋀r. 0<r ⟹ ∃s>0. ∀x. 0<x ∧ ¦x-0¦<s
                                          ⟶ ¦(F x)-F_lim¦< r"
```

and having massaged it enough, we are ready to start our proof. This goal seems simpler than our original one. We can see the correspondence here with the way a mathematician might state 'If we show this, we are done' or 'It is enough to prove this'. After this, it is just a matter of following the plan we developed in 2.4.3. The proof is given in Appendix .2.

2.5 Defining Sine, Cosine and Π

Once Robison has shown the existence of $\lim_{x\to 0} F(x)$, he uses it to define π[8], cosine and sine. We named them `Sin`, `Cos` and `Pi` to avoid conflict with the `sin`, `cos` and `pi` already defined in Isabelle. They are given by

```
definition Pi :: "real" where "Pi = 2*F_lim"
```

and

[8]We might conjecture that $\lim_{x\to 0} F(x)$ is $\pi/2$ for any positive value of p, and that Robison chose $p = 1$ merely because it was convenient. It would be interesting to repeat the proofs in the last section with a general value of p and see if the same results can be proved.

```
definition Sin :: "real ⇒ real"
                     where "Sin = (λx. S ((2*x) / Pi))"
```

and

```
definition Cos :: "real ⇒ real"
                     where "Cos = (λx. C ((2*x) / Pi))"
```

We can easily get a lower bound on the value of `Pi`[9]

```
lemma Pi_greq_2: assumes "p = 1" shows "Pi≥2"
proof -
  have "F 1 = 1" using Sp_eq_1 F_def `p = 1` by simp
  moreover from `p = 1` have "F_lim ≥ F 1" by (simp
   only: F_lim_bound)
  ultimately have "F_lim ≥1" by linarith
  then show "Pi≥2" by (simp only: Pi_def)
qed
```

Robison states that if we assume `p=Pi/2` we can verify that `Sin` and `Cos` satisfy the axioms given at the start. We are able to carry out this verification, but our lemmas must assume `p=1` (at least those that use the property that `Pi≥0` and we of course cannot assume `p=Pi/2` at the same time. But we are not required to do this: all we must show is that `Pi/2` assumes the same role for `Sin` and `Cos` that `p` has for `S` and `C`. This ambiguity in Robison's statement would probably never be noticed without the context of the formalisation.

We cannot instantiate the locale to these verified axioms, as we would like in order to get all the theorems that must be true for sine and cosine. This would lead to infinite recursion[10] as the locale would be instantiated within itself, and then within the one within itself etc.. What we can do is copy-paste any proofs that are valid for S, C and p, and replace them with their counterparts sin, cos and $\pi/2$. Alternatively, we can expand sin, cos and $\pi/2$ into their definitions in terms of S, C and p and easily reprove the lemmas in the way that we verified sin, cos and $\pi/2$ satisfied the locale axioms.

2.5.1 The limit of $\sin(x)/x$

Finally, we can formalise Robison's last proof, that $\lim_{x\to 0}\sin(x)/x = 1$. Again, this is actually a one-sided limit. We are able to follow Robison's proof very closely, although the formalisation at first glance appears very different, because we use the expanded definition of a limit throughout, without the convenience of abbreviations such as lim. Robison's proof is given below, and it may be interesting to compare it with the Isabelle proof (see the last proof in Appendix .2).

[9]Deriving the power series of `Sin` and `Cos` indirectly proves that they are the `sin` and `cos` already defined in Isabelle. Once this is established, we can also be certain that `Pi` is equal to `pi`.

[10]Although it is not actually possible for this to happen in Isabelle: were are merely considering a hypothetical.

Theorem 2.5.1. $\lim_{x\to 0} \sin(x)/x = 1$.

Proof. $1 = (2/\pi)(\pi/2) = (2/\pi)\lim_{x\to 0} S(x)/x = (2/\pi)\lim_{x\to 0}(S(2x/\pi))/(2x/\pi)$ $= \lim_{x\to 0}(S(2x/\pi))/x = \lim_{x\to 0} \sin(x)/x$.

It is interesting that Robison gives this proof since it is really just a calculation, whereas he omitted proofs that were much more intricate (e.g. the proof that S is continuous, the proof that $C(x) < 1$). The reason may be that this was the last theorem in his paper, and he wished to end with a flourish.

Chapter 3

Conclusion

3.1 Future Work

We have now shown that $\lim(\sin x)/x = 1$. Formalisation of McKinzie and Tuckey's paper on Euler's derivation of the power series for sine and cosine requires a closely-related statement in nonstandard analysis: $\sin \epsilon$ is infinitely close to ϵ where ϵ is an infinitely small number. The Transfer Principle (described in Section 1.2.1) allows us to convert certain well-formed statements in standard analysis to nonstandard analysis. We cannot get this particular statement directly, since it concerns the infinitely-close relation. We may also require other properties of sin and cos, which will need to be transferred from standard to nonstandard analysis. We chose to follow Robison and prove the standard versions of the trigonometric theorems first so that this formalisation could be compared with Robison's reasoning.

At present, nearly all of the later lemmas have $p = 1$ as an assumption. That is, the lemmas concerning F, but more seriously, any lemmas concerning sin and cos which require $\pi > 0$. Since two of the 'axioms' of the locale, specialised to sin and cos, seem to require $\pi > 0$ to prove, this would probably end up being nearly all of the lemmas. It might be possible to repeat the proofs for F with a general positive p, since it seems that Robison may have chosen $p = 1$ simply for convenience. This would remove the need for further assumptions within the locale. Another option is to begin a new locale within the original one, that has $p = 1$ as an assumption. However, $p = 1$ ought to be thought of as the removal of an assumption. An alternative would be to end our locale just before we need to make the assumption $p = 1$. Then we would define a new locale

```
locale circular_functions2 =
  fixes S :: "real ⇒ real"
  fixes C :: "real ⇒ real"
  assumes Cxminusy : "C(x)*C(y) + S(x)*S(y) = C(x-y)"
    and   Sp_eq_1 [simp]: "S 1 = 1"
    and   Sx_greq_0  [intro]:
    " ⟦  1 ≥ x ; x ≥ 0 ⟧ ⟹ S x ≥ 0"
```

which is identical to the previous one, but with p replaced everywhere with 1. The assumption $p > 0$ is now not required since it is provable. This can be considered as partial instantiation of our locale with $p = 1$.

A related issue is whether, now that we have defined π, sin and cos, and have even verified that they satisfy the conditions given by the locale, we need to repeat all the proofs of the properties we proved for S and C. There is no possibility of instantiating the locale within itself. Even if we did reprove all the properties, this would only require expanding their definitions in terms of S and C, using the corresponding properties of S and C, and putting it back in terms of sin and cos. E.g., our proof of continuity for C was quite long. But given C is continuous, showing that $\cos = C(2x/\pi)$ is continuous will be very quick. If we took the option, described above, of beginning a new locale, not within the first, in which we assume $p = 1$, we will be able to instantiate the first locale, with all its properties of S and C, to sin and cos. We would also like to completely remove the assumptions of the locale. This requires finding a model for the axioms of the locale, and then instantiating it to the model. In this case, it means finding alternative definitions of any function with the properties of S and C. Since we are able to define π given an S and C of any periodicity, it is not necessary to specifically find a definition of sine and cosine. In theory at least, this should make the task a little easier.

Robison's paper gave defining properties of sine and cosine that did not depend on calculus or Euler's number e. It is unclear why this would be particularly useful for anything except our specific purpose, which was the derivation of the power series of sine and cosine. However our theory would be useful once sine and cosine have been defined using geometry or some other method, since so long as the three basic assumptions of the locale could be satisfied, all the properties of S and C proved in the locale will automatically be true for the new definition of sine and cosine. A geometric definition of sine and cosine would be something to work towards, especially for other formalisations of the reasoning of historical mathematicians, since before Euler, the only known characterisation of sine and cosine was in terms of geometry.

3.2 Introspection

It would be convenient if the proofs in the formalisation could be transferred fully formed from the mind to the machine, but in reality, it never happens that way. Very often, mistakes in the conception of the proof are discovered when a formalisation of it is attempted, because a step in the proof will be found to be unprovable, or even disprovable. However, sometimes the proof steps are mistaken not because they are unprovable, or disprovable, but for more subtle reasons. In that case, the proof can be formalised, and it is up to the human to notice the mistake.

There were at least two instances of this during this formalisation. One occurred when I wished to formalise the proof that $\lim_{x\to 0} C(x) = C(0)$. The definition of a limit of a function $f : \mathbb{R} \to \mathbb{R}$ at $a \in \mathbb{R}$ is

$$\forall \epsilon > 0, \exists \delta > 0, \forall x, (0 < |x - a| < \delta \to |f(x) - f(a)| < \epsilon)$$

and I was confused by the '$\forall \epsilon > 0$'. Intuitively, it seems that limits should not depend on 'large' ϵ. Therefore it should be enough to prove

$$\exists r > 0, \forall \epsilon, (r > \epsilon \wedge \epsilon > 0 \to \exists \delta > 0, \forall x, (0 < |x - a| < \delta \to |f(x) - f(a)| < \epsilon)).$$

I formalised the proof that these two definitions were equivalent. I then used the second definition to formalise the proof that $\lim_{x \to 0} C(x) = C(0)$. The proof required unfolding the definition completely. I later realised that the condition $r > \epsilon$ in the second definition was not used during this proof. Although the two definitions are equivalent, and so it would not be incorrect to use either, the increase in complication of the second definition had no practical use at all in the proof where I had used it. Thus I replaced it with the first definition, and shortened the proof.

Originally I formalised the proof that $\lim_{n \to \infty} C(p/2^n)) = 1$ using an innocuous-seeming logarithm in order to define the n for which $C(p/2^n)) - 1 < \epsilon$. However, logarithms in Isabelle are defined using Euler's number, e, which is defined using its power series, which is closely related to the power series of sine and cosine. So the argument was at risk of becoming circular. Jacques Fleuriot pointed this out and gave an alternative proof in terms of the Squeeze Theorem. This proof was much shorter, and therefore more readable. On second reflection it also seemed closer to what Robison was hinting at, since he gave a sequence and its limit between which $C(p/2^n))$ was sandwiched. This could be proved by going back to the definitions of a convergent sequence, but it was quite unnecessary. If you wish to prove something that seems like it might obey a general principle, or is in terms of a definition that needs to be unfolded, it is worth looking for general lemmas first, they are quite likely to have been proven already and will save a great deal of work.

3.3 Conclusion

The broader aim of this formalisation is to provide a definition of sine and cosine in Isabelle which could be used to formalise Euler's derivation of their power series. As the current definition in Isabelle is in terms of their power series we needed to find an alternative definition to avoid a circular argument. Definitions using calculus, which was avoided by Euler, or indirectly relying on the power series by using Euler's number e were unsuitable. Nor could we use a definition in terms of geometry, since the notion of angle has not yet been fully defined in Isabelle. We turned to a paper by Gerson Robison which gives three axioms that uniquely specify two functions S and C which depend on a parameter p and are identical to sine and cosine when $p = \pi/2$. We took these as assumptions for a locale, and by following Robison's paper, were able to formalise the proofs of many properties of S and C, including continuity. We did not think it was useful to show uniqueness of S and C within the locale, since it would not help remove the assumptions of the locale, and would not be helpful in proving anything else. Since his paper only used elementary real analysis, Robison often either omitted proofs entirely, or simply cited the lemmas which should be enough to prove the statement. For the proof that $\lim_{x\to 0} S(x)/x$ exists, we have four extra lemmas taking over seventy lines to formalise just one line of Robison's paper. This was not even the most extreme case, since our formalisation of the proofs that S and C were continuous at zero was much more intricate, whereas Robison simply indicated to use two lemmas for C and gave no indication for S. It was not always obvious which theorems he meant for us to use, e.g. in his proof that $\lim_{n\to\infty} C(p/2^n) = 1$ we believe he meant to use the Squeeze Theorem, but he does not explicitly say this. By contrast, the Isabelle proofs always make reference to the theorems or methods that are used. We also discovered that Robison had assumed in one place that $C(x) < 1$ for $x \in (0, 1]$ and that $S(x) > 0$ for $x \in (0, 1]$. By the standards of his paper, these were nontrivial to prove. Our proof of these was reminiscent of Robison's proof of continuity of S and C at 0. Formalising the limits took some care, as most of the limits in Robison's paper are actually one-sided limits. For the limit of S and C around 0, we decided to use the symmetry of these functions around 0 to extend Robison's one-sided limit to an ordinary two-sided limit. Then we could use their limits at 0 to show they were continuous under the normal definition in Isabelle. However, for the limit of $S(x)/x$ at 0, we formalised only the limit from the right, as it is in Robison's paper. Once we had formalised the existence of $\lim_{x\to 0} S(x)/x$, we were able to define π, sine and cosine in terms of this limit, S and C. The culmination of the formalisation was to show $\lim_{x\to 0} \sin(x)/x = 1$, which is a key property needed for formalising Euler's reasoning concerning sine and cosine.

.1 Robison's Lemmas

Here we give a complete[1] list of the theorems and lemmas Robison proved or mentioned in his paper [28], in the order that he mentioned them[2]. We give the names of the corresponding lemmas in our formalisation of this paper in Isabelle, `Circular_Functions.thy`. However, there are many more lemmas in the formalisation than these.

.1.1 Lemmas concerning S and C

`C0_eq_1`: $C(0) = 1$.

`Cminusx`: $C(-x) = C(x)$.

`Cx_sq_plus_Sx_sq_eq_1`: $(C(x))^2 + (S(x))^2 = 1$.

`Sx_bounds`: $0 \leq S(x) \leq 1$ for all $x \in [0, p]$.

`S0_eq_Cp_eq_0`: $S(0) = C(p) = 0$.

`Cpminus_x`: $C(p - x) = S(x)$.

`Spminusx`: $S(p - x) = C(x)$.

`Spdiv2_eq_Cpdiv2`: $S(p/2) = C(p/2)$.

`Cx_bounds`: $0 \leq C(x) \leq 1$ for all $x \in [0, p]$.

`Sxplusy`: $S(x + y) = S(x)C(y) + C(x)S(y)$.

`S2x`: $S(2x) = 2S(x)C(x)$.

`S2p`: $S(2p) = 0$.

`Fpdiv2_eq_1`: $2(C(p/2))^2 = (S(p/2))^2 = 1$.

`Spdiv2_Cpdiv2`: $C(p/2) = S(p/2) = \sqrt{2}/2$.

`Spdiv2_minus`: $S(-p/2) = -S(p/2) = -\sqrt{2}/2$.

`Sminusp`: $S(-p) = -1$.

`C2p_eq_minus1`: $C(2p) = -1$.

`Sminusx`: $S(-x) = -S(x)$.

`Cxplusy`: $C(x + y) = C(x)C(y) - S(x)S(y)$.

`C2x`: $C(2x) = (C(x))^2 - (S(x))^2 = 2(C(x))^2 - 1$.

[1]We omit the theorem concerning T as it is not relevant.

[2]Robison also implicitly assumed some lemmas of a similar level which he did not prove. There are also lemmas which he proved for C, but did not prove the corresponding ones for S.

`Crecurrence`: $(C(x)+1)/2 = (C(x/2))^2$.

`Sxminusy`: $S(x-y) = S(x)C(y) - C(x)S(y)$.

`Spplusx_eq_Cx`: $S(p+x) = C(x)$.

`Cpplusx_eq_minusSx`: $C(p+x) = -S(x)$.

`S2pplusx_eq_minusSx`: $S(2p+x) = -S(x)$.

`C2pplusx_eq_minusCx`: $C(2p+x) = -C(x)$.

`S4pplusx_eq_Sx`: $S(4p+x) = S(x)$.

`C4pplusx_eq_Cx`: $C(4p+x) = C(x)$.

`C_nonincreasing_on_0_p`: C is nonincreasing on $[0,p]$.

`S_nondecreasing_on_0_p`: S is nondecreasing on $[0,p]$.

`Cseq_goes_to_1`: For any constant c, if $C(x) \geq 1-c$, then $C(x/2) \geq 1-c/2$.

`limCpdiv2'`: $\lim_{n\to\infty}(p/2^n) = 1$.

`C_continuous_0`: C is continuous at 0.

`S_continuous_0`: S is continuous at 0.

`S_cont` and `C_cont`: S and C are continuous.

(no equivalent): S and C are uniquely defined on $[0,p]$.

.1.2 Lemmas concerning F

Hereafter Robinson assumes $p = 1$.

`F_cont`: $F(x)$ is continuous on $(0,1]$.

`F_seq_dec`: For any x and positive integer n such that $0 < x < nx \leq 1$, the sequence $F(x), F(2x), ..., F(nx)$ decreases.

`F_decreasing_on_0_1`: $F(x)$ is a decreasing function on $(0,1]$.

`bound_on_C_frac_seq`: For any nonnegative integer n

$$\frac{2}{1+C(2^{-n})} \leq \frac{2}{2-2^{-n}}.$$

`Frecurrence`: $F^2(x/2) = 2/(1+C(x))F^2(x)$.

`F_seq_has_limit`: The sequence $\{F(2^{-i})\}, i = 1, 2, ...$ has a limit.

.1.3 The limit of $\sin(x)/x$

Once sine and cosine are defined in terms of S and C, Robison proves

`lim_sin_x_over_x`: $\lim_{x\to 0}\sin(x)/x = 1$.

.2 The Circular Functions Theory

```
(*author: Imogen I. Morris with indicated contributions
    by Jacques D. Fleuriot*)
theory Circular_Functions imports Complex_Main begin


locale circular_functions =
  fixes S :: "real ⇒ real"
  fixes C :: "real ⇒ real"
  fixes p :: "real"
  assumes Cxminusy : "C(x)*C(y) + S(x)*S(y) = C(x-y)"
    and   Sp_eq_1 [simp]: "S p = 1"
    and   Sx_greq_0  [intro]:
          "⟦p ≥ x ; x ≥ 0⟧ ⟹ S x ≥ 0"
    and   p_gr_0: "p > 0"

context circular_functions begin

lemma C0_eq_1 [simp]: "C 0 = 1"
  proof -
    have "C 0 = C(p-p)" by simp
    also have "... = C p *C p + S p *S p"
      by (subst Cxminusy) arith
    also have "... = C p * C p + 1" by simp
    also have "... = (C p)^2 + 1" by algebra
    finally have "C 0 ≥ 1" by auto
    have "C 0 = C(0-0)" by simp
    also have "... = C 0 *C 0 + S 0 *S 0"
      by (subst Cxminusy) arith
    also have "... = C 0 *C 0 + (S 0)^2" by algebra
    also have "... ≥ C 0 *C 0" by auto
    finally have "C 0 ≥ C 0 *C 0" by auto
    then have "1 ≥ C 0" using `C 0 ≥ 1` by auto
    then show "C 0 = 1" using `C 0 ≥ 1` by auto
  qed

lemma S0_eq_Cp_eq_0:
  shows S0_eq_Cp:"S 0 = C p"
    and Cp_eq_0[simp]:"C p = 0"
    and S0_eq_0[simp]: "S 0 = 0"
  proof -
    have "C(p)*C(0) + S(p)*S(0) = C(p-0)"
      by (rule Cxminusy)
    then have "C(p) + S(p)*S(0) = C(p-0)" by simp
    then have "C(p) + S(0) = C p" by simp
    then show 1: "S 0 = 0" by simp
    have "C(p)*C(p) + S(p)*S(p) = C(p-p)"
      by (rule Cxminusy)
    then have "C(p)*C(p) + 1 = 1" by simp
    then show 2: "C p = 0" by simp
    show "S 0 = C p" using 1 and 2 by simp
  qed


lemma Cminusx: "C(-x) = C x"
  proof -
   have  "C 0 *C(x) + S(0)*S(x) = C(0-x)"
     by (rule Cxminusy)
   then have "C(x) + S(0)*S(x) = C(0-x)" by simp
   then have "C(x) + 0*S(x) = C(0-x)" by simp
   then show "C(-x) = C(x)" by simp
  qed

lemma Cx_sq_plus_Sx_sq_eq_1: "(C x)² + (S x)² = 1"
  proof -
    have "C(x)*C(x) + S(x)*S(x) = C(x-x)"
      by (simp add: Cxminusy)
    then have "C(x)*C(x) + S(x)*S(x) = 1" by simp
    then show ?thesis by algebra
  qed

lemma  Sx_bounds: shows "⟦p ≥ x ; x ≥ 0⟧ ⟹ S x ≥ 0"
    and Sx_leq_1:"⟦p ≥ x ; x ≥ 0⟧ ⟹ 1 ≥ S x"
  proof -
    show "⟦p ≥ x ; x ≥ 0⟧ ⟹ S x ≥ 0" by auto
  next
    have "(C x)² + (S x)² = 1"
      by (rule Cx_sq_plus_Sx_sq_eq_1)
    moreover then have " (S x)² = 1 - (C x)²"
      by algebra
    moreover have "(C x)² ≥ 0" by simp
    ultimately have "1 ≥ (S x)²" by simp
    then show "1 ≥ S x"
      by (simp add: power2_le_imp_le)
  qed

lemma Cpminusx: "C(p-x) = S x"
```

```
  proof -
    have "C (p-x) = C p* C x + S p * S x"
      by (subst Cxminusy) simp
    then show "C(p-x) = S x" by simp
  qed

lemma Spminusx: "S(p-x) = C x"
  proof -
    have "C x = C (p-(p-x))" by simp
    also have "... = S(p-x)" by (rule Cpminusx)
    finally show ?thesis by simp
  qed

lemma Spdiv2_eq_Cpdiv2: "S(p/2) = C(p/2)"
  proof -
    have "S(p - p/2) = C(p/2)" by (rule Spminusx)
    then show ?thesis by simp
  qed

lemma Cx_bounds:
    shows Cx_greq_0:"⟦p ≥ x ; x ≥ 0⟧ ⟹ C x ≥ 0"
    and Cx_leq_1:"⟦p ≥ x ; x ≥ 0⟧ ⟹ 1 ≥ C x"
  proof -
    assume a0: "p ≥ x"
    assume a1: "x ≥ 0"
    def  y == "p-x"
    have "p ≥ y" using y_def and a1 by auto
    moreover have "y ≥ 0" using y_def and a0 by auto
    ultimately  have 1:"S y ≥ 0" by (rule Sx_bounds)
    have 2:"S y ≤ 1" using `p ≥ y` and `y ≥ 0`
      by (rule Sx_bounds)
    have "S y = C x" using y_def
      by (simp add: Spminusx)
    then show "C x ≥ 0" using 1 by simp
    from `S y = C x` show "1 ≥ C x" using 2 by simp
  qed

lemma Sxplusy: "S(x)*C(y)+C(x)*S(y) = S(x+y)"
  proof -
    have 1:"(p-(x+y)) = ((p-x)-y)" by simp
    have "S(x+y) = C(p-(x+y))"
      by (subst Cpminusx) simp
    also then have "C(p-(x+y)) = C((p-x)-y)"
      by (subst 1) simp
    also have "C((p-x)-y) = C(p-x)*C(y)+S(p-x)*S(y)"
      by (subst Cxminusy) simp
    also have "... = S(x)*C(y)+S(p-x)*S(y)"
```

```
      by (subst Cpminusx) simp
    also have "... = S(x)*C(y)+C(x)*S(y)"
      by (subst Spminusx) simp
    finally show ?thesis by simp
  qed

lemma S2x: "S (2*x) = 2*S x*C x"
  proof -
    have "S(2*x)=S(x+x)" by simp
    also have "...= S(x)*C(x)+C(x)*S(x)"
      by (simp add: Sxplusy)
    also have "... = 2*S(x)*C(x)" by algebra
    finally show ?thesis by simp
  qed

lemma S2p[simp]: "S(2*p) = 0"
  proof -
    have "S(2*p) = 2*S(p)*C(p)" by (rule S2x)
    also have "...= 0" by simp
    finally show ?thesis by simp
  qed

lemma Fpdiv2_eq_1[simp]:
  shows "2*(C(p/2))² = 1" and "2*(S(p/2))² = 1"
  proof -
    have " 1 = S p" by simp
    also have "S p = S(2*(p/2))" by simp
    also have "S(2*(p/2)) = 2*S(p/2)*C(p/2)"
      by (subst S2x) simp
    also have "...= 2*C(p/2)*C(p/2)"
      by (subst Spdiv2_eq_Cpdiv2) simp
    finally show "2*(C(p/2))² = 1" by algebra
    have " 1 = S p" by simp
    also have "S p = S(2*(p/2))" by simp
    also have "S(2*(p/2)) = 2*S(p/2)*C(p/2)"
      by (subst S2x) simp
    also have "...= 2*S(p/2)*S(p/2)"
      by (subst Spdiv2_eq_Cpdiv2) simp
    finally show "2*(S(p/2))² = 1" by algebra
  qed

lemma Spdiv2_Cpdiv2[simp]:
  shows "S(p/2) = (sqrt 2)/2"
    and "C(p/2) = (sqrt 2)/2"
  proof -
    have "2*(C(p/2))² = 1" by (rule Fpdiv2_eq_1)
    then have "(C(p/2))² = 1/2" by algebra
    then have "sqrt ((C(p/2))²) = sqrt (1/2)"
      by (subst NthRoot.real_sqrt_eq_iff)
    have "sqrt ((C(p/2))²) = ¦C(p/2)¦"
      by (rule NthRoot.real_sqrt_abs)
    have "p≥p/2" and "p/2≥0" using p_gr_0 by auto
    then have "C(p/2)≥0" by (rule Cx_greq_0)
    then have "¦C(p/2)¦ = C(p/2)" by simp
    from `sqrt ((C(p/2))²) = ¦C(p/2)¦`
      and `sqrt ((C(p/2))²) = sqrt (1/2)` and this
      have "C(p/2) = sqrt (1/2)" by simp
    then have "C(p/2) = (sqrt 1)/(sqrt 2)"
      by (simp add: real_sqrt_divide)
    then have "C(p/2) = 1/(sqrt 2)"
      by (simp add: real_sqrt_divide)
    also have "... = (sqrt 2)*1/((sqrt 2)*(sqrt 2))"
      by (subst NthRoot.real_divide_square_eq) simp
    finally show "C(p/2) = (sqrt 2)/2" by simp
  next
    have "2*(S(p/2))² = 1" by (rule Fpdiv2_eq_1)
    then have "(S(p/2))² = 1/2" by algebra
    then have "sqrt ((S(p/2))²) = sqrt (1/2)"
      by (subst NthRoot.real_sqrt_eq_iff)
    have "sqrt ((S(p/2))²) = ¦S(p/2)¦"
      by (rule NthRoot.real_sqrt_abs)
    have "p≥p/2" and "p/2≥0" using p_gr_0 by auto
    then have "S(p/2)≥0" by (rule Sx_greq_0)
    then have "¦S(p/2)¦ = S(p/2)" by simp
    from `sqrt ((S(p/2))²) = ¦S(p/2)¦`
      and `sqrt ((S(p/2))²) = sqrt (1/2)` and this
      have "S(p/2) = sqrt (1/2)" by simp
    then have "S(p/2) = (sqrt 1)/(sqrt 2)"
      by (simp add: real_sqrt_divide)
    then have "S(p/2) = 1/(sqrt 2)"
      by (simp add: real_sqrt_divide)
    also have "... = (sqrt 2)*1/((sqrt 2)*(sqrt 2))"
      by (subst NthRoot.real_divide_square_eq) simp
    finally show "S(p/2) = (sqrt 2)/2" by simp
  qed

lemma Spdiv2_minus[simp]: shows "S(-p/2) = -(sqrt 2)/2"
                            and "-S(p/2) = -(sqrt 2)/2"
  proof -
   have  "S 0 = 0" by (rule S0_eq_Cp_eq_0)
   then have 1: "S(p/2+(-p/2)) = 0" by simp
   have "S(p/2)*C(-p/2)+C(p/2)*S(-p/2) = S(p/2+(-p/2))"
     by (rule Sxplusy)
   then have "S(p/2)*C(-p/2)+ C(p/2)*S(-p/2) = 0"
```

```
      using 1 by simp
    then have
      "((sqrt 2)/2)*((sqrt 2)/2)+((sqrt 2)/2)*S(-p/2)=0"
      by (simp add: Cminusx)
    then have
      "((sqrt 2)/2)*S(-p/2)=-((sqrt 2)/2)*((sqrt 2)/2)"
      by simp
    then have 2: "S (-p/2)*sqrt 2 = - 1"
      by (auto simp: algebra_simps)
    have "sqrt 2 ≠ 0" by simp
    then have "S(-p/2) = -1/(sqrt 2)" using 2
      by (rule Fields.division_ring_class.eq_divide_imp)
    also have "... = (sqrt 2)*(-1)/((sqrt 2)*(sqrt 2))"
      by (subst NthRoot.real_divide_square_eq) simp
    finally show "S(-p/2) = -(sqrt 2)/2" by simp
 next
    show "-S(p/2) = -(sqrt 2)/2" by simp
 qed


lemma Sminusp[simp]: "S(-p) = -1"
  proof -
    have "S(-p) = S(-p/2 + -p/2)" by simp
    also have "... = S(-p/2)*C(-p/2)+C(-p/2)*S(-p/2)"
      by (subst Sxplusy) simp
    also have "... = S(-p/2)*C(p/2)+C(p/2)*S(-p/2)"
      by (simp add: Cminusx)
    also have "...=
(-(sqrt 2)/2)*((sqrt 2)/2)+(-(sqrt 2)/2)*((sqrt 2)/2)"
      using Spdiv2_minus(1) by auto
    finally  show "S(-p) = -1" by simp
  qed


lemma C2p_eq_minus1[simp]: "C(2*p) = -1"
  proof-
    have "C(p -(-p)) = S(-p)" by (rule Cpminusx)
    then show "C(2*p) = -1" by simp
  qed

lemma Sminusx: "S(-x) = -S(x)"
  proof-
    have 1: "C(p-(-x)) = S(-x)" by (rule Cpminusx)
    have "C(p-(-x)) = C(-(p+x))"
      by (subst Cminusx) simp
    also have  "... = C(-p-x)" by simp
    also have "... = C(-p)*C(x) + S(-p)*S(x)"
```

```
      by (subst Cxminusy) simp
    finally have "C(p-(-x)) = -S(x)"
      by (simp add: Cminusx)
    then show ?thesis using 1 by simp
  qed

lemma Cxplusy: "C(x)*C(y) - S(x)*S(y) = C(x+y)"
  proof -
    have "C(x)*C(y)-S(x)*S(y)=C(x)*C(-y)+S(x)*S(-y)"
      by (simp add: Cminusx Sminusx)
    also have "... = C(x+y)" by (subst Cxminusy) simp
    finally show ?thesis by simp
  qed

lemma C2x: shows "(C x)² - (S x)² = C(2*x)"
                   and "2*(C x)² - 1= C(2*x)"
  proof -
    show 1:"(C x)² - (S x)² = C(2*x)"
      by (simp add: power2_eq_square Cxplusy)
    have "(C x)²+(S x)² = 1"
      by (rule Cx_sq_plus_Sx_sq_eq_1)
    then have "(S x)² = 1 -(C x)²" by algebra
    then have "(C x)² - (1 -(C x)²) = C(2*x)"
      using 1 by (rule subst)
    then show "2*(C x)² - 1= C(2*x)" by simp
  qed

lemma Crecurrence: "(C x + 1)/2 = (C(x/2))²"
  proof -
    have "2*(C(x/2))² - 1= C x" by (simp add: C2x)
    then have "2*(C(x/2))²= C x + 1" by linarith
    then have "(C(x/2))²= (C x + 1)/2" by simp
    then show ?thesis by simp
  qed

lemma Srecurrence: "4*((S(x/2))²)*(1-(S(x/2))²)=(S x)²"
  proof -
    have "2*S(x/2)*C(x/2) = S (2*(x/2))"
      by (rule S2x[symmetric])
    then have "2*S(x/2)*C(x/2) = S x" by simp
    then have "(2*S(x/2)*C(x/2))² = (S x)²" by simp
    then have 1:"4*(S(x/2))²*(C(x/2))² = (S x)²"
      by (auto simp:power_mult_distrib)
    have 2: "C(x/2)² = 1 - S(x/2)²"
    proof -
      have "(C (x / 2))² + (S (x / 2))² = 1"
        using Cx_sq_plus_Sx_sq_eq_1 by simp
      then show ?thesis
        by algebra
    qed
    from 1 show ?thesis by (subst 2[symmetric]) simp
  qed

lemma Sxminusy: "S(x)*C(y)-C(x)*S(y) = S(x-y)"
  proof -
    have "S(x-y) = S(x)*C(-y)+C(x)*S(-y)"
      by (subst Sxplusy) simp
    also have "... = S(x)*C(y)-C(x)*S(y)"
      by (simp add: Cminusx Sminusx)
    finally show ?thesis by simp
  qed

lemma Spplusx_eq_Cx: "S(p+x) = C x"
  proof -
    have "S(p+x) = S(p)*C(x)+C(p)*S(x)"
      by (subst Sxplusy) simp
    also have "... = C x" by simp
    finally show ?thesis by simp
  qed

lemma Cpplusx_eq_minusSx: "C(p+x) = - S x"
  proof -
    have "C(p+x) = C(p)*C(x)-S(p)*S(x)"
      by (subst Cxplusy) simp
    also have "... = - S x" by simp
    finally show ?thesis by simp
  qed

lemma S2pplusx_eq_minusSx: "S(2*p+x) = -S x"
  proof -
    have  "S(2*p+x) = S(p+(p+x))" by simp
    also have "... = C (p+x)" by (rule Spplusx_eq_Cx)
    also have "... = - S x"
      by (rule Cpplusx_eq_minusSx)
    finally show ?thesis by simp
  qed

lemma C2pplusx_eq_minusCx: "C(2*p+x) = -C x"
  proof -
    have "C(2*p+x) = C(p+(p+x))" by simp
    also have "... = - S (p+x)"
      by (rule Cpplusx_eq_minusSx)
    also have "... = - C x"
      by (subst Spplusx_eq_Cx) simp
    finally show ?thesis by simp
  qed

lemma S4pplusx_eq_Sx: "S(4*p+x) = S x"
  proof -
    have "S(4*p+x) = S(2*p+(2*p+x))" by simp
    also have "... = - S(2*p+x)"
      by (rule S2pplusx_eq_minusSx)
    also have "... = S x"
      by (subst S2pplusx_eq_minusSx) simp
    finally show ?thesis by simp
  qed

lemma C4pplusx_eq_Cx: "C(4*p+x) = C x"
  proof -
    have "C(4*p+x) = C(2*p+(2*p+x))" by simp
    also have "... = - C(2*p+x)"
      by (rule C2pplusx_eq_minusCx)
    also have "... = C x"
      by (subst C2pplusx_eq_minusCx) simp
    finally show ?thesis by simp
  qed

lemma C_nonincreasing_on_0_p:
    assumes (*"p ≥  x " and*) " x  ≥ 0"
    and "p ≥ x+y" (*and "x+y ≥ 0"*)
    and "y ≥  0 "
    shows "C(x+y) ≤ C x"
  proof -
    have y_in_range1: "p ≥ y"
      and y_in_range2: "y ≥ 0"
    proof -
      show "y ≥ 0" using assms(3) by simp
      show "p ≥ y" using assms(2) and assms(1) by simp
    qed
    have x_in_range1: "p ≥ x" and x_in_range2: "x ≥ 0"
    proof -
      show "x ≥ 0" using assms(1) by simp
      show "p ≥ x" using assms(2) and assms(3) by simp
    qed
    have "C(x+y) = C(x)*C(y)-S(x)*S(y)"
      by (subst Cxplusy) simp
    also have "... ≤ C(x)-S(x)*S(y)"
    proof -
      have "C x ≤ C x" by simp
      moreover have "C y ≤ 1"
        using y_in_range1 and y_in_range2
        by (simp add: Cx_leq_1)
      ultimately  have "C x ∗ C y ≤ C x"
        by (simp add: Cx_greq_0 mult_left_le
                      x_in_range1 x_in_range2)
      then show ?thesis by simp
    qed
    also have "... ≤ C(x)"
    proof -
      have "S(x) ≥ 0"
        by (simp add: x_in_range1 x_in_range2
                                    Sx_greq_0)
      moreover have "S(y) ≥ 0"
        by (simp add: y_in_range1 y_in_range2
                                    Sx_greq_0)
      ultimately show ?thesis by simp
    qed
    finally show ?thesis by simp
  qed


lemma S_nondecreasing_on_0_p:
    assumes (∗"p ≥  x " and∗) " x  ≥ 0"
    and "p ≥ x+y" (∗and "x+y ≥ 0"∗)
    and "y ≥  0 "
    shows "S(x+y) ≥ S x"
  proof -
    have y_in_range1: "p ≥ y"
      and y_in_range2: "y ≥ 0"
    proof -
      show "y ≥ 0" using assms(3) by simp
      show "p ≥ y" using assms(2) and assms(1)
        by simp
    qed
    have x_in_range1: "p ≥ x"
     and x_in_range2: "x ≥ 0"
    proof -
      show "x ≥ 0" using assms(1) by simp
      show "p ≥ x" using assms(2) and assms(3)
        by simp
    qed
    have "S(x+y) = C(p-(x+y))"
      by (subst Cpminusx) simp
    also have "...  ≥ C(p-x)"
    proof -
      have 1: "0 ≤ p -(x+y)" by (simp add: assms(2))
      have 2: "p-(x+y)+y ≤ p"
        by (simp add: x_in_range2)
      have "C(p-x) = C(p-(x+y)+y)" by simp
      also have "... ≤ C(p-(x+y))"
        using 1 and  2 and assms(3)
        by (rule C_nonincreasing_on_0_p)
      finally show ?thesis by simp
    qed
    also have " C(p-x) = S x"
      by (subst Cpminusx) simp
    finally show ?thesis by simp
  qed

lemma C_nondecreasing_on_minusp_0:
     assumes (*"-p ≤  x " and*) " x ≤ 0"
     and "-p ≤ x-y" (*and "x-y ≤ 0"*)
     and "y ≥  0 "
     shows "C(x-y) ≤ C x"
  proof -
    have 1:" -x ≥ 0" using assms(1) by simp
    have 2:"p ≥ - x + y" using assms(2) by simp
    have "C(-x+y) ≤ C (-x)" using 1 2 assms(3)
      by (rule C_nonincreasing_on_0_p)
    then have "C(-(-x+y)) ≤ C (-x)"
      by (subst Cminusx) simp
    then have "C(-(-x+y)) ≤ C x"
      by (simp add: Cminusx)
    then show "C(x-y) ≤ C x" by simp
  qed

lemma S_nondecreasing_on_minusp_0:
     assumes (*"-p ≤  x " and*) " x ≤ 0"
     and "-p ≤ x-y" (*and "x-y ≤ 0"*)
     and "y ≥  0 "
     shows "S(x-y) ≤ S x"
  proof -
    have 1:" -x ≥ 0" using assms(1) by simp
    have 2:"p ≥ - x + y" using assms(2) by simp
    have "S(-x+y) ≥ S (-x)" using 1 2 assms(3)
      by (rule S_nondecreasing_on_0_p)
    then have "-S(-(-x+y)) ≥ S (-x)"
      by (subst Sminusx) simp
    then have "S(-(-x+y)) ≤ S x"
      by (simp add: Sminusx)
    then show "S(x-y) ≤ S x" by simp
  qed

(*The following lemma is not quite as strong as it
   could be - however it seems necessary to first
```

```
 prove sine is continuous before we can prove the
   strong version*)
lemma S_nondec_lemma:
assumes x_in_range1: "p ≥ x" and x_in_range2: "x ≥ 0"
    and y_in_range1: "p ≥ y" and y_in_range2: "y ≥ 0"
  shows "S(x-y) ≤ S x"
  proof-
    have "S(x-y) = S(x)*C(y)-C(x)*S(y)"
      by (subst Sxminusy) simp
    also have "... ≤ S(x)-C(x)*S(y)"
    proof -
      have "S x ≤ S x" by simp
      moreover have "C y ≤ 1"
        using y_in_range1 and y_in_range2
        by (simp add: Cx_leq_1)
      ultimately  have "S x * C y ≤ S x"
        by (simp add: Sx_greq_0 mult_left_le
                    x_in_range1 x_in_range2)
      then show ?thesis by simp
    qed
    also have "... ≤ S(x)"
    proof -
      have "C(x) ≥ 0"
        by (simp add: x_in_range1 x_in_range2
                                      Cx_greq_0)
      moreover have "S(y) ≥ 0"
        by (simp add: y_in_range1 y_in_range2
                                      Sx_greq_0)
      ultimately show ?thesis by simp
    qed
    finally show ?thesis by simp
  qed


lemma Cseq_goes_to_1:
  assumes "p ≥  x " and " x  ≥ 0"
  and "C x ≥ 1 - c"
  shows "C (x/2) ≥ 1 - c/2"
  proof -
    have "C (x/2) ≥ (C (x/2))²"
    proof -
      have " C(x/2) ≤ 1" using assms(1) and assms(2)
        by (simp add: Cx_bounds)
      moreover have "0 ≤ C(x/2)"
        using assms(1) and assms(2)
        by (simp add: Cx_bounds)
      ultimately have "C (x/2)*C (x/2) ≤ C (x/2)"
```

```
        by (rule mult_left_le)
      then have "(C (x/2))² ≤ C (x/2)"
        by (subst power2_eq_square)
      then show ?thesis by simp
    qed
    moreover have "(C (x/2))² = (C x + 1)/2"
      by (subst Crecurrence) simp
    moreover have "... ≥ (1+(1-c))/2"
      using assms(3) by simp
    ultimately show ?thesis by simp
  qed


lemma Sseq_goes_to_0:
assumes "p ≥  x " and " x  ≥ 0"
    and "c ≤ (sqrt 39)/10" and "c ≥ 0"
    and "S x ≤ c"
  shows "S (x/2) ≤ (4/5)*c"
  proof -
    have "S x ≥ 0" using assms(1) assms(2)
      by (rule Sx_greq_0)
    from assms(5) and this have "(S x)² ≤ c²"
      by (rule power_mono)
    then have "4*(S (x / 2))² *(1-(S (x / 2))²) ≤ c²"
      using Srecurrence by simp
    then have 1:"(S (x/2))² *(1-(S (x/2))²) ≤ c²/4"
      by simp
    have 2:"(S(x/2))²*(1+(-(S(x/2))²))
                        =(S(x/2))²-(S(x/2))²*(S(x/2))²"
      by (subst distrib_left) simp
    from 1 2 have "(S(x/2))²-(S(x/2))⁴≤c²/4" by simp
    have 3:"p ≥ x/2" using assms(1) p_gr_0 by simp
    have 4:"x/2 ≥ 0" using assms(2) by simp
    have "S (x - x/2) ≤ S x"
      using assms(1) assms(2) 3 4
      by (rule S_nondec_lemma)
    then have 5:"S (x/2) ≤ c" using assms(5) by simp
    have "0 ≤ S (x/2)" using 3 4 by (rule Sx_greq_0)
    from 5 and this have "(S (x/2))⁴ ≤ c⁴"
      by (rule power_mono)
    from `(S (x/2))²-(S (x/2))⁴ ≤ c²/4` and this
      have "(S (x / 2))² ≤ c²/4 + c⁴" by simp
    also have "c²/4 + c⁴ ≤ c²/4 + (39/100)*c²"
    proof -
      from assms(3) assms(4)
        have "c²::nat ≤ ((sqrt 39)/(10::real))²"
        by (rule power_mono)
      also have "... = ((sqrt 39)^2)/(10^2)"
        by (simp add: power_divide)
      finally have "c^2 ≤ 39/100" by simp
      moreover have "c^2 ≥ 0" using assms(4) by simp
      ultimately have "c^2*c^2 ≤ (39/100)*c^2"
        by (rule mult_right_mono)
      then show ?thesis by simp
    qed
    also have "... = (1/4)*c^2 + (39/100)*c^2" by simp
    also have "... = (16/25)*c^2" by simp
    finally have 6:"(S (x/2))^2 ≤ (16/25)*c^2" by simp
    have "((4::real)/5)^2 = 16/25"
      by (simp add: field_class.power_divide)
    then have "(16/25)*c^2 = ((4/5)*c)^2"
      by (metis semiring_normalization_rules(30))
    from this and 6 have 7:"(S (x/2))^2 ≤ ((4/5)*c)^2"
      by simp
    have "(4/5)*c ≥ 0" using assms(4) by simp
    from 7 and this have "S (x / 2) ≤ (4/5)*c"
      by (rule power2_le_imp_le)
    then show "S (x / 2) ≤ (4/5)*c" by simp
  qed

lemma Cpseq: "C(p/(2^n)) ≥ 1 - 1/(2^n)"
  proof (induct n)
    case 0 show ?case by simp
  next
    case (Suc n)
    hence IH: "C(p/(2^n)) ≥ 1 - 1/(2^n)".
    have "p ≥ p/(2^n)" using  p_gr_0
      by (simp add: mult_imp_div_pos_le)
    moreover have "p/(2^n) ≥ 0" using p_gr_0
      by simp
    ultimately have "C((p/(2^n))/2) ≥ 1 - (1/(2^n))/2"
      using IH by (rule Cseq_goes_to_1)
    then show ?case by (simp add: power_commutes)
  qed


lemma Sprecurr:
 shows "S(p/(2^n))/(2*(1-1/2^{n+1})) ≥ S(p/2^{n+1})"
 and "S(p/(2^n)) ≥ 2*S(p/2^{n+1})*(1 - 1/(2^{n+1}))"
  proof -
    have "S(p/(2^n)) = S(p/2^{n+1}+p/2^{n+1})" by simp
    also have "...=
  S(p/2^{n+1})*C(p/2^{n+1})+C(p/2^{n+1})*S(p/2^{n+1})"
      by (rule Sxplusy[symmetric])
    finally have 1:
      "S(p/(2^n)) = 2*S(p/2^(n+1))*C(p/2^(n+1))"
      by simp
    have 2: "1 - 1/(2^(n+1)) ≤ C(p/(2^(n+1)))"
      by (rule Cpseq)
    have "0 ≤ S(p/(2^(n+1)))"
    proof (rule Sx_greq_0)
      have "(n+1)≥ 0" by simp
      then show "p/(2^(n+1)) ≥ 0" using p_gr_0
        by simp
      have "p ≥ p/(2^n)" using  p_gr_0
        by (simp add:
           linordered_field_class.mult_imp_div_pos_le)
      then show "p ≥ p/(2^(n+1))" using  p_gr_0
        by simp
    qed
    then have 3: "0 ≤ 2*S(p/(2^(n+1)))" by simp
    from 2 3 have "(2*S(p/2^(n+1)))*(1-1/(2^(n+1)))
                   ≤(2*S(p/2^(n+1)))*C(p/2^(n+1))"
      by (rule mult_left_mono)
    then show
      "S(p/(2^n))≥2*S(p/2^(n+1))*(1-1/(2^(n+1)))"
      using 1 by simp
    then have
      "S(p/(2^n))/2 ≥ S(p/2^(n+1))*(1-1/(2^(n+1)))"
      by simp
    moreover have "1 - 1/(2^(n+1))≥(0::real)"
    proof -
      have "0 ≤ 1/(2::real)" by simp
      also have "...≤1- 1/(2^(n+1))" by simp
      finally show ?thesis by simp
    qed
    ultimately have
"(S(p/(2^n))/2)/(1-1/(2^(n+1)))≥
       (S(p/2^(n+1))*(1-1/(2^(n+1))))/(1-1/(2^(n+1)))"
      by (rule divide_right_mono)
    moreover have "(1 - 1/(2^(n+1))) ≠ (0::real)"
    proof -
      have "0 < 1/(2::real)" by simp
      also have "...≤1- 1/(2^(n+1))" by simp
      finally have "(1 - 1/(2^(n+1))) > (0::real)"
        by simp
      then show ?thesis
        by (auto intro: order.strict_implies_not_eq)
    qed
    ultimately have
   "(S(p/(2^n))/2)/(1 - 1/(2^(n+1))) ≥ S(p/2^(n+1))"
```

```
      by (auto intro: mult_divide_mult_cancel_right)
    then show
      "S(p/(2^n))/(2*(1-1/2^(n+1))) ≥ S(p/2^(n+1))"
      by simp
  qed


lemma Spseq: shows "S(p/(2^(n+3))) ≤ (4/5)^(n+3)"
  proof (induct n)
    case 0
    have "S(p/2^2) ≤ 2/3"
    proof -
      have "S(p/2) ≤ 1"
        by (auto intro: Sx_leq_1 simp: real_le_lsqrt)
      moreover have
      "S(p/(2^1)) ≥ 2*S(p/2^(1+1))*(1 - 1/(2^(1+1)))"
        by (rule Sprecurr)
      ultimately have "1 ≥ 2*S(p/2^2)*(1 - 1/(2^2))"
        by simp
      then show ?thesis by simp
    qed
    have "S(p/2^3) ≤ 8/21"
    proof -
      have
      "S(p/(2^2)) ≥ 2*S(p/2^(2+1))*(1 - 1/(2^(2+1)))"
        by (rule Sprecurr)
      from this and `S(p/2^2) ≤ 2/3` have
        "2/3 ≥ 2*S(p/2^3)*(1 - 1/(2^3))" by simp
      then show ?thesis by simp
    qed
    have "8/(21::real) = 1000/2625" by simp
    also have "...≤ 1344/2625" by simp
    also have "... = 64/125" by simp
    also have "... = (4/5)^3"
      by (simp add: field_class.power_divide)
    finally have "8/(21::real)≤ (4/5)^3" by simp
    from this and `S(p/2^3) ≤ 8/21` show ?case
      by simp
  next
    case (Suc n)
    hence IH: "S(p/(2^(n+3))) ≤ (4/5)^(n+3)".
    have "p ≥ p/(2^(n+3))" using  p_gr_0
      by (simp add: mult_imp_div_pos_le)
    moreover have "p/(2^(n+3)) ≥ 0" using p_gr_0
      by simp
    moreover have "(4/5)^(n+3) ≤ (sqrt 39)/10"
    proof (induct n)
```

```
      case 0
      have "128*(128::real) =(16384::real)" by simp
      also have "... ≤ (24375::real) " by simp
      also have "... = 25*25*39" by simp
      finally have "(25*sqrt(39))^2 ≥ (128::real)^2"
        by simp
      then have ineq:"(25*sqrt(39)) ≥ 128"
        by (rule power2_le_imp_le) simp
      also have "(4/5)^3 = 64/ (125::real)"
        by algebra
      also have "... = 128/250" by simp
      also have "... ≤ (25*sqrt(39))/(250::real)"
        using ineq by (auto simp: divide_right_mono)
      also have "... = (sqrt 39)/10" by simp
      finally have "(4/5)^3 ≤ (sqrt 39)/10" by simp
      thus ?case by simp
    next
      case (Suc n)
      hence IH: "(4/5)^(n+3) ≤ (sqrt 39)/10".
      have "(4/5) ≤ (1::real)" by auto
      from IH have
        "(4/5)*(4/5)^(n+3) ≤ (4/5)*((sqrt 39)/10)"
        by simp
      also have "... ≤ (sqrt 39)/10"
        using `4/5 ≤ 1` by simp
      finally show ?case by (simp add: numeral_3_eq_3)
    qed
    moreover have "(4/5)^(n+3)≥(0::real)"
      by (subst zero_le_power) auto
    ultimately have
      "S((p/(2^(n+3)))/2)≤(4/5)*(4/5)^(n+3)"
      using IH by (rule Sseq_goes_to_0)
    then show ?case by (simp add: numeral_3_eq_3)
  qed

lemma pdiv2_le_p: assumes n_def: "n > 0"
                      shows "p/(2^(n::nat)) < p"
 proof -
    have "n > 0" using n_def by simp
    then have "(1::real) < 2^n" by simp
    have "0<p" by (rule p_gr_0)
    moreover have "p ≤ p" by simp
    moreover have "0 < (1::real)" by simp
    ultimately have "p/(2^n) < p/1"  using  `1 < 2^n`
      by (rule frac_less2)
    then show "p/(2^n) < p" by simp
  qed
```

```
(* no need for n>0*)
lemma Cpdiv2_leq_1: assumes n_def: "n > 0"
                     shows "C(p/(2^(n::nat))) ≤ 1"
  proof -
    have "n > 0" using n_def by simp
    then have "(1::real) < 2^n" by simp
    have "p/2^n > (0::real)" using `2^n > 1`
      by (simp add: p_gr_0)
    have "p/(2^n) < p" using n_def
      by (rule pdiv2_le_p)
    from `p/2^n > (0::real)` and this
      show "C(p/(2^n)) ≤ 1"
      by (simp add: Cx_leq_1)
  qed

lemma Spdiv2_greq_0: assumes n_def: "n > 0"
                      shows "S(p/(2^(n::nat))) ≥ 0"
  proof -
    have "n > 0" using n_def by simp
    then have "(1::real) < 2^n" by simp
    have "p/2^n > (0::real)" using `2^n > 1`
      by (simp add: p_gr_0)
    have "p/(2^n) < p" using n_def
      by (rule pdiv2_le_p)
    from `p/2^n > (0::real)` and this
      show "S(p/(2^n)) ≥ 0"
      by (simp add: Sx_greq_0)
  qed

(*author: Jacques D. Fleuriot*)

lemma Cpdiv2_leq_1': "C(p/(2^(n::nat))) ≤ 1"
proof (cases n)
  assume "n = 0" thus ?thesis by simp
next
  fix m
  assume "n = Suc m"
  then have "n > 0" by simp
  thus  ?thesis using Cpdiv2_leq_1 by blast
qed

lemma Spdiv2_greq_0':   "S(p/(2^(n::nat))) ≥ 0"
proof (cases n)
  assume "n = 0" thus ?thesis by simp
next
  fix m
  assume "n = Suc m"
  then have "n > 0" by simp
  thus  ?thesis using Spdiv2_greq_0 by blast
qed


lemma LIMSEQ_squeeze:
 assumes squeeze: "∀n. a n ≤ b n ∧ b n ≤ c n" and
         lima: "a ⟶ (L::real)" and
         limc: "c ⟶ L"
   shows limb: "b ⟶ L"
proof -
  {fix r assume r_gt_0: "(r::real) > 0"
   obtain m p where alr: "∀n≥m. ¦a n - L¦ < r"
     and clr: "∀n≥p. ¦c n - L¦ < r"
     using lima limc r_gt_0
     by (metis dist_real_def lim_sequentially)
   then have "∃no. ∀n≥no. ¦b n - L¦ < r"
   proof -
     obtain no where no: "no = max m p" by simp
     {fix n assume n_no: "n ≥ no"
     have squeeze_n: "a n ≤ b n ∧ b n ≤ c n"
       by (simp add: squeeze)
     then  have  "¦b n - L¦ < r"
       using n_no no alr clr max.boundedE
       by fastforce}
     then have "∀n≥no. ¦b n - L¦ < r" by blast
     then show ?thesis by blast
   qed}
   then show ?thesis by (simp add: LIMSEQ_iff)
qed

lemma LIMSEQ_inverse_2_pow_n:
               "(λn. 1 - 1/2ⁿ) ⟶ (1::real)"
proof -
   have seq_1: "(λn. 1) ⟶ (1::real)" by simp
   have "(λn. 1/(2ⁿ)) ⟶ (0::real)"
     by (simp add: LIMSEQ_divide_realpow_zero)
   then have
     "(λn. (1::real) - 1 / 2ⁿ) ⟶ 1 - 0"
     by (auto intro!: tendsto_diff simp only:)
   thus ?thesis by simp
qed

lemma limCpdiv2': "(λn. C(p/2ⁿ)) ⟶ 1"
proof -
   have a:
```

```
"∀n. 1 - 1/(2^n) ≤ C (p/2^n) ∧ C (p/2^n) ≤ 1"
        by (simp add: Cpdiv2_leq_1' Cpseq)
   have b: "(λn. 1 - 1/(2^n)) ⟶ (1::real)"
     by (rule LIMSEQ_inverse_2_pow_n)
   have  "(λn. 1) ⟶ (1::real)" by simp
   thus ?thesis using a b LIMSEQ_squeeze by simp
qed

(*end of Jacques Fleuriot's proofs*)


lemma limCminuspdiv2': "(λn. C(-p/2^n)) ⟶ 1"
proof -
  show ?thesis using limCpdiv2'
    by (simp add: Cminusx)
qed


lemma limSpdiv2': "(λn. S(p/2^(n+3))) ⟶ 0"
proof -
   have a:
     "∀n. S(p/2^(n+3))≥0 ∧ (4/5)^(n+3)≥S(p/2^(n+3))"
         by (simp add: Spdiv2_greq_0' Spseq)
   have b0: "(λn. (4/5)^n) ⟶ (0::real)"
     by (auto simp: LIMSEQ_rabs_realpow_zero2)
   from b0 have b1:
     "(λn. (((4/5)^n))*((4/5)^3)) ⟶ (0::real)"
     by (auto simp:tendsto_mult_left_zero)
   have
"(λn.(4/(5::real))^(n+3))=(λn.(((4/5)^n))*((4/5)^3))"
     by (auto simp: power_add)
   from this b1 have b:
   "(λn. (4/(5::real))^(n+3)) ⟶ (0::real)"
   by simp
   have  "(λn. 0) ⟶ (0::real)" by simp
   thus ?thesis using a b LIMSEQ_squeeze by simp
qed


lemma C_continuous_0: "isCont C 0"
  proof (subst isCont_def,subst LIM_def)
    show
"∀r.0<r ⟶
(∃s>0. ∀x. x≠0 ∧ dist x 0<s ⟶ dist (C x) (C 0)<r)"
    proof (rule allI, rule impI)
      fix r
      assume r_def: " (0::real) < r"
      have
"∃s>0. ∀x. x≠0∧dist x 0<s ⟶ dist (C x) (C 0)<r"
      proof -
        have
          "∀r>0. ∃no. ∀n≥no. dist (C (p/2^n)) 1<r"
          using limCpdiv2'
          by (simp add: lim_sequentially)
        then have
          "∃no. ∀n≥no. dist (C (p/2^n)) 1 < r"
          using r_def by auto
        then obtain no where no_def:
          "∀n≥no. dist (C (p/2^n)) 1 < r"
          by auto
        then obtain n where n_def: "n ≥ no" by auto
        then have "¦ 1 - C (p/2^n)¦ < r"
          using no_def by (auto simp: dist_real_def)
        have 1: "p/(2^n) > 0" by (simp add: p_gr_0)
        have 2:
"∀x. x≠0∧dist x (0::real)<p/(2^n)
                              ⟶ dist (C x) (C 0)<r"
        proof (rule allI,rule impI)
          fix x
          assume a0: "x≠0 ∧ dist x (0::real)<p/(2^n)"
          from this have "x ≠ 0" by simp
          from a0 have "¦x¦ < (p/(2^n))"
            by (auto simp: dist_real_def)
          consider (a) "x>0" | (b) "x=0" | (c) "x<0"
            by linarith
          then have  "¦ 1 - C x¦ < r"
          proof cases
            case a
            from `¦x¦ < (p/(2^n))`
              have "x < (p/(2^n))" by auto
            have "x < p"  using `x < (p/(2^n))`
            proof (cases n)
              assume "n = 0" thus ?thesis
                using `x < (p/(2^n))` by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < n" by simp
              then have "(p/(2^n)) < p"
                by (rule pdiv2_le_p)
              from this and `x < (p/(2^n))`
                show ?thesis  by simp
            qed
            from this and a have Cx_up: "C x ≤ 1"
```

```
    by (simp add: Cx_leq_1)
  from a have 1:"0 ≤ x" by simp
  have 2:"x + ((p/(2^n)) -x) ≤ p"
  proof (cases n)
    assume "n = 0" thus ?thesis by auto
  next
    fix m
    assume "n = Suc m"
    then have "0 < n" by simp
    then have "(p/(2^n)) < p"
      by (rule pdiv2_le_p)
    from this show ?thesis by auto
  qed
  from `x < (p/(2^n))`
    have 3:"0 ≤ ((p/(2^n)) -x)" by auto
  from 1 2 3
    have "C (x + ((p/(2^n)) - x)) ≤ C x"
    by (rule C_nonincreasing_on_0_p)
  then have Cx_low: "C (p/(2^n)) ≤ C x"
    by simp
  from Cx_up and Cx_low show "¦ 1 - C x¦ < r"
  using `¦1 - C (p / 2 ^ n)¦ < r` by auto
next
  case b
  show "¦ 1 - C x¦ < r" using `x=0`
    and `x ≠ 0` by auto
next
  case c
  obtain x' where x'_def: " x' = -x"
    by auto
  have c': "x'>0" using x'_def c by auto
  from `¦x¦ < (p/(2^n))`
    have "x' < (p/(2^n))"
    using x'_def by auto
  have "x' < p"  using `x' < (p/(2^n))`
  proof (cases n)
    assume "n = 0" thus ?thesis
      using `x' < (p/(2^n))` by auto
  next
    fix m
    assume "n = Suc m"
    then have "0 < n" by simp
    then have "(p/(2^n)) < p"
      by (rule pdiv2_le_p)
    from this and `x' < (p/(2^n))`
      show ?thesis  by simp
  qed
```

```
            from this and c' have Cx_up: "C x' ≤ 1"
              by (simp add: Cx_leq_1)
            from c' have 1:"0 ≤ x'" by simp
            have 2:"x' + ((p/(2ⁿ)) -x') ≤ p"
            proof (cases n)
              assume "n = 0" thus ?thesis by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < n" by simp
              then have "(p/(2ⁿ)) < p"
                by (rule pdiv2_le_p)
              from this show ?thesis by auto
            qed
            from `x' < (p/(2ⁿ))`
              have 3:"0 ≤ ((p/(2ⁿ)) -x')" by auto
            from 1 2 3 have "C(x'+((p/(2ⁿ))-x'))≤C x'"
              by (rule C_nonincreasing_on_0_p)
            then have Cx_low: "C (p/(2ⁿ)) ≤ C x'"
              by simp
            from Cx_up and Cx_low have "¦ 1 - C x'¦<r"
              using `¦1 - C (p / 2 ^ n)¦ < r` by auto
            then show "¦ 1 - C x¦ < r" using Cminusx
              by (simp add: x'_def)
          qed
          then show "dist (C x) (C 0) < r"
            using dist_real_def by simp
        qed
        from 1 and 2 show ?thesis by auto
      qed
      then show
"∃s>0. ∀x. x≠0 ∧ dist x 0<s ⟶ dist (C x) (C 0)<r"
        by auto
    qed
  qed


lemma S_continuous_0: "isCont S 0"
  proof (subst isCont_def,subst LIM_def)
    show
"∀r.0<r ⟶ (∃s>0. ∀x. x≠0 ∧ dist x 0<s
                                  ⟶ dist (S x) (S 0)<r)"
    proof (rule allI, rule impI)
      fix r
      assume r_def: "(0::real) < r"
      have
"∃s>0. ∀x. x≠0∧dist x 0<s ⟶ dist (S x) (S 0)<r"
```

```
      proof -
        have
  "∀r>0. ∃no. ∀n≥no. dist (S (p/2^(n+3))) 0 < r"
          using limSpdiv2'
          by (simp add: lim_sequentially)
        then have
          "∃no. ∀n≥no. dist (S (p/2^(n+3))) 0<r"
          using r_def by auto
        then obtain no where no_def:
          "∀n≥no. dist (S (p/2^(n+3))) 0 < r"
          by auto
        then obtain n where n_def: "n ≥ no" by auto
        then have "¦ 0 - S (p/2^(n+3))¦ < r"
          using no_def by (auto simp: dist_real_def)
        have 1: "p/(2^(n+3)) > 0"
          by (simp add: p_gr_0)
        have 2:
"∀x. x≠0∧dist x (0::real)<p/(2^(n+3))
                               ⟶ dist (S x) (S 0)<r"
        proof (rule allI,rule impI)
          fix x
          assume a0:
           "x ≠ 0 ∧ dist x (0::real) < p/(2^(n+3))"
          from this have "x ≠ 0" by simp
          from a0 have "¦x¦ < (p/(2^(n+3)))"
            by (auto simp: dist_real_def)
          consider (a) "x>0" | (b) "x=0" | (c) "x<0"
            by linarith
          then have  "¦ S x¦ < r"
          proof cases
            case a
            from `¦x¦ < (p/(2^(n+3)))`
              have "x < (p/(2^(n+3)))" by auto
            have "x < p"  using `x < (p/(2^(n+3)))`
            proof (cases n)
              assume "n = 0"
              have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              thus ?thesis using `x < (p/(2^(n+3)))`
                by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              from this and `x < (p/(2^(n+3)))`
                show ?thesis  by simp
            qed
            from this and a have Sx_low: "S x ≥ 0"
              by (simp add: Sx_greq_0)
            from a have 1:"0 ≤ x" by simp
            have 2:"x + ((p/(2^(n+3))) -x) ≤ p"
            proof (cases n)
              assume "n = 0"
              have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              then show ?thesis by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              from this show ?thesis by auto
            qed
            from `x < (p/(2^(n+3)))`
              have 3:"0 ≤ ((p/(2^(n+3))) -x)"
              by auto
            from 1 2 3 have
              "S x ≤ S (x + ((p/(2^(n+3))) - x))"
              by (rule S_nondecreasing_on_0_p)
            then have Sx_up: "S (p/(2^(n+3))) ≥ S x"
              by simp
            from Sx_up and Sx_low show "¦ S x¦ < r"
              using `¦0 - S (p/2^(n+3))¦ < r`
              by auto
          next
            case b
            show "¦S x¦ < r" using `x=0` and `x ≠ 0`
              by auto
          next
            case c
            obtain x' where x'_def: " x' = -x"
              by auto
            have c': "x'>0" using x'_def c by auto
            from `¦x¦ < (p/(2^(n+3)))`
              have "x' < (p/(2^(n+3)))" using x'_def
              by auto
            have "x' < p"  using `x' < (p/(2^(n+3)))`
            proof (cases n)
              assume "n = 0"
              have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              thus ?thesis using `x' < (p/(2^(n+3)))`
                by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              from this and `x' < (p/(2^(n+3)))`
                show ?thesis  by simp
            qed
            from this and c' have Sx_low: "S x' ≥ 0"
              by (simp add: Sx_greq_0)
            from c' have 1:"0 ≤ x'" by simp
            have 2:"x' + ((p/(2^(n+3))) -x') ≤ p"
            proof (cases n)
              assume "n = 0"
              have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p" by (rule
        pdiv2_le_p)
              then show ?thesis by auto
            next
              fix m
              assume "n = Suc m"
              then have "0 < (n+3)" by simp
              then have "(p/(2^(n+3))) < p"
                by (rule pdiv2_le_p)
              from this show ?thesis by auto
            qed
            from `x' < (p/(2^(n+3)))`
              have 3:"0 ≤ ((p/(2^(n+3))) -x')"
              by auto
            from 1 2 3 have
              "S(x'+((p/(2^(n+3)))-x'))≥S x'"
              by (rule S_nondecreasing_on_0_p)
            then have Sx_up: "S(p/(2^(n+3)))≥S x'"
              by simp
            from Sx_up and Sx_low have "¦S x'¦<r"
              using `¦ 0 - S (p/2^(n+3))¦ < r`
              by auto
            then show "¦ S x¦ < r" using Sminusx
              by (simp add: x'_def)
          qed
          then show "dist (S x) (S 0) < r"
```

```
              using dist_real_def by simp
          qed
          from 1 and 2 show ?thesis by auto
        qed
        then show
     "∃s>0. ∀x. x≠0∧dist x 0<s ⟶ dist (S x) (S 0)<r"
          by auto
      qed
    qed

(* really a special case of LIM_offset*)
lemma limit_subst:
  "((S::(real ⇒ real)) ─(a::real)→ (S a))
                         = ((λx. S(x+a)) ─0→ (S a))"
    proof
      assume asm: "S ─a→ S a"
      show "(λx. S (x + a)) ─0→ S a"
      proof (simp add: LIM_def)
        have
"∀r>0. ∃s>0. ∀x. x ≠ a ∧ dist x a < s
                              ⟶ dist (S x) (S a) < r"
          using asm by (simp add: LIM_def)
        then have
"∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ dist (x+a) a < s
                       ⟶ dist (S (x + a)) (S a) < r)"
          by force
        then have
"∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ ¦(x+a)-a¦ < s
                       ⟶ dist (S (x + a)) (S a) < r)"
          using dist_real_def by simp
        then show
  "∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ dist x 0 < s
                       ⟶ dist (S (x + a)) (S a) < r)"
          using dist_real_def by simp
      qed
    next
      assume asm: "(λx. S(x+a)) ─0→ (S a)"
      show "S ─a→ S a"
      proof (simp add: LIM_def)
      have
"∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ dist x 0 < s
                       ⟶ dist (S (x + a)) (S a) < r)"
        using asm by (simp add: LIM_def)
      then have
"∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ ¦(x+a)-a¦ < s
                       ⟶ dist (S (x + a)) (S a) < r)"
        using dist_real_def by simp
      then have
"∀r>0. ∃s>0. ∀x. (x ≠ 0 ∧ dist (x+a) a < s
                       ⟶ dist (S (x + a)) (S a) < r)"
        using dist_real_def by simp
      then have
"∀r>0. ∃s>0. ∀x. ((x-a) ≠ 0 ∧ dist ((x-a)+a) a <s
                   ⟶ dist (S ((x-a)+a)) (S a) < r)"
        by blast
      then show
"∀r>0. ∃s>0. ∀x. x ≠ a ∧ dist x a < s
                              ⟶ dist (S x) (S a) < r"
        by force
      qed
    qed

lemma S_cont: "isCont S a"
    proof -
      from C_continuous_0 have "C ─0→ (C 0)"
        unfolding isCont_def
        by simp
      then have 1:"C ─0→ 1"
        by (subst C0_eq_1[symmetric]) simp
      have 2:"(λ(x::real). S a) ─(0::real)→ S a"
        by simp
      from S_continuous_0 have "S ─0→ (S 0)"
        unfolding isCont_def
        by simp
      then have 3:"S ─0→ 0"
        by (subst S0_eq_0[symmetric]) simp
      have 4:"(λx. C a) ─(0::real)→ C a"
        by simp
      from 1 2 have "(λ(x::real). (C x)*(S a)) ─0→ S a"
        using  tendsto_mult
        by fastforce
      moreover from 3 4 have 6:"(λx. (S x)*(C a))─0→ 0"
        using  tendsto_mult
        by fastforce
      ultimately have
        "(λx. (S x)*(C a)+(C x)*(S a)) ─0→ 0+S a"
        using tendsto_add by fastforce
      moreover have
        "(λx. (S x)*(C a)+(C x)*(S a))=(λx. S(x+a))"
        using Sxplusy by simp
      ultimately have "(λx. S(x+a)) ─0→ S a"
        by simp
      then have "S ─a→ S a"
        by (simp add: limit_subst)
      then show ?thesis unfolding isCont_def
        by simp
    qed


  lemma C_cont: "isCont C a"
    proof (simp add: isCont_def)
    from S_cont have "S ─(p+a)→ S(p+a)"
      unfolding isCont_def by simp
    then have "(λx. S(x+p)) ─(p+a)-p→ S(p+a)"
      by (rule LIM_offset)
    moreover have "(λx. S(x+p)) = (λx. S(p+x))"
    proof
      fix x
      have 1:"(x + p) = (p + x)" by simp
      then show "S (x + p) = S (p + x)"
        by (subst 1) simp
    qed
    ultimately have "(λx. S(p+x)) ─a→ S(p+a)" by auto
    then show "C ─a→ C a" using Spplusx_eq_Cx by simp
  qed

  definition F :: "real ⇒ real" where
  " F ≡ (λx. S x / x)"

  lemma F_cont: "⟦0< a; a ≤ 1⟧ ⟹ isCont F a"
    proof -
      assume a_gr_0: "0< a"
      assume "a ≤ 1"
      have 1:"isCont (λx. x) a" by simp
      have 2:"(λx. x) a ≠ 0" using a_gr_0 by simp
      have "isCont (λx. S x / x) a" using S_cont 1 2
        by (auto intro: isCont_divide)
      then show "isCont F a" using F_def by simp
    qed

  lemma C_strict_less_one_seq:
   assumes x_greater_0:"0<x" and x_less_p:"x≤p"
    and Cx_less_1:"C x < 1"
    shows "C (x/2) < 1"
  proof -
    have "2 * (C (x/2))² - 1 < 1" using C2x(2)
      Cx_less_1 by simp
    then have "(C (x/2))² < 1" by simp
    then show "C (x/2) < 1"
      by (subst power2_less_imp_less) auto
  qed
```

```
lemma S_strict_greater_zero_seq:
  assumes x_greater_0:"0<x" and x_less_p:"x≤p"
  and Sx_greater_0:"S x > 0"
  shows "S (x/2) > 0"
proof -
  have "4∗(S (x / 2))² ∗ (1 - (S (x / 2))²) > 0"
    using Srecurrence Sx_greater_0 by simp
  then have "(S (x / 2))² ∗ (1 - (S (x / 2))²) > 0"
    by simp
  then have "(S (x / 2))² > 0" by auto
  then have "S (x / 2) ≠ 0" by simp
  from x_greater_0 have "0≤x/2" by simp
  from x_less_p have "x/2 ≤ p" using p_gr_0 by simp
  from `x/2 ≤ p` `0≤x/2` have "S (x / 2) ≥ 0"
    by (rule Sx_greq_0)
  from this `S (x / 2) ≠ 0` show "S (x / 2) > 0"
    by linarith
qed

lemma C_strict_less_one_inducted:
  assumes x_greater_0:"0<x" and x_less_p:"x≤p"
  and Cx_less_1:"C x < 1"
  shows "C (x/(2ⁿ)) < 1"
proof (induct n)
  case 0
  show ?case using Cx_less_1 by auto
next
  case (Suc n)
  then have IH: "C (x/2ⁿ) < 1".
  moreover have "0<x/2ⁿ" using x_greater_0
    by simp
  moreover have "x/2ⁿ ≤ p" using x_less_p
  proof -
    have "2ⁿ ≥ (1::real)"
      by (rule two_realpow_ge_one)
    then have "x/2ⁿ ≤ p/2ⁿ"
      using x_less_p by (simp only: divide_right_mono)
    moreover have "p ≥ p/(2ⁿ)"
      using p_gr_0 by (simp add: mult_imp_div_pos_le)
    ultimately show ?thesis by auto
  qed
  ultimately have "C ((x/2ⁿ)/2) < 1"
    by (subst C_strict_less_one_seq) auto
  then show ?case by (simp add: power_commutes)
qed
```

```
lemma S_strict_greater_zero_inducted:
   assumes x_greater_0:"0<x" and x_less_p:"x≤p"
   and Sx_greater_0:"S x > 0"
   shows "S (x/(2^n)) > 0"
proof (induct n)
  case 0
  show ?case using Sx_greater_0 by auto
next
  case (Suc n)
  then have IH: "S (x/(2^n)) > 0".
  moreover have "0<x/2^n" using x_greater_0
    by simp
  moreover have "x/2^n ≤ p" using x_less_p
  proof -
    have "2^n ≥ (1::real)"
      by (rule two_realpow_ge_one)
    then have "x/2^n ≤ p/2^n"
      using x_less_p by (simp only: divide_right_mono)
    moreover have "p ≥ p/(2^n)" using  p_gr_0
      by (simp add: mult_imp_div_pos_le)
    ultimately show ?thesis by auto
  qed
  ultimately have "S ((x/(2^n))/2) > 0"
    by (subst S_strict_greater_zero_seq) auto
  then show ?case by (simp add: power_commutes)
qed

lemma exists_natpow_greaterthan:
    assumes "1 < (a::real)" and "0 < (b::real)"
    shows "∃(k::nat). a^k > b"
  proof-
    obtain c where c_def:"c = a-1"
      and c_gr0:"c > 0" using assms(1) by auto
    have a_powr_m_grows:
      "∀m. a^(m::nat) > 1 + (m-1)*c"
    proof
      fix m
      show "a^(m::nat) > 1 + (m-(1::real))*c"
      proof (induct m)
      case 0
        have "(1::real) > 1 - c" using c_gr0 by simp
        then show ?case using assms(1) by auto
      case (Suc m)
        then have IH: "a^m > 1 + (m-(1::real))*c"
          by simp
        consider (a) "m = 0"| (b) "m = 1" | (c) "m>1"
          by linarith
```

```
 then show ?case
 proof cases
   case a
   then show "1+(real (Suc m)-1)*c<a^(Suc m)"
     using `m = 0` assms(1) by simp
 next
   case b
   have "1 + (2-1)*c = 1 + c" using c_def
     by simp
   also have "... < (1::real) + c + c + c*c"
   proof -
     have "0<c*c" using c_gr0 c_gr0
       by (rule mult_pos_pos)
     from c_gr0 and this have "0< c + c*c"
       by (rule add_pos_pos)
     then show ?thesis using c_gr0 by auto
   qed
   also have "... = (1+c)*(1+c)" by algebra
   also have "... = (1+c)*(1+c)^1"
     using c_gr0 by auto
   also have "... = (1+c)^1+1"
     using c_gr0 by  auto
   also have "... = a^2" using c_def
     by (auto simp:
            semiring_normalization_rules(29))
   finally have "1 + (2-1)*c < a^2" by simp
   moreover have "real (Suc 1) = 2" by simp
   moreover have "Suc 1 = 2" by simp
   ultimately show
     "1+(real (Suc m)-1)*c<a^(Suc m)"
     using  assms(1)
     by (subst `m = 1`, subst `m = 1`)
     (auto simp:
             semiring_normalization_rules(29))
 next
   case c
   then have "(m-(1::real))*c*c > 0"
     using c_gr0  by simp
   then have
     "(1::real)+m*c<1+m*c+(m-(1::real))*c*c"
     by simp
   also have
"... =((1::real)+m*c-c)*1+((1::real)+m*c-c)*c"
     by algebra
   also have "... = ((1::real) + m*c-c)*(1+c)"
     by (rule distrib_left[symmetric])
   also have
```

```
         "(1+m∗c-c)∗(1+c)=(1+(m-(1::real))∗c)∗(1+c)"
           by algebra
         also have
    "(1+(m-(1::real))∗c)∗(1+c)=(1+(m-(1::real))∗c)∗a"
           using c_def by simp
         also have "(1+(m-(1::real))∗c)∗a<a^m∗a"
           using IH assms(1)
           by (auto simp: mult_strict_right_mono)
         also have "... = a^(m+1)" using assms(1)
           by auto
         finally show "1+(real (Suc m)-1)∗c<a^(Suc m)"
           by auto
       qed
     qed
   qed
   obtain m where m_def: "(m::nat) >(b-1)/c + 1"
     using c_gr0 reals_Archimedean2 by auto
   from m_def have "(m-(1::real)) >(b-1)/c" by simp
   then have "c∗(m-(1::real)) > c∗((b-1)/c)"
     using c_gr0
     by (subst mult_less_cancel_left_pos)
   have "c∗((b-1)/c) = c∗(b-1)/c"
     by (rule times_divide_eq_right)
   also have "... = c/c∗(b-1)"
     by (rule times_divide_eq_left[symmetric])
   finally have "c∗((b-1)/c) = b-1" using c_gr0
     by auto
   from `c∗(m-(1::real)) > c∗((b-1)/c)`
     and this have "(m-(1::real))∗c > b-1"
     by (simp add: mult.commute)
   from a_powr_m_grows have
     "a^(m::nat) >1+(m-(1::real))∗c" by auto
   from this and `(m-(1::real))∗c > b-1`
     have "a ^ (m::nat) > b" by auto
   then show ?thesis by auto
 qed


lemma exists_natpow_lessthan2:
  assumes "1 < (a::real)" and "0 < (b::real)"
  shows "∃(k::nat). 1/(a ^ k) < b"
 proof-
   from assms(2) have "0<1/b" by simp
   from assms(1) and this have "∃(x::nat). 1/b<a^x"
     by (rule exists_natpow_greaterthan)
   then obtain x::nat where x_def: "1/b <a^x"
     by auto
    from x_def and assms(2) have "(1/b)*b < (a^x)*b"
      by (rule mult_strict_right_mono)
    then have "1 < (a ^ x)*b" using assms(2) by simp
    moreover have "1/(a ^ x) > 0" using assms(1)
      by auto
    ultimately have "(1/(a^x))*1<(1/(a^x))*((a^x)*b)"
       by (subst mult_less_cancel_left_pos)
    then have "1/(a ^ x) < b" using assms(1) by simp
    then show ?thesis by auto
  qed

lemma exists_natpowgr0_lessthan2:
     assumes "1 < (a::real)" and "0 < (b::real)"
     shows "∃(k::nat)>0. 1/(a^k) < b"
proof-
  from assms obtain k::nat where " 1/(a^k) < b"
    using exists_natpow_lessthan2 by auto
  then have "(1/(a^k))/a < b/1" using assms(1)
    by (subst frac_less2) auto
  then have "1/((a^k)*a) < b" by simp
  then have "1/(a^(k+1)) < b"
    by (subst
    semiring_normalization_rules(26)[symmetric]) simp
  then show ?thesis by auto
qed

lemma twopowers_sandwich: assumes "1≤(a::real)"
        shows "∃(n::nat). ( 2^n ≤ a) ∧ (a < 2^(n+1))"
proof -
  have "∃n. 2^n≤ a"
  proof
    have "2^0=(1::real)" by simp
    then show "2^0 ≤ a" using assms(1) by simp
  qed
  have
     "∃M.(∀m. 2^(m::nat)≤a ⟶ M≥m)∧(2^(M::nat)≤a)"
  proof (rule exI, safe)
    def A == "{m. 2^(m::nat)≤a}"
    have "∃m. 2^(m::nat) > a" using `a≥1`
      by (subst exists_natpow_greaterthan) auto
    then obtain m where m_def:"2^(m::nat) > a"
      by auto
    have "∀n\<in>A. n < m"
    proof safe
      fix n
      assume "n \<in> A"
      then have "2^n≤(a::real)" using A_def by simp
      also from m_def have "...<  2^m" by simp
      finally show "n < m" by simp
    qed
    then have A_finite:"finite A"
      using bounded_nat_set_is_finite A_def by blast
    moreover have "A ≠ {}" using `∃n. 2^n≤ a` A_def
      by blast
    ultimately have "Max A \<in> A" by (rule Max_in)
    then show "2^(Max A)≤a" using A_def by blast
    from A_finite show "⋀m. 2^m ≤ a ⟹ m ≤ Max A"
      using Max_ge A_def by blast
  qed
  then obtain M where M_def:
    "⋀m.2^(m::nat)≤a ⟹ M≥m"
    and M_in_set:"2^(M::nat)≤ a" by auto
  have "2^(M+1) > a"
  proof (rule ccontr)
    assume "  a < 2^(M+1)"
    then have "a ≥ 2^(M+1) " by simp
    from M_def this have "M ≥ M+1" by simp
    then show False by simp
  qed
  from this and M_in_set show ?thesis by auto
qed


lemma C_strict_less_one: assumes x_gr_0:"0<x"
                              and "x≤p"
                             shows "C x<1"
proof -
  consider (a) "x<p/2" | (b) "p/2≤x"
    by linarith
  then show ?thesis
  proof cases
    case a
    assume "x<p/2"
    then have "p/x ≥1" using x_gr_0 by auto
    then obtain n where "2^(n+1) > p/x"
      and "p/x≥2^n" using twopowers_sandwich
      by blast
    from `p/x≥2^n` have "p≥2^n∗x" using x_gr_0
      by (simp only: pos_le_divide_eq)
    moreover from `2^(n+1) > p/x` have "2^n∗x≥p/2"
      using x_gr_0 by (auto simp: field_simps)
    ultimately have "C(p/2 + (2^n∗x-p/2)) ≤C (p/2)"
      by (subst C_nonincreasing_on_0_p) auto
    then have "C(2^n∗x) ≤C (p/2)" by simp
    also have "... = sqrt 2 / 2"
      by (rule Spdiv2_Cpdiv2(2))
    finally have "C(2ⁿ*x) < 1" using sqrt2_less_2
      by linarith
    from `2ⁿ*x≥p/2` p_gr_0 `p≥2ⁿ*x` this
      have "C((2ⁿ*x)/ 2 ^ n) < 1"
      by (subst C_strict_less_one_inducted) auto
    then show "C(x) < 1" by auto
  next
    case b
    assume "p/2≤x"
    from p_gr_0 assms(2) this
      have "C(p/2 + (x-p/2)) ≤C (p/2)"
      by (subst C_nonincreasing_on_0_p) auto
    then have "C(x) ≤C (p/2)" by simp
    also have "... = sqrt 2 / 2"
      by (rule Spdiv2_Cpdiv2(2))
    finally show "C(x) < 1" using sqrt2_less_2
      by linarith
  qed
qed

lemma S_strict_greater_zero:
    assumes x_gr_0:"0<x" and "x≤p"
      shows  "S x > 0"
proof -
  consider (a) "x<p/2" | (b) "p/2≤x"
    by linarith
  then show ?thesis
  proof cases
    case a
    assume "x<p/2"
    then have "p/x ≥1" using x_gr_0 by auto
    then obtain n where "2ⁿ⁺¹ > p/x" and "p/x≥2ⁿ"
      using twopowers_sandwich
      by blast
    from `p/x≥2ⁿ` have "p≥2ⁿ*x" using x_gr_0
      by (simp only: pos_le_divide_eq)
    moreover from `2ⁿ⁺¹ > p/x` have "2ⁿ*x≥p/2"
      using x_gr_0 by (auto simp: field_simps)
    ultimately have "S(p/2 + (2ⁿ*x-p/2)) ≥S (p/2)"
      by (subst S_nondecreasing_on_0_p) auto
    have "0 < sqrt 2 / 2" by simp
    also have "sqrt 2 / 2 = S (p/2)"
      by (rule Spdiv2_Cpdiv2(1)[symmetric])
    also have "... ≤ S(2ⁿ*x)"
      using `S(p/2 + (2ⁿ*x-p/2)) ≥S (p/2)` by simp
    finally have "S(2^n*x) > 0" by linarith
    from `2^n*x≥p/2` p_gr_0 `p≥2^n*x` this
      have "S((2^n*x)/ 2 ^ n) > 0"
      by (subst S_strict_greater_zero_inducted) auto
    then show "S(x) > 0" by auto
  next
    case b
    assume "p/2≤x"
    from p_gr_0 assms(2) this
      have "S(p/2 + (x-p/2)) ≥ S (p/2)"
      by (subst S_nondecreasing_on_0_p) auto
    have "0 < sqrt 2 / 2" by simp
    also have "... = S (p/2)"
      by (rule Spdiv2_Cpdiv2(1)[symmetric])
    also have "... ≤ S (x)"
      using `S(p/2 + (x-p/2)) ≥ S (p/2)` by simp
    finally show "S(x) > 0" by linarith
  qed
qed

lemma F_seq_dec_lemma:
assumes x_gr_0: "0<x" and nx_leq_p:"n*x≤p"
and "(n::real)>1" and F_cond: "F (n*x) < F ((n-1)*x)"
shows "F ((n+1)*x) < F (n*x)"
proof -
  from F_cond have
    "(S (n*x))/(n*x) < (S ((n-1)*x))/((n-1)*x)"
    using F_def by auto
  then have
 "((S (n*x))/(n*x))*x < ((S ((n-1)*x))/((n-1)*x))*x"
    using x_gr_0
    by (simp only: mult_strict_right_mono)
  then have "(S (n*x))/n < (S ((n-1)*x))/(n-1)"
    using x_gr_0 by auto
  then have Step1: "(n-1)*(S (n*x))<n*(S ((n-1)*x))"
    using `n>1` by (auto simp: field_simps)
  have "(n-1)*x = (n*x)+(-x)" by algebra
  then have "S ((n-1)*x) = S ((n*x)+(-x))" by simp
  also have "... = S (n*x)*C (-x) + C (n*x)*S (-x)"
    by (subst Sxplusy) auto
  also have "... = S(n*x) * C x - C (n*x) * S x"
    using Sminusx Cminusx by simp
  finally have Step2:
    "S((n-1)*x)=S(n*x)*C x-C(n*x)*S x"
    by simp
  have "(n-1)*(S(n*x))<n*(S(n*x)*C x-C(n*x)*S x)"
    using Step1 by (subst Step2[symmetric])
  moreover have "(n-1)∗(S(n∗x))=n∗(S(n∗x))-S(n∗x)"
    by algebra
  ultimately have Step3:
    "n∗(S(n∗x))-S(n∗x)<n∗(S(n∗x)∗C x-C(n∗x)∗S x)"
    by simp
  from nx_leq_p `n>1` have "x ≤p/n"
    by (auto simp: field_simps)
  moreover from `n>1` p_gr_0 have "p ≥ p/n"
    by (auto simp: mult_imp_div_pos_le)
  ultimately have "x ≤ p" by simp
  from x_gr_0 this have C_leq_1:"C x≤1"
    by (simp only: Cx_leq_1)
  moreover from x_gr_0 nx_leq_p `n>1`
    have S_gr_0:"S (n∗x)>0"
    by (auto simp: S_strict_greater_zero)
  ultimately have "S(n∗x)∗C(x) ≤ S(n∗x)∗1"
    by (subst mult_le_cancel_left_pos)
  from this `n>1` have
    "n∗S(n∗x)∗C(x)-S(n∗x)≤n∗S(n∗x)-S(n∗x)" by simp
  also have "...< n∗(S(n∗x) ∗ C x - C (n∗x) ∗ S x)"
    by (subst Step3) simp
  also have "... ≤ n∗(S(n∗x) - C (n∗x) ∗ S x)"
    using C_leq_1 `n>1` S_gr_0
    by (auto simp: field_simps)
  also have "... = n∗S(n∗x) - n∗C (n∗x) ∗ S x"
    by algebra
  finally have
    "n∗S(n∗x)∗C(x)+n∗C(n∗x)∗S x<n∗S(n∗x)+S(n∗x)"
    by linarith
  moreover have "n∗S(n∗x) + S(n∗x) = (n+1)∗S(n∗x)"
    by algebra
  moreover have
"n∗S(n∗x)∗C x+n∗C(n∗x)∗S x=n∗(S(n∗x)∗C x+C(n∗x)∗S x)"
    by algebra
  ultimately have
    "n∗(S(n∗x)∗C(x)+C(n∗x)∗S x)<(n+1)∗S(n∗x)"
    by simp
  then have "n∗S(n∗x+x) < (n+1)∗S(n∗x)"
    by (subst Sxplusy[symmetric])
  then have "S(n∗x+x)/(n+1) < S(n∗x)/n" using `n>1`
    by (auto simp: field_simps)
  then have "(S(n∗x+x)/(n+1))/x < (S(n∗x)/n)/x"
    using x_gr_0
    by (subst divide_strict_right_mono) auto
  then have "S(n∗x+x)/((n+1)∗x) < S(n∗x)/(n∗x)"
    by auto
  moreover have "n∗x+x = (n+1)∗x" by algebra
  ultimately have
    "S((n+1)∗x)/((n+1)∗x)<S(n∗x)/(n∗x)" by simp
  then show "F ((n+1)∗x) < F (n∗x)" using F_def
    by simp
qed

lemma nat_0_1_2_2imp3_induct
   [case_names 0 1 2 3 step]:
  assumes "P 0" "P 1" "P 2" "P 2 ⟹ P 3"
        "⋀n. n ≥ 3 ⟹ P n ⟹ P (Suc n)"
  shows   "P n"
proof (induction n rule: less_induct)
  case (less n)
  show ?case
    using assms(5)[OF _ less.IH[of "n - 1"]]
    by (cases "n ≤ 3") (insert assms(1-4),
             auto simp: eval_nat_numeral le_Suc_eq)
qed

lemma F_seq_dec: assumes p_one: "p=1" and "0 < x"
shows "⟦x<(n::nat)∗(x::real);n∗x≤1⟧
                         ⟹ F(n∗x)<F((n-(1::real))∗x)"
proof (induction n rule: nat_0_1_2_2imp3_induct)
  case 0
  then have "x<0" by simp
  moreover have "0 < x" using assms(2) by simp
  ultimately show "F(real 0∗x)<F((real 0-1)∗x)"
    by simp
next
  case 1
  then have "x<x"  by simp
  then show "F(real 1∗x)<F((real 1-1)∗x)" by simp
next
  case 2
  then have "0<x" and  "x≤p" using p_one by auto
  have "F (2∗x) = (S (2∗x))/(2∗x)" using F_def
    by simp
  also have "... = (2∗(S x)∗(C x))/(2∗x)"
    by (simp only: S2x)
  also have "... = (F x)∗(C x)" using F_def by simp
  also have "... < F x"
  proof -
    from `0<x` `x≤p` have "C x <1"
      by (rule C_strict_less_one)
    from `0<x` `x≤p` have "S x > 0"
      by (rule S_strict_greater_zero)
    from this `0<x` have "F x > 0"
```

```
      unfolding F_def by simp
    from this `C x < 1` show "F x * C x < F x"
      by (simp add: mult_less_cancel_left_pos)
  qed
  finally show "F (real 2 * x) < F ((real 2 - 1) * x)"
    by simp
next
  case 3
  then have F2: "F(real 2*x)<F((real 2-1)*x)"
    by simp
  from 3 have "0 < x" by simp
  moreover have "real 2 * x ≤ p" using 3 p_one
    by auto
  moreover have "real 2 >1" by simp
  ultimately have "F((real 2+1)*x)<F((real 2)*x)"
    using F2 by (rule F_seq_dec_lemma)
  then show "F(real 3*x)<F((real 3-1)*x)" by simp
next
  case step
  fix n::nat
  assume "3 ≤ n"
  assume imp: "x < real n * x ⟹ real n * x ≤ 1
          ⟹ F (real n * x) < F ((real n - 1) * x)"
  assume aSucn:"x < real (Suc n) * x"
  assume "real (Suc n) * x ≤ 1"
  have "real n ≤ real (Suc n)" by simp
  then have "real n * x ≤ real (Suc n) * x"
    using `0 < x`
    by (simp add: mult_le_cancel_left_pos)
  then have "real n * x ≤ 1"
    using `real (Suc n) * x ≤ 1` by simp
  have "real n > 1" using `3 ≤ n` by simp
  then have "x < real n * x" using `0 < x` by simp
  from `real n * x ≤ 1` `x < real n * x`
    have Fn:"F (real n * x) < F ((real n - 1) * x)"
    by (simp only:imp)
  from assms(2) have "0 < x" by simp
  moreover have "real n * x ≤ p"
    using `real n * x ≤ 1` p_one by auto
  moreover have "real n >1" using `3 ≤ n` by simp
  ultimately have "F((real n+1)*x)<F((real n)*x)"
    using Fn by (rule F_seq_dec_lemma)
  moreover have "real n +1 = real (Suc n)" by simp
  moreover have "real n = real (Suc n) - 1" by simp
  ultimately show
    "F(real (Suc n)*x)<F((real (Suc n)-1)*x)" by metis
qed
```

```
lemma F_seq_dec_inducted: assumes p_one: "p=1"
      and "(m::nat)≤n-1" and "0 < x" and "m>0"
shows "⟦x<(n::nat)∗(x::real);n∗x≤1⟧ ⟹ F(n∗x)<F(m∗x)"
proof (rule_tac i="m" and  j="n-1" in  inc_induct)
  show "(m::nat)≤n-1" by (subst assms(2)) simp
  assume "x< (n::nat)∗(x::real)"
  assume "n∗x ≤ 1"
  from `p=1` `0 < x` have
    almost:"⟦x<(n::nat)∗(x::real);n∗x≤1⟧
                          ⟹ F(n∗x)<F((real n-1)∗x)"
    by (rule F_seq_dec)
  from `0 < x` `x< (n::nat)∗(x::real)`
    have "real n - 1 = real (n - 1)" by simp
  from almost `x< (n::nat)∗(x::real)` `n∗x ≤ 1` this
    show "F (real n ∗ x) < F ((real (n - 1)) ∗ x)"
    by simp
  fix N
  assume "N<n-1"
  assume "x < real n ∗ x"
  assume  "real n ∗ x ≤ 1"
  assume  "m ≤ N"
  assume  "F (real n ∗ x) < F (real (Suc N) ∗ x)"
  from `0<m` `m ≤ N` have "0<N" by simp
  then have "1 < Suc N" by simp
  from this `0<x` have "x < (Suc N)∗x" by simp
  from `N<n-1` have "Suc N < n" by simp
  from this `0<x` have "(Suc N)∗x≤n∗x" by simp
  from this `real n ∗ x ≤ 1` have "(Suc N)∗x≤1"
    by simp
  from `p=1` `0<x` `x < (Suc N)∗x` `(Suc N)∗x≤1`
    have "F(real (Suc N)∗x)<F((real (Suc N)-1)∗x)"
    by (rule F_seq_dec)
  then have "F (real (Suc N) ∗ x) < F ((real N) ∗ x)"
    by simp
  from `F (real n ∗ x) < F (real (Suc N) ∗ x)` this
    show "F (real n ∗ x) < F (real  N ∗ x)"
    by simp
qed


lemma cont_lessthan: fixes f::"(real ⇒ real)"
                     fixes  e y :: real
  assumes "isCont f y" and "e>0"
  shows "∃s>0. ∀t. (y-s<t)∧(t<y) ⟶ ¦f y - f t¦<e"
proof -
```

```
  from assms have
"∀r>0. ∃s>0. ∀x. x≠y∧dist x y<s
                              ⟶ dist (f x) (f y)<r"
    by (simp only: isCont_def LIM_def)
  then have
"∃s>0. ∀x. x≠y∧dist x y<s ⟶ dist (f x) (f y)<e"
    using `e>0`  by (simp only: allE)
  then obtain "s" where s_def:
"s>0∧(∀x. x≠y∧dist x y<s ⟶ dist (f x) (f y)<e)"
    by auto
  then have "s>0" by simp
  moreover have "∀t. (y-s<t)∧(t<y) ⟶ ¦f y-f t¦<e"
  proof (safe)
    fix t
    assume "y-s < t"
    assume "t < y"
    then have "t ≠ y" by simp
    moreover have "dist t y < s"
      using `y-s < t` `t < y` dist_real_def by auto
    moreover have
      "t≠y∧dist t y<s ⟶ dist (f t) (f y)<e"
      using s_def by auto
    ultimately have "dist (f t) (f y) < e" by auto
    then show "¦f y - f t¦ < e" using dist_real_def
      by auto
  qed
  ultimately show ?thesis by auto
qed

lemma exists_n_strict_interval_length_gr_1:
          assumes "(b::real)>1" and "c≥0"
          shows "∃(n::nat). c< n ∧ n<c+b"
proof -
  consider (a) "⌈c⌉>c"| (b) "⌈c⌉=c"
    by linarith
  then show ?thesis
  proof cases
    case a
    moreover have ceil_in_int:"b+c>⌈c⌉"
      using assms by linarith
    ultimately have "c< (nat⌈c⌉)∧(nat ⌈c⌉)<c+b"
      using `c≥0` by auto
    then show ?thesis by auto
  next
    case b
    then have "⌈c⌉+1 < c +b" using `(b::real)>1`
      by linarith
    moreover have "⌈c⌉+1>c" by linarith
    ultimately have "c< nat (⌈c⌉+1)∧nat (⌈c⌉+1)<c+b"
      using `c>=0` by auto
    then show ?thesis by auto
  qed
qed

lemma frac_mult_by_denom:
  "b≠0 ⟹ (a/b)∗b=(a::real)"
  by simp


theorem  F_decreasing_on_0_1:
    assumes x0_gr_0:"0 < (x0::real) "
    and y_gr_x0:"x0<y"
    and y_leq_1:"(y::real)≤1"
    and p_one: "p = 1"
    shows "F (y) < F x0"
  proof-
  have "0 < (y - x0)/x0 " using assms by simp
  from this have "∃(x::nat). 1/(2^x)<((y-x0)/x0)"
    by (auto simp: exists_natpow_lessthan2)
  then obtain k::nat where k_def:"1/(2^k)<(y-x0)/x0"
    by auto
  then have "(1/(2^k))∗x0 < ((y - x0)/x0)∗x0"
    using assms(1) real_mult_less_iff1 by blast
  then have "(1/(2^k))∗x0 < (y - x0)"
    using assms(1) by simp
  def x1 == "x0 + (1/(2^k))∗x0"
  then have "x1 < y" using `(1/(2^k))∗x0<(y-x0)`
    by simp
  have "((2^k)::real)∗(1/(2^k)) = 1"
    by (subst times_divide_eq_right) simp
  then have "x0 = (2^k)∗(1/(2^k))∗x0" by auto
  then have "x1 = (2^k)∗((1/(2^k))∗x0)+(1/(2^k))∗x0"
    using x1_def by simp
  then have "x1 = (2^k+1)∗((1/(2^k))∗x0)"
    by
    (subst semiring_normalization_rules(2)[symmetric])
     simp
  then have x1_equals:
    "x1=(real (2^k+1))∗((1/(2^k))∗x0)"
    by simp
  have "p=1" using p_one by simp
  moreover from x0_gr_0 have gr_0:"0<((1/(2^k))∗x0)"
    by simp
  moreover have
```

```
    "((1/(2^k))*x0)<real (2^k+1)*((1/(2^k))*x0)"
  proof -
    have "real (2 ^ k+1) > (1::real) " by simp
    from gr_0 this have
      "((1/(2^k))*x0)*1<((1/(2^k))*x0)*real(2^k+1)"
      by (subst mult_less_cancel_left_pos)
    then show ?thesis by (simp only: mult.commute)
  qed
  moreover have "real (2^k+1)*((1/(2^k))*x0)≤1"
    using `x1 < y` `(y::real)≤1`
    by (subst x1_equals[symmetric]) simp
  ultimately
    have "F(real(2^k+1)*((1/(2^k))*x0))
                   <F((real(2^k+1)-1)*((1/(2^k))*x0))"
    by (rule F_seq_dec)
  moreover have x0_equals:
    "(real(2^k+1)-1)*((1/(2^k))*x0)=x0"
    by simp
  ultimately have "F x0 > F x1"
    by (subst x0_equals[symmetric]) (subst x1_equals)
  obtain e where e_def: "e = F x0 - F x1" by simp
  then have "e > 0" using `F x0 > F x1` by simp
  moreover have "isCont F y"
    using  x0_gr_0 y_gr_x0 y_leq_1
    by (simp only: F_cont)
  ultimately have
    "∃s>0. ∀t. y-s<t∧t<y ⟶ ¦F y-F t¦<e"
    by (simp only: cont_lessthan)
  then obtain s where s_def:
    "∀t. y-s<t∧t<y ⟶ ¦F y-F t¦<e" and s_gr_0:"s>0"
    by auto
  then have in_particular:
    "y-s<x1∧x1<y ⟶ ¦F y-F x1¦<e" by simp
  consider (a) "x1≤y-s"| (b) "y-s<x1∧x1<y"
                        | (c) "x1≥y"
    by linarith
  then show "F y < F x0"
  proof cases
    case (b)
    then have b:"y - s < x1 ∧ x1 < y" by simp
    have "F y - F x1 ≤ ¦F y - F x1¦" by simp
    also from in_particular b have "¦F y-F x1¦<e"
      by (rule mp)
    also have "... = F x0 - F x1" by (rule e_def)
    finally show "F y < F x0" by simp
  next
    case (a)
    then have a:"x1 ≤ y - s " by simp
    from s_gr_0 have
      "∃(x::nat)>0. 1/(2^x)<(s/x0)*(2^k)"
      using x0_gr_0
      by (auto simp: exists_natpowgr0_lessthan2)
    then obtain j where j_def:"1/(2^j)<(s/x0)*(2^k)"
      and "j>0" by auto
    have "x0*(1/(2^(j+k))) > 0" using x0_gr_0
      by simp
    from j_def have "(1/(2^j))/(2^k) < s/x0"
      by (subst mult_imp_div_pos_less) auto
    then have "(1/(2^j*2^k)) < s/x0"
      by (subst divide_divide_eq_left[symmetric])
    then have "(1/(2^(j+k))) < s/x0"
      by (subst
          semiring_normalization_rules(26)[symmetric])
    then have "x0*(1/(2^(j+k))) < x0*(s/x0)"
      using x0_gr_0
      by (subst mult_less_cancel_left_pos)
    then have "x0*(1/(2^(j+k))) < s" using x0_gr_0
      by simp
    then have "(s/(x0*(1/(2^(j+k)))))>
                (x0*(1/(2^(j+k))))/(x0*(1/(2^(j+k))))"
      using x0_gr_0
      by (subst divide_strict_right_mono) auto
    then have "(s/(x0*(1/(2^(j+k)))))>1"
      using x0_gr_0 by simp
    moreover have "((y-s-x1)/(x0*(1/(2^(j+k)))))≥0"
      (*can only prove this non-strict version*)
    proof -
      from a have "y-s-x1≥0"
        using x1_def s_def y_gr_x0 by auto
      from this `x0*(1/(2^(j+k))) > 0` show ?thesis
        by (rule divide_nonneg_pos)
    qed
    ultimately have
"∃n. ((y-s-x1)/(x0*(1/(2^(j+k)))))<(n::nat)
               ∧n<((y-s-x1)/(x0*(1/(2^(j+k)))))
                           +(s/(x0*(1/(2^(j+k)))))"
      by (rule exists_n_strict_interval_length_gr_1)
    then obtain n::nat where n_def:
"((y-s-x1)/(x0*(1/(2^(j+k)))))<(n::nat)
                ∧n<((y-s-x1)/(x0*(1/(2^(j+k)))))
                            +(s/(x0*(1/(2^(j+k)))))"
      by auto
    then have cond1:"y-s < x1+n*(x0*(1/(2^(j+k))))"
          and cond2: " x1+n*(x0*(1/(2^(j+k)))) < y"
    proof safe
      assume "((y-s-x1)/(x0*(1/(2^(j+k)))))<(n::nat)"
      from this `x0*(1/(2^(j+k))) > 0`
        have
"((y-s-x1)/(x0*(1/(2^(j+k)))))*(x0*(1/(2^(j+k))))
                        <(n::nat)*(x0*(1/(2^(j+k))))"
        by (rule  mult_strict_right_mono)
      then show "y-s < x1+n*(x0*(1/(2^(j+k))))"
        using `x0*(1/(2^(j+k))) > 0`
        by (simp only: frac_mult_by_denom)
      assume "n<((y-s-x1)/(x0*(1/(2^(j+k)))))
                              +(s/(x0*(1/(2^(j+k)))))"
      from this and `x0*(1/(2^(j+k))) > 0`
        have
"n*(x0*(1/(2^(j+k))))<(((y-s-x1)/(x0*(1/(2^(j+k)))))
         +(s/(x0*(1/(2^(j+k))))))*(x0*(1/(2^(j+k))))"
        by (rule  mult_strict_right_mono)
      then have
"n*(x0*(1/(2^(j+k))))
  <((y-s-x1)/(x0*(1/(2^(j+k)))))*(x0*(1/(2^(j+k))))
          +(s/(x0*(1/(2^(j+k)))))*(x0*(1/(2^(j+k))))"
        by (simp only: ring_distribs(2))
      then show "x1+n*(x0*(1/(2^(j+k)))) < y"
        using `x0*(1/(2^(j+k))) > 0`
        by (simp only: frac_mult_by_denom)
    qed
    def x2 == "x1+n*((1/(2^(j+k)))*x0)"
    from x1_def
      have x1_1st_form:
"x1=(2^j)*(1/2^j)*(1/2^k)*x0+(2^(j+k))*(1/2^(j+k))*x0"
      by simp
    have "1/(real 2^j)*(1/(2^k))=1*1/((2^j)*(2^k))"
      by (subst times_divide_times_eq[symmetric])
          simp
    also have "... = 1/(2^(j+k))"
      by (subst semiring_normalization_rules(26))
          simp
    finally have powers_denom:
      "1/(real 2^j)*(1/(2^k))=1/(2^(j+k))"
      by simp
    from x1_1st_form have "x1=(2^j)*((1/(2^(j+k)))*x0)
                           +(2^(j+k))*((1/2^(j+k))*x0)"
      by (subst powers_denom[symmetric]) simp
    then have x1_final_form:
      "x1 = (2^j+2^(j+k))*(1/(2^(j+k)))*x0"
      by (simp only: ring_distribs(2))
    have "2^j+2^(j+k) ≤ n+2^j+2^(j+k)-1"
```

```
      using `((y-s-x1)/(x0*(1/(2^(j+k)))))≥0`
      n_def by auto
    moreover have
 "(1/(2^(j+k)))*x0<(n+2^j+2^(j+k))*(1/(2^(j+k)))*x0"
    proof -
      have "(1::real) ≤(2::real)^j"
        by (rule two_realpow_ge_one)
      moreover have "1  ≤ (2::real)^(j+k)" by simp
      ultimately have
        "1<(n+(2::real)^j+(2::real)^(j+k))"
        by linarith
      moreover have "(1/(2^(j+k)))*x0 > 0"
        using x0_gr_0 by simp
      ultimately have "1*((1/(2^(j+k)))*x0)
                  <(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)"
        by (subst mult_strict_right_mono) auto
      then show ?thesis by simp
    qed
    moreover have
      "(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)≤1"
    proof -
      have
 "(2^j+2^(j+k))*(1/2^(j+k))*x0+n*(x0*(1/2^(j+k)))<y"
        using cond2 x1_final_form by blast
      moreover have "(1/2^(j+k))*x0=(x0*(1/2^(j+k)))"
        by simp
      ultimately
        have
 "(2^j+2^(j+k))*((1/2^(j+k))*x0)+n*((1/2^(j+k))*x0)<y"
        by simp
      moreover have
"((2^j+2^(j+k))+n)*((1/(2^(j+k)))*x0)
  = (2^j+2^(j+k))*((1/2^(j+k))*x0)+n*((1/2^(j+k))*x0)"
        by (subst semiring_normalization_rules(1))
            simp
      ultimately have
        "((2^j+2^(j+k))+n)*((1/(2^(j+k)))*x0)<y"
        by simp
      have "n+2^j+2^(j+k) = (2^j+2^(j+k))+n" by simp
      from `((2^j+2^(j+k))+n)*((1/(2^(j+k)))*x0) < y`
        have "(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0) < y"
        by (subst `n+2^j+2^(j+k) = (2^j+2^(j+k))+n`)
      then show ?thesis using y_leq_1  by linarith
    qed
    ultimately have
      "F(real(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0))
             <F(real(2^j+2^(j+k))*((1/(2^(j+k)))*x0))"
```

```
   using p_one `x0*(1/(2^(j+k))) > 0`
   by (subst F_seq_dec_inducted) auto
  then have x2_expanded:
   "F(real(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0))<F(x1)"
   by (subst x1_final_form) simp
  have "real (n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)=x2"
  proof -
   have "(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)
 = n*((1/2^(j+k)*x0))+(2^j+2^(j+k))*((1/2^(j+k))*x0)"
    by (subst semiring_normalization_rules(1))
      simp
   then have "real(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)
              =x1+real n*(1/2^(j+k)*x0)"
    using x1_final_form by simp
   then show ?thesis using x2_def by simp
  qed
  from x2_expanded have "F (x2) < F(x1)"
   by (subst
`real(n+2^j+2^(j+k))*((1/(2^(j+k)))*x0)=x2`[symmetric])
  from cond1 cond2 x2_def have "y - s < x2 ∧ x2 < y"
   by simp
  from this s_def have "¦F y - F x2¦ < e" by auto
  then have "F (y) - F (x2) < F x0 - F x1"
   using e_def by simp
  then have "F (y) - F (x2) < F x0 - F x2"
   using `F (x2) < F(x1)` by simp
  then show "F (y) < F x0" by simp
 next
  case (c)
  from `x1<y` this show ?thesis by simp
 qed
qed

lemma bound_on_C_frac_seq:
   assumes "p=1"
   shows "2/(1+C (1/(2^n))) ≤ 2/(2-1/(2^n))"
proof -
 from Cpseq have "C (1/(2^n)) ≥ 1-1/(2^n)"
  by (subst `p=1`[symmetric])
 have "(2::real)^n ≥ 1"
  by (rule two_realpow_ge_one)
 then have "0 < 1 / (2::real)^n"
  and "1 / (2::real)^n ≤1" by auto
 from `1 / 2^n ≤1`  have "1 ≤ 2-1/((2::real)^n)"
  by simp
 then have "0 < 2-1/((2::real)^n)" by linarith
 from `0 < 1 / 2^n` `1 / 2^n ≤1`
```

```
    have "1 + C (1 / 2^n)≥1"
    using Cx_greq_0 `p=1` by auto
  then have "1+C (1 / 2^n)>0" by simp
  from this `0 < 2-1/((2::real)^n)`
    have "0<(1+C(1/2^n))*(2-1/2^n)" by simp
  from this `C (1/(2^n)) ≥ 1-1/(2^n)`
    show ?thesis by (subst divide_left_mono) auto
qed

lemma Frecurrence: assumes "x>0" and "x≤p"
                   shows "(F(x/2))^2 = 2/(1+C x)*(F x)^2"
proof -
  from Cx_sq_plus_Sx_sq_eq_1[where x="x/2"]
    have in_terms_of_C:"(C(x/2))^2=1-(S(x/2))^2" by simp
  from Srecurrence have
    "4*(S(x/2))^2*((C(x/2))^2)=(S x)^2"
    by (subst in_terms_of_C)
  then have "4*(S(x/2))^2*((C x+1)/2)=(S x)^2"
    by (subst Crecurrence)
  then have "4*(S(x/2))^2 *(C x+1)=2*(S x)^2"
    by algebra
  from this `x>0` have first_form:
    "2^2 *(S(x/2))^2/x^2 *(C x+1)=2*(S x)^2/x^2" by simp
  have S_equivalence:
    "((S(x/2))/(x/2))^2=2^2 *(S(x/2))^2/x^2" using `x>0`
    by (simp add: power_divide)
  from first_form have second_form:
    "((S(x/2))/(x/2))^2 *(C x+1) = 2*(S x)^2/x^2"
    by (subst S_equivalence)
  have "C x + 1>0" using assms Cx_greq_0 by fastforce
  from second_form have
    "((S(x/2))/(x/2))^2=(2*(S x)^2/x^2)/(C x+1)"
    using `C x + 1>0`
    by (simp only: eq_divide_imp)
  then have "((S(x/2))/(x/2))^2=2/(1+C x)*(S x/x)^2"
    by (simp add: power_divide)
  then show ?thesis by (subst F_def) (subst F_def)
qed


lemma F_bounded_by_twoseq:
   assumes "p=1"
   shows "(F(1/2^(n+1)))^2 ≤(2^(n+2))/(2^n+1)"
proof (induct n)
  case 0
  have "1>(0::real)" and "1≤p" by (auto simp: `p=1`)
  then have "(F(1/2))^2 = 2/(1+C 1)*(F 1)^2"
```

```
    by (rule Frecurrence)
  also have "... = 2"
    using Cp_eq_0 Sp_eq_1 `p=1` F_def by simp
  also have "... ≤ (2^(0+2))/(2^{0}+1)" by simp
  finally show ?case by simp
next
  case (Suc n)
  hence IH: "(F(1/2^{n+1}))^{2} ≤(2^{n+2})/(2^{n}+1)".
  from `p=1` have bound_on_C_frac:
    "2/(1+C(1/2^{n+1}))≤2/(2-1/2^{n+1})"
    by (rule bound_on_C_frac_seq)
  have "1*(1::real) ≤ 2 * 2^{n}"
    by (rule mult_mono) auto
  then have 1:"(1/2^{n+1})≤p"
    using `p=1` two_realpow_ge_one by simp
  from `(1/2^{n+1})≤p` have "(1/2^{n+1})≤(1::real)"
    by (subst `p=1`[symmetric])
  then have "2 - 1 / 2^{n+1} ≥ (1::real)"
    by simp
  then have "2 - 1 / 2^{n+1} > (0::real)"
    by linarith
  have 2: "(1/2^{n+1})>(0::real)" by simp
  from 2 1 have "(F((1/2^{n+1})/2))^{2}
              =(2/(1+C(1/2^{n+1})))*((F(1/2^{n+1}))^{2})"
    by (rule Frecurrence)
  also have
    "...≤(2/(2-1/2^{n+1}))*((2^{n+2})/(2^{n}+1))"
  proof (rule mult_mono)
    show "(F(1/2^{n+1}))^{2}≤2^{n+2}/(2^{n}+1)"
      by (subst IH) simp
    show "2/(1+C(1/2^{n+1}))≤2/(2-1/2^{n+1})"
      by (subst bound_on_C_frac) simp
    have "0 ≤ (2::real)" by simp
    from this `2 - 1 / 2^{n+1} > (0::real)`
    show "0 ≤ 2 / ((2::real) - 1 / 2^{n+1})"
      by (rule divide_nonneg_pos)
    show "0 ≤ (F (1 /2^{n+1}))^{2}" by simp
  qed
  also have
    "...=2^(Suc n+2)/(2^Suc n+1+(1-1/2-1/(2^{n+1})))"
  proof -
    have 1:
"((2::real)/(2-1/2^{n+1}))*((2^{n+2})/(2^{n}+1))
                 =2*(2^{n+2})/((2-1/2^{n+1})*(2^{n}+1))"
      by (rule times_divide_times_eq)
    have 2: "(2::real)*(2^{n+2}) = 2^(Suc n+2)"
      by fastforce
    have "((2::real)+-1/2^{n+1})*(2^n+1)
                       =2^(Suc n)+2-(1/2^{n+1})*(2^n+1)"
      by (subst distrib_right) simp
    also have "...=2^(Suc n)+1+(1-1/2-1/(2^{n+1}))"
      by (subst distrib_left) simp
    finally have 3: "((2::real)-1/2^{n+1})*(2^n+1)
                       =2^(Suc n)+1+(1-1/2-1/(2^{n+1}))"
      by simp
    from 1 show
"(2/((2::real)-1/2^{n+1}))*((2^{n+2})/(2^n+1))
        = 2^(Suc n+2)/(2^(Suc n)+1+(1-1/2-1/(2^{n+1})))"
      by (simp only: 2 3)
  qed
  also have "...≤2^(Suc n+2)/(2^(Suc n)+(1::real))"
  proof -
    have "2^(Suc n) ≥ (0::real)"
      using two_realpow_ge_one by simp
    then have 2: "2^(Suc n) + 1 > (0::real)"
      using two_realpow_ge_one by linarith
    have "1/(2^{n+1})≤1/(2::real)" by simp
    then have 1:
    "2^Suc n+(1::real)+(1-1/2-1/(2^{n+1}))≥2^(Suc n)+1"
      by linarith
    moreover have "(2::real)^(Suc n + 2) ≥ 0"
      by simp
    moreover then have
"0<((2::real)^(Suc n)+1+(1-1/2-1/2^{n+1}))*(2^(Suc n)+1)"
      using 1 2 by auto
    ultimately show
"(2::real)^(Suc n+2)/(2^(Suc n)+1+(1-1/2-1/(2^{n+1})))
            ≤(2::real)^(Suc n+2)/(2^(Suc n)+(1::real))"
      by (rule divide_left_mono)
  qed
  finally show
    "(F(1/2^(Suc n+1)))^2≤2^(Suc n+2)/(2^(Suc n)+1)"
    by simp
  qed


lemma twoseq_bounded_by_four:
                        "(2^{n+2})/(2^n+1)<(4::real)"
proof -
  thm divide_less_eq_1_pos
  have "2^n ≥ (1::real)" using two_realpow_ge_one
    by simp
  then have "2^n+1 > (0::real)" by linarith
  have "2^{n+2} = (4::real)*2^n" by simp
  have "2^n<2^n+(1::real)" by simp
  then have "2^n/(2^n+1)<(1::real)"
    using `2^n+1 > (0::real)` by simp
  then have "(4::real)∗2^n/(4∗(2^n+1))<(1::real)"
    using `2^n+1 > (0::real)` by simp
  then have "(2^(n+2))/(4∗(2^n+1))<(1::real)"
    using `2^(n+2) = (4::real)∗2^n` by simp
  then show ?thesis using `2^n+1 > (0::real)`
    by (auto simp: field_simps)
qed

lemma F_bounded: assumes "p=1"
                 shows "(F(1/2^(n+1)))^2<4"
proof -
  from `p=1` have "(F(1/2^(n+1)))^2≤2^(n+2)/(2^n+1)"
    by (rule F_bounded_by_twoseq)
  also have "... < 4" by (rule twoseq_bounded_by_four)
  finally show ?thesis by simp
qed


lemma F_seq_has_limit: assumes "p=1"
      shows "∃L. (λn. F(1/2^(n+1))) ⟶ L"
proof (subst convergent_def[symmetric],
  rule Bseq_mono_convergent,subst Bseq_def,
  subst real_norm_def,intro exI conjI,safe)
  fix m n::nat
  show "2>(0::real)" by simp
  from `p=1` have "(F (1/2^(n+1)))^2 < 4"
    by (rule F_bounded)
  then have "sqrt((F (1/2^(n+1)))^2) < sqrt(4)"
    by (subst real_sqrt_less_iff)
  then have "¦(F (1/2^(n+1)))¦ < 2"
    by (subst real_sqrt_abs[symmetric]) simp
  then show "¦F (1/2^(n+1))¦ ≤ 2" by auto
  assume "m ≤ n"
  have "(0::real)<1/2^(n+1)" by simp
  def mpl1 == "m+1"
  have "1/2^mpl1 ≤ (1::real)"
    using two_realpow_ge_one by simp
  then have "1 / 2^(m+1) ≤ (1::real)"
    by (subst mpl1_def[symmetric])
  show "F (1 / 2^(m+1)) ≤ F (1 / 2^(n+1))"
  proof cases
    assume "m < n"
    have "1 / 2^(n+1) < (1::real) / 2^(m+1)"
      by (rule divide_strict_left_mono)
```

```
         (auto simp: `m < n`)
    from `0<1/2^{n+1}` this `1/2^{m+1}≤1` `p=1`
      have "F (1/2^{m+1}) < F (1/2^{n+1})"
      by (rule F_decreasing_on_0_1)
    then show "F (1/2^{m+1}) ≤ F (1/2^{n+1})"
      by simp
  next
    assume "  m < n"
    from this `m ≤ n` have "m=n" by linarith
    then show "F (1/2^{m+1}) ≤ F (1/2^{n+1})"
      by simp
  qed
qed

lemma F_limit_unique: assumes "p=1"
        shows " ∃!L.(λn. F(1/2^{n+1})) ⟶ L"
  using LIMSEQ_unique F_seq_has_limit assms by blast

definition F_lim :: "real"
 where "F_lim ≡ (THE L. (λn. F(1/2^{n+1})) ⟶ L)"

lemma F_lim_bestdef: assumes "p=1"
        shows "(λn. F(1/2^{n+1})) ⟶ F_lim"
 by (subst F_lim_def,rule theI',
     rule F_limit_unique,rule assms)

lemma F_limseq_bound: assumes "x>0" and "x≤1"
                                    and "p=1"
          shows "F (1 / 2 ^(n + 1)) ≤ F_lim"
proof -
  have "incseq (λn. F(1/2^{n+1}))"
  proof (subst incseq_def, safe)
    fix m n::nat
    assume "m ≤ n"
    have "(0::real)<1 / 2 ^{n+1}" by simp
    def mpl1 == "m+1"
    have "1 / 2^mpl1 ≤ (1::real)"
      using two_realpow_ge_one by simp
    then have "1 / 2^{m+1} ≤ (1::real)"
      by (subst mpl1_def[symmetric])
    show "F (1 / 2^{m+1}) ≤ F (1 / 2^{n+1})"
    proof cases
      assume "m < n"
      have "1 / 2^{n+1} < (1::real) / 2^{m+1}"
       by (rule divide_strict_left_mono)
          (auto simp: `m < n`)
      from `0<1 / 2^{n+1}` this
```

```
          `1 /2^(m+1) ≤ 1` `p=1`
      have "F (1 / 2^(m+1)) < F (1 / 2^(n+1))"
        by (rule F_decreasing_on_0_1)
      then show "F (1 / 2^(m+1)) ≤ F (1 / 2^(n+1))"
        by simp
    next
      assume "  m < n"
      from this `m ≤ n` have "m=n" by linarith
      then show "F (1 / 2^(m+1)) ≤ F (1 / 2^(n+1))" by simp
    qed
  qed
  moreover from `p=1` have
    "(λn. F(1/2^(n+1))) ⟶ F_lim"
    by (rule F_lim_bestdef)
  ultimately show "F (1 / 2^(n+1)) ≤ F_lim"
    by (rule incseq_le)
qed

lemma F_lim_bound: assumes "x>(0::real)"
                   and "x≤1" and "p=1"
                   shows "F x ≤ F_lim"
proof -
  obtain n::nat where "n>1/x"
    using `x>0` ex_less_of_nat by auto
  have "2^(n+1) ≥ (1::real)"
    by (rule two_realpow_ge_one)
  then have "1/2^(n+1) > (0::real)" by auto
  from `x>0` `2^(n+1) ≥ (1::real)`
    have "0<( 2^(n+1))*(1/x)" by simp
  have "(2::real)^(n+1) > (n+1)"
    by (rule of_nat_less_two_power)
  from `n>1/x` this have "2^(n+1) > 1/x" by linarith
  from this zero_less_one `0<( 2^(n+1))*(1/x)`
    have "1/2^(n+1) < 1/(1/x)"
    by (rule divide_strict_left_mono)
  from `x>0` have "x=1/(1/x)" by simp
  from `1/2^(n+1) < 1/(1/x)`
    have "1/2^(n+1) < x" by (subst `x=1/(1/x)`)
  from `0<1/2^(n+1)` this `x ≤ 1` `p=1` have
    that: " F x < F (1/2^(n+1))"
    by (rule F_decreasing_on_0_1)
  from assms have "F (1/2^(n+1)) ≤ F_lim"
    by (rule F_limseq_bound)
  from this that show "F x ≤ F_lim" by linarith
qed
```

```
lemma F_limit: assumes "p=1"
shows "∀r>0.∃s>0. ∀x. x>0∧dist x 0<s
                              ⟶ dist (F x) F_lim<r"
proof (safe,subst dist_real_def,subst dist_real_def)
  from `p=1` have that:
    "∀r>0. ∃no. ∀n≥no. ¦F(1/2^(n+1))-F_lim¦<r"
    by (subst dist_real_def[symmetric],
       subst LIMSEQ_def[symmetric],rule F_lim_bestdef)
  fix r::real assume "0<r"
  from `0<r` and that have
    "∃no. ∀n≥no. ¦F(1/2^(n+1))-F_lim¦<r" by simp
  then obtain no::nat where
    "∀n≥no. ¦F(1/2^(n+1))-F_lim¦<r" by blast
  moreover obtain n::nat where "n≥no" by blast
  ultimately have "¦F(1/2^(n+1))-F_lim¦<r" by simp
  have "(0::real) < 1/2^(n+1)" by simp
  def npl1 == "n+1"
  have "1 / 2^npl1 ≤ (1::real)"
    using two_realpow_ge_one by simp
  then have "1/2^(n+1) ≤ (1::real)"
    by (subst npl1_def[symmetric])
  have "∀x. x>0∧¦x-0¦<1/2^(n+1) ⟶ ¦F x-F_lim¦<r"
  proof safe
    fix x::real assume "x>0" and "¦x-0¦<1/2^(n+1)"
    from `¦x-0¦ < 1 / 2 ^ (n + 1)`
      have "x<1/2^(n+1)" by simp
    from this `1/2^(n+1) ≤ (1::real)`
      have "x ≤ 1" by linarith
    from `x>0` `x ≤ 1` `p=1`
      have "F x ≤ F_lim" by (rule F_lim_bound)
    from `x>0` `x<1/2^(n+1)`  `1/2^(n+1)≤1` `p=1`
      have "F (1/2^(n+1)) < F x"
      by (rule F_decreasing_on_0_1)
    from `F x ≤ F_lim` `F (1/2^(n+1)) < F x`
      show "¦F x - F_lim¦ < r"
    using `¦ F (1/2^(n+1)) - F_lim¦ < r` by auto
  qed
  from `0<1/2^(n+1)` this
    show "∃s>0. ∀x. 0<x∧¦x-0¦<s ⟶ ¦F x-F_lim¦<r"
    by blast
qed

definition Pi :: "real" where "Pi ≡ 2*F_lim"

definition Sin :: "real ⇒ real"
  where "Sin ≡ (λx. S ((2*x) / Pi))"
```

```
definition Cos :: "real ⇒ real"
  where "Cos ≡ (λx. C ((2*x) / Pi))"

lemma Pi_greq_2: assumes "p = 1" shows "Pi≥2"
proof -
  have "F 1 = 1" using Sp_eq_1 F_def `p = 1` by simp
  moreover from `p = 1` have "F_lim ≥ F 1"
    by (simp only: F_lim_bound)
  ultimately have "F_lim ≥1" by linarith
  then show "Pi≥2" by (simp only: Pi_def)
qed


lemma Pi_gr_0: assumes "p = 1" shows "Pi>0"
proof -
  from assms have "Pi≥2" by (rule Pi_greq_2)
  then show "Pi>0" by linarith
qed

lemma Cosxminusy:
         "Cos(x)*Cos(y) + Sin(x)*Sin(y) = Cos(x-y)"
proof -
  have Pi_stuff: "(2*x)/Pi-(2*y)/Pi = (2*(x-y))/Pi"
    by algebra
  have "Cos(x)*Cos(y)+Sin(x)*Sin(y)
     =C((2*x)/Pi)*C((2*y)/Pi)+S((2*x)/Pi)*S((2*y)/Pi)"
    by (subst Sin_def, subst Cos_def,
        subst Sin_def,subst Cos_def) simp
  also have "... = C((2*x)/Pi-(2*y)/Pi)"
    by (rule Cxminusy)
  also have "... = C((2*(x-y))/Pi)"
    by (subst Pi_stuff) simp
  finally show "Cos(x)*Cos(y)+Sin(x)*Sin(y)=Cos(x-y)"
    by (subst Cos_def)
qed

lemma SinPidiv2_eq_1: assumes "p=1"
                         shows "Sin (Pi/2) = 1"
proof -
  have "Sin (Pi/2) = S ((2*(Pi/2)) / Pi)"
    by (subst Sin_def) simp
  also have "... = S(1)" using `p=1` Pi_gr_0 by simp
  also have "... = 1"
    by (subst `p=1`[symmetric],subst Sp_eq_1) simp
  finally show ?thesis by simp
qed
```

```
lemma frac_preserve_ineq:
      assumes "a≥(b::real)" "d≥0" "c≥0"
        shows "(c∗a) / d≥(c∗b) / (d::real)"
by (simp add: assms(1) assms(2) assms(3)
        divide_right_mono mult_left_mono)

lemma Sinx_greq_0: assumes "p=1" and "Pi ≥ x"
                       and "x ≥ 0"
                     shows "Sin x ≥ 0"
proof-
  have "Sin x  =  S ((2∗x) / Pi)"
    by (subst Sin_def) simp
  also have "... = S (2∗(x/Pi))" by simp
  also have "... =2∗ S (x/ Pi)∗C(x/Pi)" by (rule S2x)
  finally have intermsof_SC:"Sin x=2∗S(x/ Pi)∗C(x/Pi)"
    by simp
  have "Pi≥0"  using `p=1` Pi_gr_0 by simp
  then  have low:"x/Pi≥0" using  `x ≥ 0` by simp
  from `Pi ≥ x` `Pi≥0` have "Pi/Pi≥x/Pi"
    by (auto simp:divide_simps)
  then have up:"1≥x/Pi" using `Pi≥0`
    by (auto simp:divide_simps)
  from low up have "C(x/Pi)≥0"
    by (simp only: Cx_greq_0 `p=1`)
  moreover from low up have "S(x/Pi)≥0"
    by (simp only: Sx_greq_0 `p=1`)
  ultimately show ?thesis by (simp add: intermsof_SC)
qed

lemma scale_abs_ineq: assumes "c>(0::real)"
  shows "(¦a- b¦ < r) = (¦c∗a- c∗b¦ < c∗r)"
proof
  assume "¦a- b¦ < r"
  from this `c>0` have "c∗¦a- b¦ < c∗r"
    by (rule mult_strict_left_mono)
  from  `c>0` have "c = ¦c¦" by simp
  then have "c∗¦a- b¦ = ¦c∗(a-b)¦"
    by (simp add: abs_mult)
  also have "... = ¦c∗a- c∗b¦"
    by (subst right_diff_distrib) simp
  finally have "c∗¦a- b¦ = ¦c∗a- c∗b¦" by simp
  from `c∗¦a- b¦ < c∗r` show "¦c∗a- c∗b¦ < c∗r"
    by (subst `c∗¦a- b¦ = ¦c∗a- c∗b¦`[symmetric])
next
  from `c>0` have "1/c>0" by simp
  assume "¦c ∗ a - c ∗ b¦ < c ∗ r"
  from this `1/c>0` have "(1/c)∗¦c∗a-c∗b¦<(1/c)∗(c∗r)"
```

```
    by (rule mult_strict_left_mono)
  from  `1/c>0` have "1/c = ¦1/c¦" by simp
  then have "1/c*¦c*a-c*b¦ = ¦1/c*(c*a-c*b)¦"
    by (simp add: abs_mult)
  also have "... = ¦1/c*c*a-1/c*c*b¦"
    by (subst right_diff_distrib) simp
  finally have "1/c*¦c * a- c * b¦ = ¦a-b¦"
    using `c>0` by simp
  from `(1/c)*¦c * a - c * b¦ < (1/c)*(c*r)`
    show "¦a-b¦< r" using `c>0`
    by (subst `1/c*¦c * a- c * b¦ = ¦a-b¦`[symmetric])
        simp
qed

lemma S_scale_abs_ineq: assumes "p = 1"
shows "¦(S((2*x)/Pi))/x-1¦<r
          =(¦(Pi/2)*((S((2*x)/Pi))/x)-Pi/2¦<(Pi/2)*r)"
proof-
  from `p=1` have "Pi/2>0 "by (simp add: Pi_gr_0)
  then have "¦(S((2*x)/Pi))/x-1¦<r=
     (¦(Pi/2)*((S((2*x)/Pi))/x)-(Pi/2)*1¦<(Pi/2)*r)"
    by (rule scale_abs_ineq)
  then show ?thesis by simp
qed

lemma forall_scaling: assumes "c>0"
     shows "(∀r. P (r::real)) = (∀r. P (c*r))"
proof safe
  fix r::real
  assume "∀r. P r"
  then show "P (c*r)" by auto
next
  fix r::real
  assume "∀r. P (c*r)"
  then have "P (c*((1/c)*r))" using `c>0` by blast
  moreover have "c*((1/c)*r) = r" using `c>0` by simp
  ultimately show "P r" by simp
qed

lemma ineq_scaling: assumes "c>(0::real)"
                      shows "(c*r >0) = (r >0)"
proof
  assume "(0::real) < c * r"
  then have "c*0 < c * r" by simp
  from `c>0` this show "r >0"
    by (subst mult_less_cancel_left_pos[symmetric])
next
```

```
  assume "r >0"
  from this `c>0` show "c*r >0" by simp
qed

lemma S_rearrangement: assumes "p=1"
shows
  "(Pi/2)*((S((2*x)/Pi))/x)=(S((2/Pi)*x))/((2/Pi)*x)"
proof -
  from `p=1` have "Pi>0" by (simp add: Pi_gr_0)
  then show ?thesis by (simp add: field_simps)
qed


theorem lim_sin_x_over_x: assumes "p=1"
shows "∀r>0.∃s>0. ∀x. x>0∧dist x 0<s
                              ⟶ dist ((Sin x)/x) 1<r"
proof -
  from `p=1` have "Pi/2>0" by (simp add: Pi_gr_0)
  from `p=1` have "2/Pi>0" by (simp add: Pi_gr_0)
  then have "(2/Pi)*(Pi/2) = 1" by simp
  def P ==
      "λr. r>0 ⟶(∃s>0. ∀x. x>0∧dist x 0<s
  ⟶(¦(Pi/2)*((S((2*x)/Pi))/x)-(Pi/2)*1¦<(Pi/2)*r))"
  def Q == "λr s.(λx. x>0∧dist x 0<s
         ⟶(¦(S((2/Pi)*x))/((2/Pi)*x)-(Pi/2)*1¦<r))"
  have "(∀r>0.∃s>0. ∀x. x>0∧dist x 0<s
             ⟶ dist ((Sin x)/x) 1<r)
                   = (∀r>0.∃s>0. ∀x. x>0∧dist x 0<s
                          ⟶ ¦(S((2*x)/Pi))/x-1¦<r)"
    by (subst Sin_def,subst dist_real_def) simp
  also have
  "... = (∀r>0.∃s>0. ∀x. x>0∧dist x 0<s
⟶ (¦(Pi/2)*((S ((2*x)/Pi))/x)-(Pi/2)*1¦<(Pi/2)*r))"
    using `p=1` S_scale_abs_ineq by auto
  also have "... = (∀r. P r)" by (subst P_def) simp
  also from `(2/Pi)>0` have "... = (∀r. P((2/Pi)*r))"
    by (rule forall_scaling)
  also have
"...=(∀r. ((2/Pi)*r) >0
  ⟶(∃s>0. ∀x. x>0 ∧ dist x 0 < s
      ⟶(¦(Pi/2)*((S((2*x)/Pi))/x)-(Pi/2)*1¦
                             <(2/Pi)*((Pi/2)*r))))"
    by (subst P_def) simp
  also have
"... = (∀r. r>0
    ⟶ (∃s>0. ∀x. x>0∧dist x 0<s
       ⟶(¦(Pi/2)*((S((2*x)/Pi))/x)-(Pi/2)*1¦
                              <(2/Pi)∗((Pi/2)∗r))))"
    using `(2/Pi)>0` ineq_scaling by presburger
  also have
"... = (∀r. r>0
     ⟶(∃s>0. ∀x. x>0 ∧ dist x 0<s
        ⟶(¦(Pi/2)∗((S((2∗x)/Pi))/x)-(Pi/2)∗1¦<r)))"
    using `(2/Pi)∗(Pi/2) = 1` by auto
  also have
"... = (∀r. r>0
    ⟶(∃s>0. ∀x. x>0∧dist x 0<s
        ⟶(¦(S((2/Pi)∗x))/((2/Pi)∗x)-(Pi/2)∗1¦<r)))"
    using `p=1` S_rearrangement by presburger
  also have "... = (∀r. r >0 ⟶(∃s>0. ∀x. Q r s x))"
    by (subst Q_def) simp
  also have
    "... = (∀r. r >0 ⟶(∃s>0. ∀x. Q r s ((Pi/2)∗x)))"
    using `(Pi/2)>0` forall_scaling by meson
  also have
"...=(∀r. r>0
   ⟶(∃s>0. ∀x. ((Pi/2)∗x)>0 ∧ dist ((Pi/2)∗x) 0<s
      ⟶(¦(S((Pi/2)∗((2/Pi)∗x)))/((Pi/2)∗((2/Pi)∗x))
                                    -(Pi/2)∗1¦<r)))"
    by (subst Q_def) simp
  also have
"...=(∀r. r>0 ⟶(∃s>0. ∀x. x>0 ∧ dist ((Pi/2)∗x) 0<s
       ⟶(¦(S((Pi/2)∗((2/Pi)∗x)))/((Pi/2)∗((2/Pi)∗x))
                                     -(Pi/2)∗1¦<r)))"
    using `(Pi/2)>0` ineq_scaling by presburger
  also have
 "...=(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)∗x) 0<s
                          ⟶(¦(S x)/x-(Pi/2)∗1¦<r)))"
    using `(2/Pi)∗(Pi/2) = 1` by auto
  also have
"...=(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)∗x) 0<s
                           ⟶ (¦F x - (Pi/2)∗1¦<r)))"
    by (subst F_def) simp
  also have
"...=(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)∗x) 0<s
                ⟶ (¦F x-((2 ∗ F_lim)/2)∗1¦ < r)))"
    by (subst Pi_def) simp
  also have
 "...=(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)∗x) 0<s
                               ⟶ (¦F x-F_lim¦<r)))"
    by simp
  finally have
"(∀r>0.∃s>0. ∀x. x > 0 ∧ dist x 0 < s
 ⟶ dist ((Sin x)/ x) 1 < r)
```

```
     = (∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)*x) 0<s
                                 ⟶ (¦F x-F_lim¦<r)))"
    by simp
  moreover have
"(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist ((Pi/2)*x) 0<s
                                 ⟶ (¦F x-F_lim¦<r)))"
  proof safe
    fix r::real
    assume "r >0"
    moreover from `p=1` have
"(∀r. r>0 ⟶(∃s>0. ∀x. x>0∧dist x 0<s
                                ⟶ (¦F x-F_lim¦<r)))"
      by (subst dist_real_def[symmetric],rule F_limit)
    ultimately have
      "∃s>0. ∀x. x>0∧dist x 0<s ⟶ (¦F x-F_lim¦<r)"
      by blast
    then obtain s where "s>0" and s_def:
      "∀x. x>0∧dist x 0<s ⟶ (¦F x-F_lim¦<r)"
      by blast
    have
   " ∀x. x>0∧dist ((Pi/2)*x) 0<s ⟶ (¦F x-F_lim¦<r)"
    proof (subst dist_real_def, safe)
      fix x::real
      assume "x >0" and "¦((Pi/2)*x)- 0¦ < s"
      from s_def have
        "x>0∧dist x 0<s ⟶ (¦F x-F_lim¦<r)"
        by simp
      have "dist x 0 < s"
      proof (subst dist_real_def)
        from `¦((Pi/2)*x)- 0¦ < s` have "(Pi/2)*x < s"
          using `x >0` `Pi/2>0` by simp
        moreover have "Pi/2 ≥ 1"
          using Pi_greq_2 `p=1` by simp
        ultimately have "x<s"
        proof -
         have "x + - 1 * (Pi / 2 * x) ≤ 0"
           using `0 < x` `1 ≤ Pi / 2` by auto
         then show ?thesis
           using `Pi / 2 * x < s` by fastforce
        qed
        from this `x >0` show "¦x - 0¦ < s" by simp
      qed
      from `x>0` this
        `x>0∧dist x 0<s ⟶ (¦F x-F_lim¦<r)`
        show "¦F x - F_lim ¦ < r"
        by simp
    qed
    from this `s>0` show
 "∃s>0. ∀x. 0<x∧dist (Pi/2∗x) 0<s ⟶ ¦F x-F_lim¦<r"
      by blast
  qed
  ultimately show ?thesis by simp
qed


end
```

Bibliography

[1] R. C. Archibald. 'Mathematics Before the Greeks'. In: *Science* 71.1831 (1930), pp. 109–121.

[2] G. Berkeley. *The Analyst: a Discourse addressed to an Infidel Mathematician*. London: Wikisource, 1734, p. 59.

[3] J. R. Brown. *Philosophy of Mathematics: An Introduction to the World of Proofs and Pictures*. New York: Routledge, 1999.

[4] F. Cajori. *A History of Mathematical Notations*. Dover ed. Vol. 2. Dover, 1929.

[5] P. J. Davis and R. Hersh. *The Mathematical Experience*. 2nd ed. Boston: Houghton Mifflin Company, 1982.

[6] J. Dhombres. 'Les présupposés d'Euler dans l'emploi de la méthode fonctionnelle'. In: *Revue d'histoire des sciences* 40.2 (1987), pp. 179–202. URL: http://www.jstor.org/stable/23632740.

[7] W. Dunham. *Journey through Genius*. New York: Penguin Books, 1991.

[8] W. F. Eberlein. 'The Circular Function(s)'. In: *Mathematics Magazine* 39.4 (1966), pp. 197–201.

[9] Euclid. *Euclid's Elements*. Ed. by D. Densmore. Translation by T. L. Heath. Green Lion Press, 2002.

[10] L. Euler. *Introduction to Analysis of the Infinite*. Translation by J. D. Blanton. Vol. 1. Berlin: Springer-Verlag, 1988.

[11] H. Eves. *An Introduction to the History of Mathematics*. 6th ed. Philadelphia: Saunders College Publishing, 1990.

[12] J. D. Fleuriot. *A Combination of Geometry Theorem Proving and Nonstandard Analysis with Application to Newton's Principia*. 1st ed. London: Springer, 2001.

[13] R. Goldblatt. *Lectures on the Hyperreals: An Introduction to Nonstandard Analysis*. New York: Springer-Verlag, 1998.

[14] M. J. C. Gordon. 'For LCF to HOL: a short history'. In: *Proof, Language, and Interaction*. Ed. by G. Plotkin, C. P. Stirling and M. Tofte. Cambridge, Massachusetts: MIT Press, 2000.

[15] T. Gowers. *How do the power-series definitions of sin and cos relate to their geometrical interpretations?* 2014. URL: http://gowers.wordpress.com/2014/03/02/how-do-the-power-series-definitions-of-sin-and-cos-relate-to-their-geometrical-interpretations/.

[16] T. C. Hales. 'Introduction to the Flyspeck Project'. In: *Dagstuhl Seminar Proceedings*. Vol. 5021. Mathematics, Algorithms, Proofs. 2006.

[17] T. Heath. *Greek Mathematics*. 1st ed. Vol. 2. Oxford: Clarendon Press, 1921, p. 257.

[18] J. Hölzl, F. Immler and B. Huffman. 'Type Classes and Filters for Mathematical Analysis in Isabelle/HOL'. In: *4th Conference on Interactive Theorem Proving*. ITP. 2013.

[19] M. Kline. *Mathematics in Western Culture*. 1st ed. London: George Allen and Unwin Ltd., 1954, pp. 68, 232.

[20] T. J. M. Makarios. 'The independence of Tarski's Euclidean axiom'. In: *Archive of Formal Proofs* (Oct. 2012). Formal proof development: http://isa-afp.org/entries/Tarskis_Geometry.shtml. Thesis: http://hdl.handle.net/10063/2315. ISSN: 2150-914x.

[21] M. McKinzie and C. Tuckey. 'Higher Trigonometry, Hyperreal Numbers, and Euler's Analysis of Infinities'. In: *Mathematics Magazine* 74.5 (2001), pp. 339–168.

[22] L. Meikle and J. Fleuriot. 'Formalizing Hilbert's Grundlagen in Isabelle/Isar'. In: *Theorem Proving in Higher Order Logics*. Ed. by D. Basin and B. Wolff. Vol. 2758. LNCS. Heidelberg: Springer, 2003, pp. 319–334.

[23] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs*. 2009. URL: http://isabelle.in.tum.de/website-Isabelle2009-1/dist/Isabelle/doc/isar-overview.pdf.

[24] J. J. O'Connor and E. F. Robertson. *The Trigonometric Functions*. MacTutor History of Mathematics. 1996. URL: http://www-history.mcs.st-andrews.ac.uk/HistTopics/Trigonometric_functions.html.

[25] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. 1st ed. Berlin: Springer-Verlag, 1994.

[26] J. Rehmeyer. *How to (really) trust a mathematical proof*. 2008. URL: https://www.sciencenews.org/article/how-really-trust-mathematical-proof.

[27] A. Robinson. *Non-standard Analysis*. 1st ed. Amsterdam: North-Holland Pub. Co., 1966.

[28] G. B. Robison. 'A New Approach to Circular Functions, Π and *lim(sin x)/x*'. In: *Mathematics Magazine* 41.2 (1968), pp. 66–70.

[29] P. Scott. *Mechanising Hilbert's Foundations of Geometry in Isabelle*. MSc. 2008.

[30] J. Serre. *How to write mathematics badly*. 2003. URL: https://www.youtube.com/watch?v=tJZpdXWm4Gg.

[31] A. N. Whitehead and B. Russell. *Principia Mathematica.* 1^{st} ed. Cambridge: Cambridge University Press, 1910.