Using a Theorem Prover to verify the proofs of some simple theorems in Mathematics

Stefan Dirnstorfer

May 9, 1997

Supervisor: Dr. Neil Speirs

Department of Computing Science

UNIVERSITY OF NEWCASTLE-UPON-TYNE

Acknowledgements

I would like to thank Dr. Neil Speirs for supervising this project.

Abstract

The Prototype Verification System PVS is a computer aided proof checker built by SRI International Computer Science Laboratory. It provides a powerfull specification language for axioms and theorems based on higher order logic. The interactive proof checker allows the verification of a proof such that logical correct deducing is garanteed.

This project uses PVS in order to formally verify some theorems in geometry.

Keywords: PVS, Logic, Proof, Verifying, Geometry

Contents

1 Introduction

Checking and proving logical specifications is a well established branch of artificial intellegence. The work with exact matters in computer programming or hardware design imposes a highly significant sensitivity to hidden logical errors. Checking the correctness of these systems is a dounting task and often beyond human capabilities. One promising approach is the use of the computers precision itself to check given specifications. Although this branch of artificial intelligence is not yet powerful enough to allow a computer to do the whole work on its own, reasonable progress has been made in mechanizing logic thinking.

The tool which was used in this project was the Prototype Verification System PVS programmed by SRI International Computer Science Laboratory. PVS provides its own specification language based on typed higher order logic capable of expressing axioms and theorems using quantifiers, functions and logical expressions. Finally PVS can help in finding propositions or theorems based upon the axioms by providing powerful built-in simplification rules. Proofs are conducted interactively. A series of commands have to be entered to lead the computer the right way. Since each command allows only true deductions, correctness is assured unless errors are made in the specification.

The practical part of this project was to examine the power of PVS in a mathematical application. The geometric world was chosen for specification and proof checking as the practical part of this work for two main reasons. Firstly a clear and simple set of axioms are available and a lot short proofs were already carried out in this area. And secondly the possibility of visualizing the subject and judging on the basis of a daily experience eases the work with abstract expressions that can be stated in a efficient and common representation available to represent the objects used. (e.g. $\overline{AB}$)

In order to carry out all the work a lot of technical details had to be learned. The aim of this project is to show the general capabilities of this software and the difficulties which are encountered without a detailed introduction into the usage of the software. Since the basis of PVS has some similarities to programming languages most features of the specification language will be explained in relation to them. Understanding the fundamentals of logic and programming is therefore essential for understanding of this dissertation.

2 The specification language

The specification of the geometric axioms and theorems is done in a language especially designed for stating logical relations as they are used for mathematical modeling. All the topics discussed here are taken from manual "The PVS Specification Language"[SOR93]. As idicated above the language of PVS includes many elements of sequential programming languages. The knowledge of these principals is presumed in the following discussion.
Although the specification initially consists of plain ASCII code it is, in order to increase readability, pretty printed using LaTeX. Basically it is only a transformation of predicate names like `EXISTS` and `FORALL` into their mathematical notation $\exists$ and $\forall$. Also the other symbols used are well known in mathematics and are introduced to ease the apprehension. For the needs of geometric notation there are several symbols added to the standard configuration. For example the function `angle(A,B,C)` is written as $\angle ABC$.
The programming environment of PVS is the emacs editor. Many commands written in emacs lisp support the typing and specifying.

2.1 Types

Types are a common feature in most programming languages. Although they imply stricter rules for programmer they enable a simple method of checking for correctness. Many syntactic errors can be detected at an early stage without conducting any calculations or proof attempts on the code. In PVS this type checking has to be done before a proof attempt is started and gives information about the correctness of the syntax within few seconds.

classical types

Like every other programming language PVS provides the basic types for whole numbers (`nat`) or reals (`real`). Due to the different aims in programming and logical specification it is not sufficient for PVS only to be supplied with rules for the evaluation of calculations performed on these types. There must also be an implementation of all the rules necessary to simplify expressions involving variables of these types. The most essential rules are for example associativity and commutativity. They are declared as axioms in a prelude file which is

loaded automatically and contains all kinds of useful definitions to be used in further specification in PVS. However the fact that types for numbers are specified in PVS language itself means that they are derived from a much more basic type provided by PVS.

Abstract types

Abstract types are types where nothing about their behavior and properties is assumed a priori apart from the assumption that the type is non-empty, i.e. a variable of this type has at least one possible value. It is possible to specify possible values for an abstract type and to declare the behavior of this type in the context of certain functions in an axiomatic form. In the Geometry the type "Point" is an abstract type. There is no formula given that creates a Point instance from a set of coordinates.

Subtypes

Subtypes can be used to derive a new type from a parent type by selecting some values of parent type to represent the new sub type. This can be compared to limited form of inheritance in object oriented languages. A variable of a subtype is always at the same time of its parent type, but not necessarily the other way around. This type is represented by "FROM $< ParentType >$".
Very useful is the facility to leave sub types abstract. That means that there is no way assumed to deduce if a variable is of a particular subtype. This can be used to derive the type "Line" of the type "Shape". Latter one represents an arbitrary geometric object. Obviously there is no simply way to determine if something is a straight line or not. Although you might have a very clear imagination of what a line is, there is no straight forward way to define it logically. It is still open to non-euclidian geometries to change the shapes considered to be lines without touching the definitions made in the first place.

Functions and predicates

Although functions are used in basically all programming languages only a few implement them consequently as types. This makes it possible to take functions as arguments of functions which is the essential feature that makes the logical

specification of higher order. In the code a function type is represented as "[<domain> → <range>]".
The basic idea of a function in PVS differs in some way from the idea of sub-programs. There is no concept of executing it as a procedure. It is simply a mapping of a domain type onto a range type. Whereas in a program there is a unique and straight forward performed way of simplifying or evaluating a function you can choose it to go the other way around in PVS. If you have for example a variable ℓ of type "Line" it can be useful to expand it to a function that produces this line, e.g. $\overleftrightarrow{AB}$.
Functions are used in this project to obtain variables of type "Line", "Segment", "Ray" and "Angle" from a tuple of Points. It is worth mentioning that these functions are, like their range types, defined to be abstract.
A certain type of function with boolean range, i.e. that return only true or false, are termed predicates. The best known predicates are the quantifiers $\forall$ and $\exists$. A binary predicate used in geometry is congruence "$\cong$" which is a equivalence relation on the set of shapes.

Sets

The set type is a sophisticated type predefined in the prelude file. It is internally represented as a predicate over a given type, that determines if a variable is a member or not. Access to the set type can be performed through well known functions $\in$, $\subset$, etc. Since many geometric relations like "intersecting" or "laying on" can easily expressed in term of set theory ($\cup$, $\in$). The type "set of points" underlies all geometric shapes. The identifier for this type is "setof[Type]".
The geometric type "Shape" is implemented as a set of Points. The complete set of all Points is whole space, in this case the plane. All shapes are subsets of this plane.

2.2 Expressions

The PVS language provides the standard set of expression constructs. There are arithmetic and logic operators was well as IF-THEN-ELSE clauses. Expressions yielding the type boolean are often referred to as propositions. They are used to state relations between elements of the geometric world using the discussed

operators.

Binding expressions

A important class of predicates are those that generate a local scope for variables. Since geometric relations can only be expressed with reference to instantiated variables basically all propositions bind their own variables. Most commonly used are the quantifiers $\forall$ and $\exists$. The syntax for the $\forall$ quantifier is then:

$$\forall(x_1, x_2, \ldots : \text{ Type}) : \text{ predicate}(x_1, x_2, \ldots)$$

The variables $x_1, \ldots$ are then locally valid within the body of the quantifier.

Set expressions

The specification itself does not explicitly refere to the internal representation of sets as predicates. During the proof process sets are often reduced to their original predicate form. Memberships $A \in \overleftrightarrow{BC}$ is expanded as $\overleftrightarrow{BC}(A)$.
One representation of predicates is similar to the mathematical construct for sets which takes the following form: { x | predicate(x) }. This expression is now a predicate itself which yields true for all x that satisfy the predicate. In this case the result is predicate itself. The power of this construct is to bind a variable and produce a predicate from a boolean expression. Reading the set type as a predicate is exactly how this construct is used to define set in mathematics. The intersection of two sets S_1 and S_2 would then be { x | $S_1(x)$ $\wedge$ $S_2(x)$ }

2.3 Theories

A specification covering axioms, theorems, Types and variables is combined in a theory. A theory is comparable to a program or a library in a programming language. This project includes two theories named "Hilbert" and "Lobachesky". The assuming part, bounded with the keywords "ASSUMING" and "ENDASSUMING", contains the axioms and declarations. Outside the assuming part theorems and conjectures are specified. The Hilbert theory for example has an assuming part declaring the types ("Point",...), the functions ($\overleftrightarrow{PP'}$,...) and a series of axioms.

The types are declared at the beginning of the assuming part initiated with the keyword "TYPE". The types can be left abstract or set equal to other types. The functions are variables a function type. The function name is given the type as stated behind the colon. After pretty printing this has a rather odd form in the Hilbert theory (e.g. $\overleftrightarrow{PP'}$). The arguments (P, P′ and P″) are marked with function identifier (e.g. $\leftrightarrow$). The function type reveals what type this identifier leads to (e.g. Line).
The axioms and conjectures have an identifier (e.g. "LineExists") that allows reference during proofs. The keywords used to declare assumptions are "AXIOM" and "POSTULATE" which are treated by PVS to be equivalent. Conjectures are introduced with the keywords "LEMMA" or "THEOREM". There are many more keywords which have one of these two meanings. The keyword is then followed by a boolean expression containing the assumed fact or the conjecture to be proven.

3 Specifying geometric axioms

The basic concepts this specification was already discussed in the previous section. In this section the emphasis is on the meaning of the logic expressions stated in this work and why certain formulations are chosen. The axioms are taken from the book "Foundations of Euclidean and non-Euclidean geometry" [L.F83] Since the specification in the book were done in a human language a not necessarily trivial translation into logic expressions had to be performed.
The axioms are taken from the so called Hilbert geometry. This axiom system differs from Euclidean axioms in that it does not imply any statement on the parallel axiom. This enables the possibility to extend the given axioms to Euclidean and several non-geometries without losing any theorems proven in Hilbert geometry.

3.1 Introducing the axioms

The following list explains the meaning of each axiom in the specification. The axioms are listed below with reference to the original formulation in italic. Many adaptions had to be made to meet the the specialty of the PVS language.

The incidence axioms

LineExists *For every two points A, B there exists a line that contains each of the points A,B.* Actually the PVS specification does not only pronounce the existence of the line. It implicitly defines that the abstract function `line` generates exactly this line. A dilemma at this point is that if the two points are equal what does the function $\overleftrightarrow{AA}$ produce? In fact it is not specified which of the infinite possible solutions is chosen. It would have been possible to define `line` as a partial function such that only arguments different from each other are allowed. But this would mean that one has to explicitly check every time the `line` function is called if the two points are really different.

OnlyLine *For every two points A, B there exists no more than one line that contains each of the points A, B.* Lines are often referred to as being the shortest connection of two points. The definition of length it extremely difficult to implement and depends on wether Euclidean or non-Euclidean geometry is used. At least it is possible to state that all "shortest connections" have to be the same.

ThreeExist, TwoPointsOnLine *There exists at least two points on a line. There exist at least three point that do not lie on a line.* This is the only axiom that assures that there is more that one point in the plane. It would be easy to satisfy all the axioms if the geometric plane contained only of one line or even only one point. However since most of the theorems make statements over all quantified points with certain properties the axiom "ThreeExist" is never used in proofs included in this work.

The order axioms

The original idea of calling three points being ordered as A-B-C is implemented together with the definition of the segment. An ordering A-B-C is syntactically equivalent to B lying on the segment $\overline{AC}$ but not being equal to A or B.

SegmentBounds The segment $\overline{AB}$ is defined as the set of all points P ordered as A-P-B together with the points A and B. Therefore the membership of A and B in the set $\overline{AB}$ has to be stated explicitly.

SegmentOnLine, SegmentSymmetry *If a point B lies between a point A and point C then the points A, B, C are three distinct points of a line, and B also lies between C and A* The first part of this axioms is implemented as the segment $\overline{AB}$ is a subset of the line $\overleftrightarrow{AB}$. I.e. all points that lie on the segment also lie on the line. The symmetry of the segment function is stated as a separate axiom straight forward as $\overline{AB}$ equals $\overline{BA}$.

SegmentExtension *For two points A and C, there always exists at least one point B on the line $\overleftrightarrow{AC}$ such that C lies between A and B.* This axiom is called "SegmentExtension" because is enables prolonging the segment to one side such that the new segment contains the old one.

SegmentOrder *Of any three points on a line there exists no more than one that lies between the other two.* The order relation in contrast to the definition of the segment requires the points to be different from each other. This is therefore explicitly stated in this axiom. Since the all quantified variables can be instantiated in any order it is sufficient to deduce from one particular order the invalidity of any other order.

TriangleIntersection *Let A, B, C be three points that do not lie on a line, and let ℓ be a line which does not meet any of the points A, B, C. If the line passes through a point of segment $\overline{AB}$, it also passes through a point of segment $\overline{AC}$ or through a point of segment $\overline{BC}$.* This axiom is an interesting example of over specification. In geometry the points A, B and C are supposed not to lie on a line in order to construct a triangle. However it is obvious that the final conclusion is also true if A, B, C lie on a line or are even equal to each other. Abandoning the strict idea of a triangle lets us specify this axiom much more succintly. If there is a point E in the intersection of the line ℓ and $\overline{AB}$ there must also be a point D in the intersection of ℓ and either $\overline{AC}$ or $\overline{BC}$. Moreover it is important that if the line does not pass into the triangle via the points A or B it does not pass out via them.

The congruence axioms

CongruentSymmetry The symmetry of the congruence relation was not stated explicitly as an axiom but at some points shapes were called congruent

to each other which implicitly defines this symmetry.

CongruentExtension *If A, B are two points on a line a, and A' is a point on the same or on an other line a' then it is always possible to find a point B' on given side of the line a' through A' such that the segment $\overline{AB}$ is congruent or equal to the segment $\overline{A'B'}$.* For the implementation of this axiom the points A and B do not need to be specified explicitly. It is enough to make reference to the segment $s = \overline{AB}$ constructed by these points. The point A$'$, the line a$'$ or a and the given direction are simply two points coded as A and R. The line is then $\overleftrightarrow{AR}$ and the direction is the ray $\overrightarrow{AR}$. The conclusion of this axiom is then the existence of a point B such that $\overline{AB}$ is congruent to s and B lies on the ray $\overrightarrow{AR}$.

CongruentTransivity *If a segment $\overline{A'B'}$ and a segment $\overline{A''B''}$ are congruent to the same segment $\overline{AB}$, then the segment $\overline{A'B'}$ is also congruent to the segment $\overline{A''B''}$.* This relation is obviously not only true for segments but for all kinds of shapes. Therefore we can state the relation for three shapes s_1, s_2 and s_3 as follows: $s_1 \cong s_2 \wedge s_2 \cong s_3 \Rightarrow s_1 \cong s_3$.

CongruentCombination *On the line a let $\overline{AB}$ and $\overline{BC}$ be two segments which except for B have no points in common. Furthermore, on the same or on another line d let $\overline{A'B'}$ and $\overline{B'C'}$ be two segments which except for B1 also have no points in common In that case, if $\overline{AB} \cong \overline{A'B'}$ and $\overline{BC} \cong \overline{B'C'}$ then $\overline{AC} \cong \overline{A'C'}$* If two segments $\overline{AB}$ and $\overline{BC}$ on one line have only one point B in common then this point must be a border point of both segments an B must lie in between A and C so that the segments do not overlap. Therefore the precondition can easily be specified as $B \in \overline{AC} \wedge B' \in \overline{A'C'}$.

RayDef *For two points A and B, the ray $\overrightarrow{AB}$ is the set consisting of the points of $\overline{AB}$ together with all points C such that A-B-C.* This is just a definition of the item "ray". A ray $\overrightarrow{AB}$ contains all points C on the segment $\overline{AB}$ and the points C that extend this segment on the side of B such that $B \in \overline{AC}$.

AngleDef *Let h,k any two distinct rays emanating from O. The pair of rays h, k is called an angle.* For this project the representation for an angle is defined by three points A,B and C. The angle is then union of the rays $\overrightarrow{BA}$ and $\overrightarrow{BC}$.

An Angle is often referred to as a number measuring its size. In pure geometry an angle $\angle ABC$ is just a shape in the form of a two rays emerging from B in the directions A and C. Since all angles congruent to each other have the same size θ this can be used to measure angles. It can be shown that the sizes θ satisfy all properties necessary to be represented by a subset of real numbers. (e.g. $[0, \pi]$)

CongruentAngle, OnlyCongruentAngle *Let $\angle(h,k)$ be an angle and a' a line and let a definite side of a' be given. Let h' be a ray on the line a' that emanates from the point O'. Then there exists one and only one ray k' such that the angle $\angle(h,k)$ is congruent or equal to the angle $\angle(h',k')$ and at the same time all interior points of the angle $\angle(h',k')$ lie on the given side of a'.*
This axiom is split into two. The first states the existence of an congruent angle the other one assures that only one congruent angle exists. In the first axiom the four variables are bound. the angle α to which a congruent angle is constructed containing the ray $\overrightarrow{AB}$. The variable I determines the side of $\overleftrightarrow{AB}$ on which the congruent angle is constructed. If I lies on $\overleftrightarrow{AB}$ both sides are possible. The variable D constructs the new angle $\angle ABD$ that is congruent α. The second axiom that two angles that are congruent to an angle α and contain a given ray $\overrightarrow{AB}$ are either equal or lie on different sides. If the points C and D lie on different sides of $\overleftrightarrow{AB}$ then the intersection of $\overline{CD}$ with $\overleftrightarrow{AB}$ is not empty.

CongruentTriAngle *If for two triangles ΔABC and $\Delta A'B'C'$ the congruences $\overline{AB} \cong \overline{A'B'}$, $\overline{AC} \cong \overline{A'C'}$ and $\angle BAC \cong \angle B'A'C'$ hold, then the congruence $\angle ABC \cong A'B'C'$ is also satisfied.* The implementation of that is trivial. Primes had to be changed to indices.

3.2 Introducing the conjectures

LineSym This lemma confirms that the arguments for the line function can be given in any order, i.e. $\overleftrightarrow{AB}$=$\overleftrightarrow{BA}$. The necessity of this lemma arises from the required order of the arguments of the line function. In the original axioms the line is defined through two points without emphasis on their order. The truth of this axiom is therefore essential for the correctness of the implementation as a function in PVS. If this did not conclude from the axioms it would have to be specified as an axiom.

SegmentSelfCongruent *Every segment is congruent to itself.* Since transitivity and commutativity are given axioms this theorem concludes that congruence is an equivalence relation.

SegmentFromRay The two rays $\overrightarrow{AB}$ and $\overrightarrow{BA}$ lie in one line have a non empty intersection. This intersection is the segment $\overline{AB}$. This theorem states the equality of two sets and makes therefore propositions for every member of the set which makes this theorem much more complicated for PVS than it looks on the first inspection.

SegmentDivision *Given distinct points A and C, there exists a point D such that A-D-C.* This is the first non-trivial theorem. It says that between every two non-equal points lies another one.

NonCongruence It can not be proven with PVS that there are Shapes that are not congruent to each other. This is discussed in the section about completeness.

3.3 Lobachevskian geometry

In mathematics it is most important that different theories can build upon each other. In PVS theories can be imported using the "IMPORT". Symmetrically the "EXPORT" clause allows control over the definitions in one theory that can be accessed by other theories. PVS keeps track of different files within one so called context which imposes good support for big theories consisting of several files.
This feature is demonstrated in the specification of Lobacheskian geometry which is a non-Euclidean extension of the Hilbert axioms.

Parallel postulate The Parallel postulate defines the existence of non intersecting straight lines. The adjective "parallel" actually has a different meaning than just not intersecting. Whereas in Euclidean geometry there is only one non intersecting line passing through a certain point there are at least two of them in Lobachevskian geometry.

InteriorDef This axioms defines the predicate "interior". We need this for the actual definition of the predicate parallel (||). For a point P and an angle $\angle ABC$ this is defined as laying on the same side as C of the line $\overleftrightarrow{AB}$ and laying on the same side as A of the line $\overleftrightarrow{BC}$. Laying on the same side of lines means beeing connect-able without crossing that line.

ParallelRay *We shall call ray $\overrightarrow{PR}$ parallel to ray $\overrightarrow{QS}$ if they do not meet, and if every ray interior to $\angle RPQ$ meets $\overrightarrow{QS}$.* For the parallelism of rays a so called angle criterion has to be satisfied and the are not allowed to intersect. The angle criterion for two rays $\overrightarrow{PR}$ and $\overrightarrow{QS}$ says that every ray interior to the angle $\angle RPQ$ intersects the ray $\overrightarrow{QS}$.

ParallelLine *Line $\overleftrightarrow{BB'}$ is parallel to line $\overleftrightarrow{AA'}$ (in the direction $\overrightarrow{AA'}$) if the following two conditions are met:*
a. $\overleftrightarrow{BB'}$ does not meet $\overleftrightarrow{AA'}$
b. for some P on $\overleftrightarrow{BB'}$ and Q on $\overleftrightarrow{AA'}$, any ray interior to $\angle$ (opening in the direction of paralleism) will meet $\overrightarrow{QA'}$
The definition of parallelism for lines consists of to parts. First as in Euclidean geometry parallel lines are not allowed to intersect. Second the angle criterion has to be satisfied in the direction of parallelism. The definition of parallel lines distingueshes between the two lines that do not meet a given line and go through a given point.

ParallelSymmetry It is not trivial to see that parallelism is actually a symmetric relation.

3.4 Consistency

A system of axioms is called consistent if it does not produce any wrong propositions. This does not necessarily exclude that putatively meaningless results can be obtained. With the specified axioms all kind of propositions on lines of the kind $\overleftrightarrow{AA}$ could be proven. Testing the consistency of a system of axioms and testing producability of chosen theorems is therefore associated with each other.
The only error source for the results worked out with PVS is, correct implementation of PVS itself provided, the human. Since logical specification has

still to be translated from sentences in human or programming language many mistakes can be made here. The geometric axioms are at least a few centuries old and it is pretty reasonable to assume them to be consistent. But translation into the specification language is still not completely determined.

How to find inconsitencies

As indicated above testing consistency is conducted by trying to deduce correct theorems. It is practically impossible to prove the non deductibility of wrong theorems. It is not sufficient to just check wether correct theorems conclude or not. Assume there was an axiom that by a spelling mistake simplifies to FALSE. Every theorem could be proven within seconds. In this simple case this would become obvious after the first proof. But the error can be hidden in some much trickier way and it always necessary to understand roughly how the result was produced. If correct theorems can not be deduced as expected or are deduced in an unexpected way some concern about the correctness of the axioms has to arise. If the specified area is something where correctness can not already be taken for granted there is always a possibility that the specified object itself has some inconsistencies. This is one of the main application of an interactive prover.

Possible mistakes

Apart from spelling mistakes implicit definitions of properties are a big source of errors. For example the congruence of s_1 and s_2 to *each other* is implicitly defines commutativity and it is easy to forget this. If this relation holds between each other there is no order like $s_1 \cong s_2$ or $s_2 \cong s_1$. But of course in PVS you have to give the arguments of the predicate in a certain order. Another tricky point is the possible equality of instantiated variables. Most of the geometric axioms require the involved points to be different from each other. These statings of inequality are most of the time obvious anyway but space consuming to specify. Therefore they are in their orignal form often stated implicitly.

3.5 Completeness

The set of geometric axioms can be called complete if it deduces every true proposition about geometry within a finite number of steps. A theorem that

can not be deduced from the specified axioms is bound to be false. In fact it is impossible to achieve completeness in higher order logic. Many aspects are left open deliberately to allow the Hilbert axioms to be a basis for Euclid, non-Euclidean and spherical geometry.
Taking the specified Hilberts axioms incompleteness is easy to see. The congruence relation ($\cong$) for instance is incomplete. It is meaningful to apply it to all kind of shapes and is therefore defined as a predicate of two arguments of type "Shape". But it is only specified for segments and angles. Another example is definition of a ray or a segment itself. Every ray that can be produced, using the defined functions, as definitely an origin and a direction. It is not clear wether any instantiation of type ray has this property. In fact, one to one translation of the definition from the book makes this not absolutely clear either. Since only rays instantiated by the `ray` function ($\overrightarrow{PP'}$) are used, this does no harm to the consistency of the axioms.
There are no rules that conclude the absence of a positively defined property. The non congruence of two shapes can not be deduced from the existing axioms. Even a new axiom stating the congruence of all Shapes to each other would be in no way a contradiction to the existing axioms. In mathematics the non congruence of two segments is concluded from the inability to deduce the congruence from a complete set of axioms. PVS does not provide any tools to exclude the deductibility of a theorem. Every proposition has an unclear validity until it is proven to be true. In other words it is impossible to deduce the non congruence of two arbitrary shapes from the specified axioms. The Theorem **NonCongruence** is an example of that.

4 The proof checker

Once all the axioms and theorems are specified one wants to verify their correctness. PVS assists the user to deduce the specified theorems from the given set of axioms. All the commands described in this section are taken from "The PVS Proof Checker: A Reference Manual"[NSR93].
The interactive proof checker can be started from editing environment. The proof checker displays a so called proof sequence containing known facts and unproven theorems. Certain operations can be performed on the sequence by typing interactive prover commands at every step. The goal is to simplify this

sequence until the truth of the theorems can be deduced trivially.
A LaTeXpretty printer transforms the stated commands and the changing proof sequence into a readable form which is free of the cryptic code typed in the first place. Since the whole sequence is rewritten at every step the proofs can become incredibly long. Four proofs contained in this project are printed on eighty pages. In the appendix four pages are printed on one page in order to save paper.
A notable point is that for some odd reason in the pretty printer for the proofs sometimes writes primes as `!1`. I.e. the variable A′ degenerates to `A!1`. This is non pretty printed form that has to be typed into the proofer environment.

4.1 The proof sequence

The proof sequence is a list of propositions consisting of an antecedence and a consequence. The following figure shows the representation of this sequence. The propositions of the antecedence are negatively numbered and the consequence is positively numbered.

{-1}	A_1
{-3}	A_2
{-3}	A_3
⋮	
{1}	B_1
{2}	B_2
{3}	B_3
⋮	

For short it is written within floating text as $A_1, A_2, A_3 \vdash B_1, B_2, B_3$. The interpretation of this sequence is:

$$(A_1 \wedge A_2 \wedge A_3) \Rightarrow (B_1 \vee B_2 \vee B_3) \tag{1}$$

All the the transformings on a sequence can be derived logically from this relation. When a proof is started the sequence has an empty antecedence and a consequence consisting of the theorem which has to be proven. Showing the truth of this theorem is equivalent with showing the truth of the sequence. This is the ultimate goal that terminates the session with the interactive prover.
The proof sequence is the node of a proof tree. Each node represents a subgoal

towards the final proof. During the session axioms and proven theorems can be added to the antecedence, causing the sequence to grow. Many other rules can be applied to simplify the sequence. Once a sequence is proven it disappears from the proof tree. As soon as all nodes are proven the proving process is finished.

4.2 Propositional rules

Splitting

The normalization process of a sequence may require a splitting into one or more subgoals. The distributive law for the logic operators $\vee$ and $\wedge$ allows the following three possible ways of splitting into two subgoals which have to be proven separately.

1. $\Gamma \vdash A \wedge B$ yields $\Gamma \vdash A$ and $\Gamma \vdash B$
2. $A \vee B \vdash \Gamma$ yields $A \vdash \Gamma$ and $B \vdash \Gamma$
3. $A \Rightarrow B \vdash \Gamma$ yields $B \vdash \Gamma$ and $\vdash A, \Gamma$

whereas Γ is a set of propositions in the antecedence or consequence.

Splitting can also be achieved by case analysis. A sequence can be split into two, one with a given proposition "A" to be true and one with "A" to be false.

3. $\Gamma \vdash \Delta$ yields $A \wedge \Gamma \vdash \Delta$ and $\Gamma \vdash A, \Delta$

Splitting copies the majority of propositions unchanged into the new subgoals. Possible simplifications of these may then have to be transformed in every single subgoal in exactly the same way. It is advisable to apply splitting at the latest possible stage.

Flattening

The associativity law implies the following rule two split a single propositions within a sequence.

1. $\Gamma \vdash A \vee B$ yields $\Gamma \vdash A, B$
2. $A \wedge B \vdash \Gamma$ yields $A, B \vdash \Gamma$

Propositional axioms

The propositional axioms recognize the truth of a sequence. It is therefore the goal of all the other commands to generate a sequence that can be proven by the propositional normalization rules that have the following form:

1. $\ldots$, FALSE, $\ldots$ $\vdash$ $\ldots$
2. $\ldots$ $\vdash$ $\ldots$, TRUE, $\ldots$
3. $\ldots$ $\vdash$ $\ldots, t = t, \ldots$
4. $\ldots$, A, $\ldots$ $\vdash$ $\ldots$, A, $\ldots$

Taking into account relation (1) these rules are obviously true. These rules are, if possible, applied automatically at every step during the session.

NOT-simplification

One mentionable simplification rule that is applied automatically at every step which which follows directly from (1).

$\ldots, \neg A, \ldots \vdash \ldots$ is equivalent to $\ldots \vdash A, \ldots$

and of course

$\ldots \vdash \ldots, \neg A, \ldots$ is equivalent to $A, \ldots \vdash A, \ldots$

Proof by contradiction This result lifts in some way the syntactic distinction between antecedence and consequence since propositions can easily move from one side to the other. This is known to mathematicians as proof by contradiction where the theorem to prove is just negated and taken as a pre-condition. The contradiction is produced when the pre-condition together with the negated theorem simplifies to FALSE. This is a valid proof in PVS according to the propositional axiom number 1.

4.3 Quantifier rules

The Quantifier rules are comparatively sophisticated and the most crucial ones in simplifying the proof sequence. Skolemizing and instantiating removes a quantifier meeting certain requirements.

Skolemizing

If it is known that there exists a variable that satisfies a existentially quantified proposition you can give it a universal name that is then known throughout the hole sequence. A existential quantifier in the antecedence or a universal quantifier in consequence can be omitted in a certain way (note that $\forall x : A$ in the consequence is $\exists x : \neg A$ in the antecedence). If the a proposition in the antecedence has the form $\exists x_1, \ldots, x_n : A$ whereas the variables x_1 to x_n have to be replaced by unique constants, $c_1 ... c_2$, written as $A[c_1/x_1, \ldots, c_n/x_n]$.

Skolemizing $A \neq A$ One weird thing happens when the new unique constant is used to bind a variable within the proposition A. For example if "$\forall B : \exists A : A \neq B$" is skolemized with the variable A, which not universally bound, you will get "$\exists A : A \neq A$". This looks like a contradiction but in fact it is not because there are two variables called "A" that are distinguished in their internal representation.
In order to avoid this problem you can use primes for the skolemizing constants. The formula above would then be $\exists A : A \neq B'$

Instantiating

Similar to the way of skolemizing you can lift universal quantifiers in the antecedence and existential quantifiers in the consequence. If a universal constant was produced by skolemizing it can be instantiated in a formula than yields a truth for all its possible instantiations. The formula "$\forall x_1, .. x_n : A$" can therefore be rewritten as "$A[c_1/x_1, .., c_n/x_n]$" where c_1 to c_n are universal constants. If for example g and h are globally known variables of type Shape the proposition "$\forall (s_1, s_2 : Shape) : s_1 \cong s_2$" can be instantiated with the g and h. According to order this yields either "$g \cong h$" or "$h \cong g$". Although both are correct deductions you have to decide for one. Choosing the right order for the instantiation is often most difficult part of the proof where some insight into the meaning of the axioms is needed.

4.4 Proof strategies

The simplification of the proof sequence often involves the straight forward application of the discussed low level rules. The PVS prover provides so called

proof strategies which try to apply to low level rules repeatedly. Especially in long proof sequences these strategies can be a big advantage. As a disadvantage they often do not choose the most efficient way to simplify the sequence. Typically is a splitting into an abundance of subgoals and making inappropriate instantiations of of bound variables. Using the strategies is sometimes a bit of a gamble and has often to be followed by the `undo` command.
The following paragraphs list certain strategies by the name they are invoked with. In the pretty printed form these commands are described by a proper English sentence. This section just gives some background information about what has actually to be typed to achieve the results in the geometric proofs.

`skolem!` All of the discussed proofs start with a universal quantification in the consequence. Skolemization is therefore always one of the very first steps. The `skolem!` command skolemizes all propositions in the sequence and tries to skolemize them. It also generates new variable names automatically.

`inst?` Nearly all axioms are universally quantified. Since they are introduced into the antecedence their bound variables have to instantiated. The `inst?` command tries to instantiate universal variables. There is no unique way to do that and only the wrong order of instantiation can spoil the proof. This rule is therefore not suitable to handle tricky situations.

`replace*` This scans the proof antecedence for equalities and uses them to replace every occurrence of the left hand side with its right hand side. Every other boolean expression A in the sequence is treated as the equality $A = TRUE$. The application of this rule is pretty printed as "Repeatedly applying the replace rule". Sometimes the equalities that are used are specified explicitly.

`ground` This command involves splitting flattening and exploiting equalities. The pretty printer states it as "Applying propositional simplification and decision procedures". It is in fact a very useful strategy. Many subgoals are finished on this command. This command also leads often to repeated splitting of a sequence. If the subgoals can not be proven by the `ground` command itself an abundance of subgoals remains. In this case clarity is often lost and other simplifications have to be done in every single subgoal. Sometimes splitting can

be avoided by hiding a proposition in a sequence such that the ground command does not take it into account.

`grind` This command does repeated skolemization and instantiation and tries to simplify the gained formulas. This is pretty printed as "Trying repeated skolemization, instantiation, and if-lifting". This is a very powerful command and can often perform whole proofs on its own after the required lemmas are appended to proof sequence. In particular set expressions often reduce very quickly to basic logical expression which can be proved by simple rules. Since it is bases on the `inst?` and `ground` strategy discussed above, sometimes correct instantiation has to be done by hand.

5 proofs on geometry

In this section the application of the prover is discussed. Three simple geometric theorems are deduced from the specified axioms. The two proofs "SegmentSelf-Congruent" and "SegmentDivision" are copied from the book "Foundations of Euclidean and non-Euclidean geometry"[L.F83]. The proofs as they are printed in the appendix have gone through several unsuccessful attempts are now revised to relatively (!) short and readable. Not all commands occuring in the proofs have been discussed yet. The pretty printer referes to them with sentences that make the steps, like hiding and deleting, self-explanatory.

5.1 Line symmetry

$$\forall(A, B : Point) : \overleftrightarrow{AB} = \overleftrightarrow{BA}$$

This theorem expresses the symmetry of the definition of the line. The line through A and B ($\overleftrightarrow{AB}$) contains the same Points as the line through B and A ($\overleftrightarrow{BA}$). The axiom we need here is obviously **OnlyLine** because the two lines contain two equal points. The axiom **LineExists** proposes the fact that a line contains the Points that construct the line. After appending these two lemmas to the antecedence the `grind` command skolemizes and instantiates the two quantified propositions in the sequence. Since the the order in which the variable A′ and B′ are instantiated in theorems does not really matter this works fine. The proofs is finished.

5.2 Segment is congruent to itself

$$\forall(A, B : Point) : \overline{AB} \cong \overline{BA}$$

To prove that a segment is congruent to itself we generate a new segment that is congruent to given segment $\overline{AB}$ and apply the transitivity of the congruence relation. The new segment that is congruent to the initial one can exists due to the **CongruentExtension** axiom. This axiom has to be instantiated with the segment $\overline{A'B'}$ for the bounded variable s of type Segment. This is quite obvious to the human but since a new function to generate the segment $\overline{A'B'}$ has to be introduced this is a step where automatized instantiation does not work. The variable used to instantiate A and R are completely arbitrary because there is no preference where this new congruent segment lies in space. After this step the new variable C' can be instantiated to produce the segment $\overline{A'C'}$ that is congruent to $\overline{A'B'}$. Now the axiom **CongruentTransivity** can be included to the sequence. After instantiating and simplifying the result is more or less there. The only thing is that the congruence relation in the antecedence and the consequence are symmetric. The symmetry of this relation is defined in the axiom **CongruentSymmetry**. After applying this axiom the proof sequence is simplified straight forwards.

5.3 The Segment as a intersection of rays

$$\forall(A, B : Point) : A \neq B \Rightarrow \overline{AB} = \overrightarrow{AB} \cap \overrightarrow{BA}$$

This theorem concludes basically from the fact that three points are ordered. Therefore every point x is ordered either x-A-B or A-x-B or A-B-x. Since only the points ordered A-x-B lie on both rays the intersection is the segment. The problem here is to bind a new variable x. This binding is done by introducing a predefined axiom with a command called `extensionality`. This new axiom stated in some way that every set that contains the same points is equal. The two shapes that have to equal are instantiated with $\overline{A'B'}$ and $(\overrightarrow{A'B'} \cup \overrightarrow{B'A'})$. After simplification and skolemizing the theorem reads now a point x is in the segment if and only if x is also a member of the the intersection of rays. Now a series of axioms have to appended to the sequence. First as discussed the axiom **SegmentOrder** instantiated to assume the order A-x-B. Then the

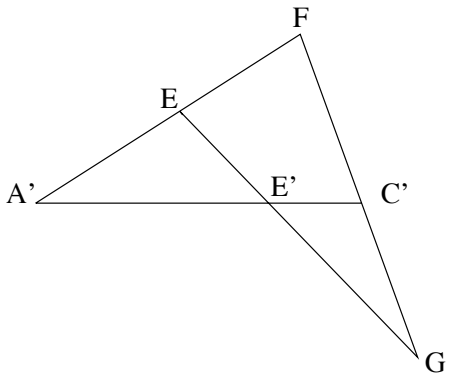
axiom **RayDef** has to be instantiated two times for the two rays $\overrightarrow{A'B'}$ and $\overrightarrow{B'A'}$. The next axiom is **SegmentSymmetry** to solve problems that would arise from two symmetrically specified segments in the ray definitions. The proof sequence can now be simplified with the proof command `flatten`. The more powerful command `ground` for example would at this point generate 18 subgoals. More advisable is splitting the sequence with case analysis into two subgoals. The first subgoal assumes that x lies one the segment $\overline{A'B'}$. This sequence can then be proven with the command `grind`. The second subgoal reduces considerably after applying $x \notin \overline{A'B'}$. The contradiction that is visible at this point is: $B' \in \overline{A'x}$ and $A' \in \overline{B'x}$. Therefore the axiom **SegmentOrder** has to be included again. This time the axiom should conclude from an existsing order A′-B′-x. After simplifying a formula in the antecedence and in the consequence make the same proposition on symmetric declaration of the segment. Therefore **SegmentSymmetry** has to be applied and the proof is finished.

5.4 Dividing a Segment

$$\forall(A, C : Point) : A \neq C \Rightarrow \exists(D : Point) : D \neq A \wedge D \neq C \wedge D \in \overline{AB}$$

The original form of this proof has the length of one paragraph. With PVS it is extended to several pages. This proof demonstrates the difficulties of translating proofs that are readable to humans into PVS.

The basic proof idea works as follows. Let the two points be skolemized as A′ and C′. Now E is a point not on the line $\overleftrightarrow{A'C'}$. The axiom **SegmentExtension** generates the point F on line $\overleftrightarrow{A'E}$. The same axiom the generates the point G on line $\overleftrightarrow{FC'}$. Now the line $\overleftrightarrow{EG}$ hits the triangle in E and must due to **TriangleIntersection** hit the triangle a second time in E′ on $\overline{A'C'}$ or $\overline{FC'}$. The later segment can be excluded because the line $\overleftrightarrow{EG}$ is different from $\overleftrightarrow{FG}$. QED.



The proof with the PVS proof checker starts skolemizing the theorem and instantiating the points line $\overleftrightarrow{A'C'}$ in the axiom **ThreePoints**. The new point is skolemized as E. The the axiom **SegmentExtension** is instantiated twice. First with A′ and E to generate F and then with F and C′ to generate G. Now the axiom the axiom **TriangleIntersection** is applied and instantiated according to the line $\overleftrightarrow{EG}$ hitting the triangle A′FC′. In order to skolemize the variable E' this axiom has to be split the sequence into two subgoals.

1) proposition -1 proposes the existence of the searched point E'. This point can be skolemized and instantiated into the theorem. Using a powerful simplification command like `grind` here produces an abundance of subgoals which can hardly be proofed. In order to stay near the original proof were case analysis on E'$\in \overline{A'C'}$ or E'$\in \overline{FC'}$ is required the other split-able propositions have to be hidden. After that splitting evolves two new subgoals.

1.1) This subgoal is generated to proof that if E′ lies on $\overline{A'C'}$ it does not hit the border in Point A′ or C′. $E' \neq A'$ is already known in the sequence. Therefore the requirement simplifies to $E' \neq C'$. To prove this it has to be shown that the lines $\overleftrightarrow{GC'}$, $\overleftrightarrow{FE}$, $\overleftrightarrow{EG}$, $\overleftrightarrow{A'F}$ and $\overleftrightarrow{FG}$ are equal to each other. Two lines are equal if they share at least two points. The **OnlyLine** axiom has to be instantiated for all these lines. To meet the requirements of this axiom **LineExists** has to be applied to all these lines and the axiom **SegmentOnLine** to the two segments $\overline{FG}$ and $\overline{A'F}$.

1.2) This subgoal proves that E′ cannot lie on $\overline{FC'}$. Here case splitting on $E' = G$ is required.

1.2.1) In this case E' is equal G. This can not be because after replacing all occurrences of E′ with G two contradicting conditions remain. The Points F,G and C′ are said to be order such that $G \in \overline{FC'}$ and $C' \in \overline{FG}$ at the same time. The axiom **SegmentOrder** reduces this statement to $C' = F$. After replacing the occurrences of F with C' the **SegmentOnline** produces that the E that lies on the segment $\overline{A'C'}$ also lies on line line $\overleftrightarrow{A'C'}$. Simplifying concludes the proof of this subgoal.

1.2.2) This is the second case of the case analysis. Here E' is now proven to be not equal to G. This case works like 1.1. Here the lines $\overleftrightarrow{FG}$, $\overleftrightarrow{FC'}$, $\overleftrightarrow{FE'}$, $\overleftrightarrow{EG}$, $\overleftrightarrow{E'G}$, $\overleftrightarrow{EF}$, $\overleftrightarrow{A'C'}$ and $\overleftrightarrow{A'F}$ are equal. The proof of this is very similar and contains mainly instantiating **OnlyLine**, **SegmentOnLine** and **LineExists**.

2) The subgoal number two was generated at the beginning. Here it has to be shown that the line $\overleftrightarrow{EG}$ has a common point with the line $\overleftrightarrow{A'F}$. This common point is obviously the point E. Instantiating and simplifying yields three subgoals that have propositions of the form E $\in \overleftrightarrow{EG}$ in it. All these three subgoals can therefore be proven by instantiating the axiom **LineExists**.

5.5 Remarks on the proofs

Finding proofs

In all four proofs the general idea of how to proof the theorem was already there. None of these ideas were found by any of the proof strategies. At least the right axioms had to be applied and most of time instantiated correctly. Using simplification procedures at an early stage nearly always spoiled the proof.
The only proof with some mathematical relevance is the proof **SegmentDivision.** It could only be completed because it was kept in absolutely parallel to original proof idea. In my first attempts I used, immediatly after entering the basic proof idea, powerful simplification commands like `grind`. As soon as the sequence is split into a series of subgoals by automatized prover commands the clarity about what is going on geometrically is mostly lost. Since the NOT simplification rule allows propositions to move from the antecedence to the consequence and vice versa there is no unique geometric interpretation of a sequence and only few of them promise a working proof idea. Without geomet-

ric imagination I can see no way to produce reasonable results. For instance the idea of showing the equality of this series of lines in step 1.2.2 involves fourteen different instantiations of three different axioms before obtaining the required results. Nobody can calculate so many steps in advance relying on the abstractly stated axioms alone. Taking into account the thousands of other possible instantiations a automatic brute force trial and error algorithm is bound to fail. Only the very last steps or simple subgoals like those produced from subgoal .2 could be simplified by looking at the syntax of the formulas only. By and large the computer is no help in finding geometrical or mathematical proofs of this kind.

Checking proofs

Another important task that PVS is designed for is proof checking. The original proof of the segment division is one paragraph long and can be fully understood within a few minutes. The remaining questions are how thoroughly does one really check all the special cases and how unambiguous are the proofs without making use of previous knowledge of geometric interpretation.
Both questions are hard to tell. Although all special cases arising in the proof are immediately clear after drawing a small sketch, there is always a risk of forgetting about one. Proofs often refer to sketches that visualize the problem but can also mislead or suggest facts that are not specified. Using PVS is not a very efficient way but lets one be definitely sure that the proof is correct.
There is some demand for mathematical proof checking. Often proofs of mathematical theorems, e.g. Fermats last theorem, are found. These proofs can be so complicated that they have to be checked by several mathematicians independently. Even then a slight chance of an error remains. Using mechanized proof checking would be very welcomed. The discouraging length of the simple geometric proofs makes this a very unlikely application of PVS. Checking mathematical proofs by humans, like Euclids theorems are checked by thousands, will continue to be the favored way.

6 Conclusion

6.1 Reflecting the practical work

Comparing the number of pages of specification and proofs it gives a ratio of four to nearly eighty. Although the proofs contain a considerably high amount of computer generated rewriting of theorems it still represents roughly the ratio of difficulties faced at each stage. The thing that attracts attention is that few pages of specification allow a compact stating of the geometric axioms together with even the foundation of a non-Euclidean geometry whereas at the same time a massive pile of pages containing the proofs just reveal the most obvious geometric results. One reason for this is that the similarity of specifying and programming makes the learning of the new language relatively easy. The basis of the interactive prover environment as well as the prover commands dealing with elementary logic are rather unknown in other areas of computer science. But this was probably not the only reason why little headway was made with the proofs. The original proofs are extremely short and simple, but they pretty often refer to facts that are only obvious to human imagination. The powerful simplification rules of PVS were not able to grant sufficient compensation for that.

A positive result of the proof checking process is that an abundance of errors in the first specification were found. It is especially difficult to specify geometric axioms logically since much reference to imagination was made. By and large the small number of successful and unsuccessful proofs revealed big number of mistakes in the specification. Projecting this success to some important specification where incorrectness would cause severe troubles, proof checking can decrease uncertainty considerably. Since many other error sources still remain, the geometric as well as any other PVS checked specifications, are far from proven to be correct.

6.2 Mathematical application

In mathematics one is generally less interested in checking the consistency of axioms but rather in detecting and proof checking of new theorems. This demand does not coincide with the advantages of PVS that were discovered in this project. Proof checking turned out to be relatively hard but gave insight

into the properties and consistency of the axiom system.
Strict implementation of mathematical relations as functions with ordered arguments leads to relatively complicated definitions. The symmetry of certain relation or the irrelevance of the order of arguments in the line function have to be stated as extra axioms or explicitly derived as a lemma. The proofs in this project spend a great part of steps on proofing trivialities like $A \in \overleftrightarrow{AB}$. Introducing more powerful simplification rules that scan the proof sequence and simplify these kind of trivialities would reduce the length of these proofs at least to half size. Programming new rules for PVS is very complex compared to the PVS specification language and is therefore no appropriate way to investigate various mathematical disciplines.
Since mathematics underlies most specifications a mathematical proofs will play an important role in the future of PVS. More theorems in well known mathematical areas will be carried out and allow future works to build upon them.

6.3 Outview

There are definitely a lot of logical problems which require a great deal of standard simplifications without insight into the meaning of the specifications. The advantage of the computer is not to reduce its performace with growing length of the sequence. Many definitions expand to a huge list of simple proposition. For example the expansion of the function "intersection" usually leads to a much longer sequence. When the human loses clarity PVS can still find simple logical properties that simplify the sequence or even prove it. The current state of PVS is from this perspective an encouraging approach. And there are a lot of improvements under way.
PVS is already powerful enough for commercial application. The presentation of a computer generated proof will increase reliability of a designed system considerably. As proof checker become more powerful, proof checking might become an essential requirement for every planed system in the near future.
A mathematical application of PVS or similar proof checkers is quite unlikely in the near future since limitations in current artificial intelligence will also confine the further future of PVS.

C Bibliography

References

[L.F83] Richard L.Faber. *Foundation Of Euclidic And Non-Euclidic Geometry.* Marcel Dekker, Inc., New York and Basel, 1983.

[NSR93] S. Owre N. Shankar and J.M. Rushby. *The Proof Checker: A Reference Manual.* Computer Science Laboratory SRI International, Menlo Park CA 94025, beta release edition, march 1993.

[SOR93] N. Shankar S. Owre and J.M. Rushby. *The PVS Specification Language.* Computer Science Laboratory SRI International, Menlo Park CA 94025, beta release edition, June 1993.