N89-16282

S3-61
167027
16 P.

Formally Verifying Ada Programs which use Real Number Types

Formally
91

David Sutherland

Odyssey Research Associates

Table of Contents

We wish to apply formal verification to programs which use real number arithmetic operations (hereinafter referred to as mathematical programs). Formal verification of a program P consists of (1) creating a mathematical model of P, (2) stating the desired properties of P in a formal logical language, and (3) proving that the mathematical model has the desired properties of step 2 using a formal proof calculus. If the model faithfully embodies P, and the properties of step 2 are a correct formalization of the desired properties of P, the formal verification provides a high degree of assurance that P is correct.

There are two principal difficulties in formally verifying mathematical programs:

1. How to model inexact machine arithmetic operations

2. How to state the desired properties of mathematical programs in view of the fact that such programs in general deliver inexact results (e.g. a square root program does not compute the exact square root)

1 Modeling Machine Arithmetic

Our starting assumption is that machine arithmetic operations can be represented as the ideal real number operations followed by rounding. The operation of rounding is modeled by a cropping function, CR, from the real numbers (denoted by R) to R. The range of CR represents the machine real numbers, sometimes called the model numbers. This was the approach taken in [1], [2], and [3] and is consistent with the proposed IEEE standard for floating point arithmetic [4].

We will assume CR satisfies the following axioms, hereinafter referred to as "the cropping function axioms":

- Axiom 1: The range of CR is finite.

----

1. Mansfield, R., A Complete Axiomatization of Computer Arithmetic I to appear in the Journal of Mathematics and Computation

2. Holm, John, Floating Point Arithmetic and Program Correctness Proofs, Ph. D. thesis, Department of Computer Science, Cornell University, August 1980

3. Brown, W. S., A Simple but Realistic Model of Floating-Point Computations, Computing Science Technical Report No. 83, Bell Laboratories, April 1981

4. A Proposed Standard for Binary Floating Point Arithmetic, Draft 10.0 of IEEE Task P754, Dec. 1982

- Axiom 2: CR(CR(x)) = CR(x)

- Axiom 3: CR(0) = 0

- Axiom 4: [x <= y <= z & CR(x) = CR(z)] -> CR(x) = CR(y)

The first axiom expresses the fact that there are only finitely many machine real numbers. The second axiom says that the result of a rounding operation (i.e. a machine real number) is unaffected by further rounding. Note that the second axiom implies that the range of CR and the set of fixed points of CR are the same. The third axiom says that 0 is a fixed point of CR, i.e. that 0 is a machine real number. The fourth axiom says that if x and z round to the same number and y is between x and z then y rounds to the same number as x and z. As usual when stating axioms in first order logic there are implicit universal quantifiers in front of the formulas displayed as Axioms 2 through 4.

The cropping function axioms are consistent with the four rounding modes which the proposed IEEE Standard would require to be supported, namely rounding to the nearest machine real number, rounding towards 0, rounding towards plus infinity and rounding towards minus infinity. They are also consistent with rounding away from zero, a mode which is not mentioned in the proposed IEEE Standard.

We can derive some useful consequences of the above axioms:

1. CR is monotone, i.e. x <= y -> CR(x) <= CR(y)

2. There is no machine real between x and CR(x).

3. 0 <= x -> 0 <= CR(x) and x <= 0 -> CR(x) <= 0.

Note that the second statement does not imply that there is no machine real that is closer to x than CR(x). Again, we do not wish to require this because the proposed IEEE Standard would require other rounding modes than rounding to the nearest machine real.

2 Modeling Program Execution

We must embed the above ideas about modeling machine arithmetic into a larger model of program execution. We base our formal model of execution on a simple informal picture of program execution. We think of the program as executing a step at a time. At each point in time, the program (or the machine it is running on) is completely described by (1) the "point" in the program where control currently is, and (2) the values of each of the program variables. The program code determines the relationship between the values of variables and the point of control before a given step and after that step. We will assume, for the sake of simplicity, that all variables have a defined value initially, but this value will be unspecified by the execution model. In addition, we will assume that the result of attempting to perform a computation which is undefined (e.g. division by 0) has a completely unspecified effect. To use the model, it will usually be necessary to prove that no undefined computations are attempted, and that the values of program variables are not referenced before they are assigned to.

How do we represent the above informal picture mathematically? We will represent "time" by the non-negative integers (which we will hereinafter refer to as the natural numbers). The "points" where control can reside will be represented simply by a finite set. The data types of program variables other than real number variables will be represented by the corresponding mathematical objects, e.g. the data type of integers will be represented as the mathematical integers. The real data type will be represented by the range of CR.

The execution of the program will be represented by a collection of functions giving the history of the flow of control in the program and the histories of the values of the program variables. Thus, there will be a function from time (i.e. the natural numbers) into the set of control points (which we will denote by PC), and for each program variable v, a function from time into the data type of v.

The functions representing histories will be required to satisfy certain conditions derived from the program. For example, if X, Y and Z are integer program variables, FX, FY and FZ the corresponding history functions, and at a certain time t control is at a program instruction

```
X := Y + Z
```

then the functions must satisfy the condition

$$FX(t + 1) = FY(t) + FZ(t)$$

For real variables, all operations are the ideal real operations followed by cropping. For example, if A, B and C are real program variables, FA, FB and FC the corresponding history functions, and at a certain time t control is at a statement

```
A := B + C
```

then the functions must satisfy the condition

$$FA(t + 1) = CR(FB(t) + FC(t))$$

3 Error Magnitude in the Model

The cropping function axioms capture certain qualitative properties of CR. They are not enough to do useful verification, however, because they say nothing about the size of the error introduced by CR. For example, the cropping function axioms are satisfied by the zero function. Thus, any program which we could verify using only the cropping function axioms would have to be correct even when running on a machine which used the zero function as its cropping function. Very few useful mathematical programs would be correct in any sense on such a machine, and thus we could not be able to verify such programs solely on the basis of the cropping function axioms. We need some additional axioms on the size of the error introduced by CR.

It is not clear, however, what kind of axioms to add. If we add axioms which give specific numerical bounds on the size of the error in a certain range, then any verification we do will only apply to machines that meet these numerical conditions. For a machine that did not meet the conditions, any verification done on the basis of the conditions would be invalid, despite the fact that many programs might still run correctly on the machine. On the other hand, some machines which met the conditions would probably actually meet much more demanding conditions. There could be programs which run correctly on such machines which we cannot prove correct because our axioms do not reflect the high degree of accuracy in the machine.

One solution to this dilemma would be to add non-specific numerical bounds on the error. In other words, add a symbol (say, "e") and add an axiom like "the percentage error between x and CR(x) is always less than e." One could then verify statements about the accuracy of mathematical programs in terms of e. For example, if P were a program to compute square roots, one might try to verify a statement like "the percentage error between P(x) and the square root of x is 5*e." If one then wanted a certain degree of accuracy from P, one could solve for the degree of accuracy in CR that would be necessary to achieve the desired accuracy from P.

There are several problems with this approach. First of all, it is very costly. With present technology in automatic theorem proving, the problem of generating and proving statements of the kind mentioned above in a mechanical proof system is intractable in terms of both the amount of computational power and the amount of human input required. Second, in some situations it forces us to do an analysis that is more detailed than necessary. Many errors in mathematical programs occur at a much lower level of numerical complexity. For example, ZBRENT is a Fortran subroutine from the IMSL library which is supposed to find a zero of a user-defined function F given a pair of endpoints A and B such that the values of F at A and B are of opposite sign. It does this by gradually moving the endpoints inward, always making sure that the values of F at the current endpoints are of opposite sign. In the process of the computation, it generates various pairs of real values X and Y which it must test to see if F(X) and F(Y) are of opposite sign. It does so by multiplying F(X) and F(Y) together and testing whether the result is negative or not. This is an incorrect (not to mention inefficient) test, since it is possible to have F(X) and F(Y) be small numbers of opposite sign whose product is so small that underflow causes the machine to compute 0 for their product. This causes ZBRENT to act as if F(X) and F(Y) are of the same sign, giving incorrect results in some cases. This programming error is not "numerical" in nature, but is inherent in the notion of inexact (although "close") computation.

What we would like is a model of machine arithmetic which captures the idea of "close" but inexact computation without referring to specific numerical constants. In the next section we present such a model. The model is based on an alternate approach to real analysis called <u>non-standard analysis</u>.

4 Non-standard Analysis

Calculus was developed in the eighteenth century based on the notion of infinitesimals. These were positive entities dx smaller than any actual positive real but not 0. Furthermore, they obeyed the laws of ordinary real arithmetic so that one could carry out ordinary algebraic manipulations like

```
y = x^2

y + dy = (x + dx)^2

(x + dx)^2 = x^2 + 2 * x * dx + (dx)^2

dy = 2 * x * dx + (dx)^2

dy/dx = 2 * x + dx
```

In particular the derivative, dy/dx, was the actual quotient of two infinitesimals.

Attempts in the nineteenth century to justify working with these extended reals were not successful and a different approach and proof technique in terms of limits was adopted instead (the so-called epsilon/delta method.)

In the early 60's logicians showed how to justify working with actual infinitesimals using so-called "non-standard models of the reals." These models are ordered algebraic structures which have all the same algebraic and ordering properties of the standard real numbers, and which contain the standard real numbers, but which also contain additional, non-standard numbers. Doing real analysis by means of such non-standard models is called <u>non-standard analysis</u>.

4.1 Non-standard Models

What exactly do we mean by a "non-standard model" of some mathematical object like the real numbers? First of all, by "mathematical object" we will just mean a non-empty set. Before we give a precise statement of "non-standard model", we must discuss the notion of a <u>first-order statement</u> about a mathematical object.

Suppose we have a mathematical object M. A <u>term of M</u> is an expression which is of one of the following forms:

1. An element e of M
2. f(t1,...,tn) where f is an n-ary function from M into M and t1,...,tn are previously defined terms of M.

Thus, if M is the real numbers, then 0, 1 and 1 + exp(5) are terms on M (where exp stands for the "e-to-the-x" function and + is the usual addition function, written infix).

A <u>first-order statement about M</u> is a statement of one of the following forms:

1. p(t1,...,tn) where p is an n-ary predicate on M
2. A statement built up from finitely many previously constructed first-order statements by the use of logical connectives (e.g. "not", "and", "or", "if-then-else", etc.)
3. A statement of the form "for all x in M, ..." where ... is a previously constructed first-order statement involving the variable x.
4. A statement of the form "there exists x in M such that ..." where ... is a previously constructed first-order statement involving the variable x.

The following are first-order statements about the real numbers:

0 < 1

not (5 = 1)

for all x in the real numbers, for all y in the real numbers,
x + y = y + x

there exists x in the real numbers such that for all y in the real numbers, x*y = y

there exists x in the real numbers such that x*x = -1

Notice that the first four statements are true of the real numbers, whereas the fourth is false of the real numbers. A first-order statement about M need not be a true statement about M; it need merely be of a certain form.

In general, there will be some facts about a given mathematical object M which can be expressed as first-order statements and some which cannot. The first four examples above are facts about the real numbers which are expressible as first-order statements. A fact about the real numbers which is not expressible as a first-order statement is the fact that every non-empty set of real numbers which has an upper bound has a least upper bound (this property is called completeness). This statement is not a first-order statement as written because it refers to sets of reals rather than just individual reals. Some statements which refer to sets of elements or other higher-order structures turn out to be equivalent to first-order statements. For example, the statement "for every bounded set S of real numbers, there is a real number x that is not in S" is not in the form of a first-order statement, but it is equivalent to the first-order statement "for all x in the real numbers, there exists y in the real numbers such that x < y." It can be shown that the completeness property is not equivalent to any first-order statement.

We will now define what we mean by a non-standard model. Suppose we have some set M (e.g. the set of real numbers). A non-standard model of M consists of:

1. A set M'

2. For each element e of M, a corresponding element e' of M'

3. For each n-ary function from M into M, a corresponding n-ary function f' from M' into M'

4. For each n-ary predicate p on M, a corresponding n-ary predicate p' on M'

such that every first-order statement which is true of M is true of M' when the elements, functions and predicates in the statement are interpreted as the corresponding elements, functions and predicates of M'. For example, suppose R' is a non-standard model of the reals. Let +' denote the binary function on R' corresponding to the addition function on the reals. Since + is commutative, and since commutativity of + is expressible as a first-order statement (see the examples above), +' must be commutative on R'. On the other hand, R' need not have the completeness property, and there are non-standard models of the reals which are not complete.

We will call the elements of M' which correspond to elements of M the standard elements of M'. We can identify elements of M with their ocrresponding elements of M', and thus speak of M as being a subset of M'. Under this identification, for each function f and each predicate p on M, the corresponding f' and p' on M' is extends f and p respectively. We will call a non-standard model M' of a mathematical object M a proper non-standard model of M if there is an element x of M' which is not in M.

It can be shown (we will not give the proof here) that every infinite mathematical object M has a proper non-standard model M'. The same does not hold for finite mathematical objects. The reason is simple. Suppose M = {e1,...,en}, and M' is a non-standard model of M. It is a true first-order statement about M that "for all x in M, x = e1 or x = e2 or ... or x = en" (the conjunction is finite). Therefore, the statement "for all x in M', x = e1' or x = e2' or ... or x = en'" is true of M', but this says that the only elements of M' are the standard elements.

4.2 Non-standard Models of the Reals

What does a proper non-standard model of the reals look like? It can be shown that every proper non-standard model of the reals consists of the standard real numbers plus the following three kinds of non-standard numbers:

1. Infinitesimals. These are numbers which are not 0 but which are smaller than any standard non-zero real number.

2. Infinite Numbers. These are numbers which are larger than any standard real number. There are both positive and negative infinite numbers. Every proper non-standard model of the reals must have infinite numbers as well as infinitesimal numbers in order to satisfy the algebraic property that every non-zero number has a multiplicative inverse. The multiplicative inverse of a non-zero infinitesimal is an infinite number.

3. Finite Non-standard Numbers. These are numbers of the form x + i where x is a non-zero standard real and i is an infinitesimal. Such numbers are neither infinitesimal nor infinite, but are not standard either.

In the original formulation of calculus, infinitesimals were informally thought of as non-zero real numbers which were in some sense "arbitrarily small". Thus, the notion of infinitesimals lends itself very well to modeling computation which is inexact, but whose inexactness can be taken to be arbitrarily small.

5 Non-standard Models of Execution

We will incorporate the idea of machine real operations which differ infinitesimally from the ideal operations by using non-standard execution models. A non-standard execution model will be a representation of program execution like that described in section 2, but with the standard mathematical objects replaced by non-standard objects. What exactly does this mean?

First, time will be represented by a proper non-standard model of the natural numbers. A proper non-standard model of the natural numbers consists of the standard natural numbers with infinite elements added. Thus, the history functions will be functions whose domain is a proper non-standard model of the natural numbers.

Second, all data types of program variables other than real variables will be represented by proper non-standard models of the standard data types (if proper non-standard models exist. For example, the data type "boolean" is finite and therefore has no proper non-standard models. Finite data types will be represented in non-standard models of execution by the standard model of the data type). For example, the data type consisting of the positive and negative integers must be represented by a proper non-standard model of the integers (which just looks like the standard integers with both positive and negative infinite numbers added).

What about the data type of machine real numbers? In section 2 we obtained the machine real data type by choosing a cropping function on the ideal reals and taking its range. We cannot replace this type by a proper non-standard of itself, because by the first cropping function axiom, this set is finite and so has no proper non-standard models. Suppose instead that we start with a proper non-standard model of the reals R' and a function CR from R' into R' satisfying the cropping function axioms and the additional axiom (called the "error axiom") that for all finite x in R', CR(x) - x is infinitesimal. This axiom formalizes the statement that on all numbers that are not "large" (i.e. not infinite), the roundoff error is "small" (i.e. infinitesimal). We will use the notation "x == y" to stand for "x - y is infinitesimal."

Unfortunately, there are no such cropping functions. In order for the error axiom to be met, the range of CR must be infinite, which contradicts the first cropping function axiom.

How can we resolve this inconsistency? There are definite cases in which we make use of the first cropping function axiom in verification, so we cannot simply abandon it. What we will do instead is, rather than assuming that CR satisfies the first cropping function axiom, assume that CR satisfies all first-order statements implied by the first cropping function axiom. It can be shown that the first cropping function axiom is not equivalent to any first-order statement, so this is a true weakening of our set of axioms. In addition, it can be shown that the resulting weaker set of axioms is consistent. The first-order consequences of the first cropping function axiom will be more than enough to verify most mathematical programs. In summary, we will represent the machine real data type in a non-standard model of execution as the range of a function CR from a non-standard model of the reals into itself such that CR satisfies cropping function axioms 2 through 4, the error axiom given above, and all first-order statements implied by the first cropping function axiom.

6 Specifying Mathematical Programs

How do we state the properties of mathematical programs we want to prove? Suppose we restrict ourselves to considering programs whose purpose is just to compute some real-valued function. If f is a real-values function of n arguments, and P is a program to compute f with parameters A1,...,An, we can state the specification of P in terms of the above formalism simply as "for all inputs x1,...,xn, P(x1,...,xn) == f(x1,...,xn)" or, in slightly more detail, "for all inputs x1,...,xn, if P is executed with the initial values of A1,...,An being x1,...,xn respectively, then P will eventually terminate with output == f(x1,...,xn)." In terms of the above formalism, P has terminated at a time t if PC(t) = stop where "stop" is a control point at the end of the program.

7 An Example Verification

To illustrate the use of the model, we will verify a program which computes the square root function by Newton's method. The proof will be informal. We will denote the ideal square root of a number x by root(x).

Newton's method begins with an initial "guess" at the square root. The guess is then refined by an iterative process. At each step, the current guess g is replaced by (g + (x/g))/2 (where x is the number whose square root is being computed). The only facts about Newton's method we will need to know for the verification are that if x is non-negative and the initial guess is bigger than root(x), then:

1. All succeeding guesses will be bigger than root(x).
2. Each new guess will be less than the previous guess.

We now give the program. We will adopt the convention of writing the symbols for machine real operations "doubled", e.g. machine real addition will be denoted by "++", to distinguish machine operations from ideal operations (which will be denoted by the usual "undoubled" symbols). The value in RESULT is output when the program terminates. The program is:

```
SQRT(X:REAL):REAL

   RESULT := X ++ 1

   LOOP

      IF RESULT ** RESULT <= X

         THEN LEAVE

      IF RESULT <= (RESULT ++ (X//RESULT))//2

         THEN LEAVE

      RESULT := (RESULT ++ (X//RESULT))//2

      END
```

```
END
```

Note that the conditions for leaving the loop are not the kind of conditions one usually sees in programs of this type. The usual approach to terminating iterative processes of this type involves either terminating when a certain degree of accuracy is reached, or when a certain bound on the number of iterations is reached, or both. In SQRT, the iteration is terminated when the iterative process in the machine ceases to act like the ideal Newton's method in one of the two ways mentioned above.

We will now verify that if SQRT is executed with the initial value of X non-negative and finite, then execution eventually terminates with

RESULT == root(the initial value of x)

We will perform the verification by establishing a series of lemmas, leading up to the result we want.

Lemma 0: if x and y are non-negative and x == y, then root(x) == root(y).

Proof: the proof breaks into 2 cases:

Case 1: x and y are infinitesimal. The square of a non-infinitesimal number is non-infinitesimal, so root(x) and root(y) must therefore be infinitesimal, and thus the difference between them is also infinitesimal.

Case 2: either x or y is not infinitesimal. Since the two numbers differ by an infinitesimal, if one is not infinitesimal the other is also not. Since the square of an infinitesimal is infinitesimal, root(x) and root(y) are also non-infinitesimal. By algebra, we have

x - y = (root(x) + root(y))*(root(x) - root(y))

Since the left side is infinitesimal and the first factor of the right side is not, the second factor of the right side must be infinitesimal.

Lemma 1: Whenever (RESULT ++ (X//RESULT))//2 is computed, RESULT is not 0.

Proof: Suppose not. Let t be the earliest time such that PC(t) is at a statement where (RESULT ++ (X//RESULT))//2 is computed and RESULT = 0 at time t. Prior to t, the program must have been executing normally, since division by 0 is the only exceptional condition that can arise (we are ignoring exceptional conditions such as STORAGE_ERROR or overflow which cannot be analyzed on the basis of the program's text).

The only points in the program where (RESULT ++ (X//RESULT))//2 is computed are in the second conditional inside the loop and in the subsequent assignment statement. Since t is the earliest time when a division by 0 is attempted, and program execution before t is normal, we can conclude that:

1. Control at time t must be at the second conditional.

2. Control at time t - 1 must be at the first conditional with RESULT = 0.

3. X at time t - 1 must be negative (by cropping function axiom 3, if RESULT is 0 then RESULT ** RESULT is also 0).

But X is assumed to be non-negative initially, and since no assignments to X can have taken place in the course of normal execution prior to t, X must be non-negative at time t - 1, a contradiction.

We can therefore assume for the rest of the lemmas that the program executes normally at all times.

<u>Lemma 2</u>: The value of X is always the same as the initial value.

<u>Proof</u>: trivial, since there are no steps in the program which assign to X.

<u>Lemma 3</u>: SQRT halts.

<u>Proof</u>: Suppose not. In this case, the set of times t where the value of RESULT decreases from time t to time t + 1 has no upper bound (else at some point control would leave the loop at the second conditional). This fact can be expressed as a first-order statement using the history function for RESULT (call it FRESULT) as follows: "for all times t there exists a time t' such that t < t' and FRESULT(t' + 1) < FRESULT(t')." Howvere, the negation of this statement is a first-order statement which is implied by the first cropping function axiom, a contradiction.

<u>Lemma 4</u>: After the initial assignment to RESULT, the value of RESULT is always >= 0 and <= X ++ 1.

<u>Proof</u>: The proof is by induction on time (i.e induction on the number of steps that have been executed). Immediately after the initial assignment to RESULT, RESULT = X ++ 1 so certainly RESULT <= X ++ 1. We must therefore establish that 0 <= X ++ 1.

Since X and 1 are finite, X + 1 is finite and so by the error axiom, X ++ 1 = CR(X + 1) == X + 1. 1 is not an infinitesimal, and X is non-negative, so X + 1 is at least distance 1 from 0. Since rounding only introduces an infinitesimal error, and the distance between X + 1 and 0 is not infinitesimal, X ++ 1 cannot be 0.

To complete the induction, we must show that at every step in execution, if 0 <= RESULT <= X ++ 1 is true before the step, then it is true after. For execution steps which are not executions of the assignment statement inside the loop, this is trivial, since no other statement changes the value of RESULT. Suppose a given step is an execution of the assignment statement inside the loop. First of all, this means that control must have passed throught the preceding conditional, so the next value of RESULT must be less than the previous value, so if RESULT is <= X ++ 1 before the assignment then the same is true after. Second, as shown in Lemma 2, in order for control to have reached this statement at all, RESULT must be non-zero, so it is strictly positive. The value of X is non-negative. Therefore, since CR of a non-negative number is non-negative, (RESULT ++ (X//RESULT))//2 must be non-negative. This completes the induction.

Lemma 5: RESULT is always finite.

Proof: Since 0 and X ++ 1 are finite and RESULT is always between them, RESULT is also finite.

Lemma 6: When SQRT terminates, RESULT == root(initial value of X).

Proof: We will denote the value of RESULT at termination by R. The proof breaks into three cases:

Case 1: R ** R = X. By Lemma 5, R is finite so by the error axiom, R * R == R ** R = X = initial value of X and the conclusion follows by Lemma 0.

Case 2: R ** R < X. Claim: R * R < X. If not, then R * R >= X, so by the montonicity of CR, R ** R = CR(R * R) <= CR(X) = X, a contradiction. The initial value of RESULT has square > X, so the assignment statement inside the loop must have been executed at least once before termination. Therefore, there exists a previous value of RESULT, call it RP, such that R = (RP ++ (X//RP))//2 <.RP and RP ** RP > X. By the same reasoning as above, the second statement implies that RP * RP > X. Therefore 0 < X/RP < RP so X/RP is finite, so (RP ++ (X//RP))//2 == (RP + (X/RP))/2. But the left side is less than root(x), while the right side is greater than root(x) by property of (ideal) Newton's method. When two numbers which differ by an infinitesimal are on either side of a fixed number, they each differ from that fixed number by an infinitesimal. This establishs the conclusion.

Case 3: R ** R > X. In this case, the program must have terminated because R <= (R ++ (X//R))//2. The assumption of the case implies that R * R > X as above, so 0 < X/R < R so X/R is finite, so (R ++ (X//R))//2 == (R + (X/R))/2. The left side is >= R, while the right side is < R by property of (ideal) Newton's method. Therefore, R - ((R + (X/R))/2) is infinitesimal. Rearranging algebraically, we get (R*R - X)/(2*R*R) is infinitesimal. The denominator is finite, so the numerator must be infinitesimal. The conclusion follows from Lemma 0.

8 The Asymptotic Interpretation

What does verification of a mathematical program executing over a non-standard model of the reals tell us about actual execution on a standard machine? This question is similar to the question "what does a proof in non-standard analysis involving infinitesimals show about analysis in the standard reals?" We will explain heuristically how non-standard analysis proofs relate to standard analysis, and argue by analogy that the same relation holds between verification of non-standard execution and execution on a standard machine.

It can (and has) been proved that the analogy is actually valid, but the proof is beyond the scope of this paper.

Consider the non-standard analysis proof that the derivative of the x^2 function is 2*x. It goes as follows: take an arbitrary infinitesimal i and compute ((x + i)^2 - x^2)/i. The result is 2*x + i. Thus, the value of the difference quotient for any infinitesimal is only infinitesimally different from 2*x. This is actually a proof that the standard x^2 function has derivative 2*x in the usual sense, although it takes some mathematical logic to prove the connection.

What does it mean to say that the derivative of x^2 is 2*x in standard analysis? It means that the limit of the expression (x + h)^2 - x^2)/h as h goes to 0 is 2*x. Thus, a non-standard analysis proof about numbers being infinitesimally different establishes a standard fact about behavior of an expression as a certain quantity gets smaller and smaller.

The same relation holds between non-standard and standard execution. Our proof that if x is non-negative and finite then SQRT(x) == root(x) actually establishes that if we run SQRT on a sequence of machines whose CR is more and more precise, the output of SQRT(x) will converge to root(x). More generally, if we have any real-valued ideal function f and a program F and we can prove in the non-standard formalism that for all finite x in the domain of f, F(x) == f(x) then this will establish that if we run F on a sequence of more and more precise machines, the output of F(x) will converge to f(x). To put it another way, we can obtain any degree of precision in F(x) by computing F(x) on a sufficiently precise machine.