An Application of a Theorem Prover

Abstract

This work uses the specification of the type interval in OBJ3 to prove some properties of the interval probability. OBJ3 is a functional language and also includes mechanisms for theorem proving.

1 Introduction

One of the most important claims of the functional programming paradigm is program provability. Due to the simple syntax ans well defined semantics programs written in functional languages are easier to prove correct than its imperative counterparts. On the other hand, ordinary computer arithmetic, or floating-point arithmetic brings within unreliability on arithmetic representation and operations.

Functional languages have for long worried about this aspect and has a tradition of proposing interesting solutions such as LISP rationals and Lazy reals [CCCL 95, CCL 95]. We advocate the natural merge between functional languages and interval arithmetic [LIN 95]. One of the most important features of interval arithmetic in relation to its predecessors is efficiency.

The introduction of the type interval into a programming language, has always had the main goal of solving algorithms with verified validation. This work uses OBJ3 [GOG 93] to formalize the type interval through some of its algebraic and topological properties. OBJ3 is a functional language and also includes mechanisms for theorem proving. The contribution of this article consists in using an algebraic language to formally specify the type interval and to perform mechanical reasoning, and in applying that specification to evaluate interval probabilities and to prove automatically propositions related to the interval probability.

Interval mathematics [MOO 66, MOO 79] is an alternative for solving numerical problems related to lack of accuracy in data, roundoff errors, truncation errors and propagation of errors in a sequence of arithmetic operations. Probability theory [BUR 72, CHU 74] and interval mathematics are classic approaches for representing imperfection in the information. The first is related to random problems, the second to numerical problems.

As an example, considerer the representation of 1/3 with four significant digits. The rouding for the nearest is 0.3333, but this value can be substituded by the interval [0.3333,0.3334], using the directed roundings [KM 81]. This interval seems more inacurate than the real value but it is more reliable because it shows the present degree of incertainty. It also shows that 0.3333 is an underestimation of the actual value. The interval representation provides information about the computed value what a single number can not do.

2 An Overview of OBJ3

OBJ3 is the latest in a series of OBJ systems, all based upon first order equational logic. A detailed description of OBJ3 can be found in [GOG 93]. In this section it is given a brief overview of the system (based on Release 2).

OBJ3 is a general-purpose declarative language, especially useful for specification and prototyping. A specification in OBJ3 is a collection of modules of two kinds: *theories* and *objects*. A theory has *loose* semantics, in the sense that it defines a variety of models. An object has *tight* or *standard* semantics; it defines, up to isomorphism, a specific model—its initial algebra. For example, the usual specification of the natural numbers as an object would describe precisely what it is understood by the natural numbers. As a theory, it would describe any model that satisfies the required properties; for instance, integers would be a model. One of the consequences of the distinction between a theory and an object is that it is valid to use induction for an object, but not for a theory.

A module (an object or a theory) is the unit of a specification. It comprises a signature and a set of (possibly conditional) equations—the axioms. The equations are regarded as rewrite rules and computation is accomplished by term rewriting, in the usual way.

An elaborate notation for defining signatures is provided. As OBJ3 is based upon *order sorted algebra* it provides a notion of subsorts which is extremely convenient in practice. For example, by declaring `Integer` as a subsort of `Float` no conversion function is needed to turn an integer into a float. Sorts and subsorts are declared respectively by `sorts` $\langle SortIdList \rangle$. or `sort` $\langle SortIdList \rangle$., `subsort` $\langle Sort \rangle < \langle Sort \rangle$ or `subsorts` $\langle SortList \rangle < \langle SortList \rangle$ ....

A general *mixfix* syntax can be used to define operators; users can define any syntax including prefix, posfix and infix. The argument and value sorts of an operator are declared when its syntactic form is declared. The general syntax for an operator is `op` $\langle OpForm \rangle$: $\langle SortList \rangle$ `->` $\langle Sort \rangle$.; in this paper, it is used names for operators which coincide with the LaTeXrepresentation of the desired mathematical symbols. This makes the encoding in OBJ3 easier to understand.

Moreover, operators may have attributes describing useful properties such as associativity, commutativity and identity. This makes possible to document the main properties of a given operator at the declaration level. As a consequence, the number of equations that need to be input by the user is reduced considerably in some cases. Most importantly, OBJ3 provides rewriting modulo these attributes.

Modules may be parameterised by theories which define the structure and properties required of an actual parameter for meaningful instantiation. The instantiation of a generic module requires a *view*—a mapping from the entities in the requirement theory to the entities in the actual parameter (module). As a simple example, an object to describe sets of arbitrary elements, say `SET`, should be parameterised by a theory which requires the argument module to have (at least) one sort. Then this module can instantiated to obtain sets of elements of the desired sort.

Apart from the mechanisms for defining generic modules and instantiating them, OBJ3 provides means for modules to import other modules and for combining modules. For example, `A + B` creates a new module which combines the signature and the equations of `A` and `B`.

OBJ3 can also be used as a theorem prover. In particular, computation is accom-

plished by term rewriting which is a widely accepted method of deduction. If the rewrite rules of the application theory are *confluent* (or *Church-Rosser*) and *terminating*, the exhaustive application of the rules works as a decision procedure for the theory: the equivalence of two expressions can always be determined by reducing both to normal form (an expression that can be reduced no longer); the equivalence holds if the two normal forms are (syntactically) the same. This proof mode is normally called *automatic*, and is supported by OBJ3. The strategy for theorem proving consists in showing that all reductions in the proof score (code for a proof) evaluate to `true`.

Unfortunately, applications in general do not enjoy the properties above. As a consequence, automatic term rewriting may fail to reduce two equivalent expressions to the same normal form, or the process may even fail to terminate. Therefore there is the need for a mechanism for applying rules in a controlled way. OBJ3 supports the *step-by-step* application of rewrite rules either forward (from left to right) or backward (from right to left).

According to [GOG 93] OBJ3 an E-strategy is a sequence of integers in parentheses given as an operator attribute following the keyword `strat`; the following example from [GOG 93]

```
op _+_: Int Int -> Int [strat ( 1 2 0 )] .
```

indicates that `_+_` on `Int` has strategy (1 2 0), i.e., this means that the system evaluates both arguments before to add them. OBJ3 does lazy evaluation in two ways. First, omitting a given argument number from the strategy; secondly, giving a negative number in a strategy to indicate that the j^{th} argument is to be evaluated only on demand.

3 The type interval

An interval of reals, now on called an *interval* for short, is a new kind of number [MOO 66, MOO 79]. The set of the intervals is called $\mathbb{IR}$. As can be seen below, for this set are defined arithmetic operations, width, distance and absolute value; additionally, an interval is also a set and then operations defined over sets are also valid between intervals. Beyond these features, intervals carry out roundoff errors when used in numerical computations. In computers it is possible to compute intervals containing exact real arithmetic results by rounding the endpoints of the intervals computed by the machine [KM 81]. Applications of intervals can be used in several branches of the science. The set $\mathbb{IR}$ may be furnished with the arithmetic operations $+$, $-$, $\cdot$ and $/$ defined as follows (see [MOO 66, MOO 79, AH 83] for more details).

An *interval* is the set

$$[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}.$$

Intervals are represented by capital letter. So $X = [2.3, 4.2] \in \mathbb{IR}$.

Let $X = [x_1, x_2]$, $Y = [y_1, y_2]$ e $Z = [z_1, z_2]$ be elements of the set $\mathbb{IR}$.

X=Y if and only if $x_1 = y_1$ and $x_2 = y_2$. This relation is reflexive, antisymmetric and transitive.

The *negative* of the interval X is:

$$-X = -[x_1, x_2] = [-x_2, -x_1] = \{x \in \mathbb{R} \mid -x_2 \leq x \leq -x_1\}.$$

and the *reciprocal*:

$$1/X = \{1/x \mid x \in \mathbb{R}, 0 \notin X\}.$$

The arithmetic operations over $\mathbb{IR}$ are defined by:

$$X + Y = [x_1 + y_1, x_2 + y_2]$$

$$X - Y = X + (-Y) = [x_1 - y_2, x_2 - y_1]$$

$$X * Y = [min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}]$$

$$X/Y = X * (1/Y).$$

¿From the arithmetic operation definitions [MOO 66, MOO 79, AH 83] have proved the following algebraic properties of the interval arithmetic (Theorem 3.1). The distributive law does not always hold contrariwise to the subdistributivity law.

Theorem 3.1

(1) $X + Y = Y + X$. (commutativity of addition)

(2) $X + (Y + Z) = (X + Y) + Z$. (associativity of addition)

(3) $X + \mathbb{0} = X$, where $\mathbb{0}$ =[0,0]. (neutral ellement of addition)

(4) $0 \in (X - X)$.

(5) $X * Y = Y * X$. (commutativity of multiplication)

(6) $X * (Y * Z) = (X * Y) * Z$. (associativity of multiplication)

(7) $X * \mathbb{I} = X$, where $\mathbb{I}$ =[1,1]. (neutral ellement of multiplication)

(8) $1 \in X/X,\ 0 \notin X$.

(9) $X * (Y + Z) \subseteq (X * Y) + (X * Z)$. (subdistributivity) ■

The *intersection* between intervals X and Y is defined by:

$$X \cap Y = \begin{cases} \emptyset & \text{,if } x_1 > y_2 \text{ or } x_2 < y_1 \\ [max(x_1, y_1), min(x_2, y_2)] & \text{,otherwise.} \end{cases}$$

As an interval is also a set, specifically a set of real numbers, Theorems 3.2 and 3.3 show some properties of intervals when they are considered sets. Intervals as sets are important for defining nested sequences, more efficient refinements, etc.

Theorem 3.2

(10) $X \cap Y = Y \cap X$. (commutativity of interval intersection)

(11) $X \cap (Y \cap Z) = (X \cap Y) \cap Z$. (associativity of interval intersection) ■

and the *union*:

$$X \cup Y = [min(x_1, y_1), max(x_2, y_2)]$$

Theorem 3.3

(12) $X \cup Y = Y \cup X$. (commutativity of interval union)

(13) $X \cup (Y \cup Z) = (X \cup Y) \cup Z$. (associativity of interval union) ■

Moore [MOO 66, MOO 79] proposes the following order:

$$X < Y \text{ if and only if } x_2 < y_1.$$

Theorem 3.4 The order above is transitive.
Kulisch e Miranker [KM 81] propose the relation $\leq$:

$$X \leq Y \text{ if and only if } x_1 \leq y_1 \text{ and } x_2 \leq y_2.$$

Moore [MOO 66, MOO 79] defines the inclusion order:

$$X \subseteq Y \text{ if and only if } y_1 \leq x_1 \text{ and } x_2 \leq y_2.$$

The *width* of the interval X is defined by $w(X) = x_2 - x_1$.

Theorem 3.5 shows some properties of the width of intervals.

Theorem 3.5

(14) $X \subseteq Y \Rightarrow w(X) \leq w(Y)$.

(15) $w(X + Y) = w(X) + w(Y)$.

(16) $w(X - Y) = w(X) + w(Y)$. ■

Let d be such that

$$d(X, Y) = max\{\mid x_1 - y_1 \mid, \mid x_2 - y_2 \mid\}.$$

Theorem 3.6 d is a *distance* in $\mathbb{IR}$:

(17) $d(X, Y) = 0$ if and only if $X = Y$.

(18) $d(X, Y) = d(Y, X)$. (simmetry)

(19) $d(X, Z) \leq d(X, Y) + d(Y, Z)$. (triangle inequality) ■

Let $\mid\ \mid$:$\mathbb{IR} \rightarrow \mathbb{R}$be defined by:

$$\mid X \mid = max(\mid x_1 \mid, \mid x_2 \mid).$$

The last theorem is concerned with properties of the absolute values of an interval.

Theorem 3.7 The *absolute value* of the interval X has the following properties:

(20) $| X | \geq 0$.

(21) $| X | = 0$ if and only if X=[0,0].

(22) $| X + Y | \leq | X | + | Y |$. (triangle inequality)

(23) $| X * Y | = | X | * | Y |$.

(24) $X \subseteq Y \Rightarrow | X | \leq | Y |$. ■

4 Specifying the Type Interval using OBJ3

Intervals can be seen like a number, for example [2,9]. But intervals are also sets, because $[2,9] = \{x \in \mathbb{R} \mid 2 \leq x \leq 9\}$. Thus the theory `INTERVAL` must include the boolean (`BOOL`) and floating-point (`FLOAT`) values with its respective operators. OBJ3 has modules which implement this two algebras. Furthermore, given the features of the interval operations, it is necessary to use floating-point sets. This may be done by specifying the theory of sets as a generic module (parameterized by the type of the elements). Then this generic module may be instantiated with the module `FLOAT`. Beyond the type of the elements, it is necessary a total order defined between the elements, because the interval theory uses properties of sets, such as the maximum (`max`) and the minimum (`min`) of a set.

`OT` is a theory which describes a total order with an infix operator `_<=_`. This operator is reflexive, antisymmetric and transitive. Beyond these properties, all the elements in this theory are comparable. `OT` parameterizes the module `SET` and then it is possible a polymorphic definition of the minimum (`min`) and the maximum (`max`) of a set. The theories of the interfaces of the parameterized modules must be defined before this module. In this case, `OT` must be defined before `SET`.

The object `SET` has, amongt others, an infix operator `U` (union) which has attributes that specifies commutativity (`comm`), associativity (`assoc`), idempotence (`idem`) and has an identity element (`id`) which is the empty set ({}). As previously described, in OBJ3 this may be defined at the level of the operator declaration, then explicit equations are not necessary to state those propertyes. The `min` and the `max` operators are partially described: the empty set is not considered. This does not present problems in this particular application. Several other operators on sets, such as intersection and difference were left out in this application.

The object `SET-FLOAT` is composed of properties of the arithmetic operations involving real numbers, here `Float`. This object is not a primitive abstract data type of the language. It was defined in order to allow as to prove properties of interval numbers in the theory `INTERVAL`.

The theory `INTERVAL` contains a description of the type (*sort*) `Interval` as well as its unary and binary operators. The module `BOOL` is an instantiation of the module `SET` where the floating-point numbers are imported using the clause `protecting`. This kind of importation is used to make it explicit that `INTERVAL` neither adds elements to the imported algebras nor identifies elements in this algebra. The relation *subsort* declares that intervals may be represented as sets. This facility of OBJ3 is particulary useful in the definition of intersection and union of intervals when the result of some operation may be the empty set. The constructor of the type interval (`[_,_]`) is a partial operator:

the lower bound of the interval must be less than or equal to the upper bound. The symbols [prec 2], [prec 4] [prec 5] [prec 6] and [prec 9] are the precedences of the operators reciprocal, negation, multiplication, subtraction and equality. This means, for example, that the precedence of the negation is lower than the precedence of the multiplication in terms that involve both (that is, negation binds tighter than multiplication). The additional properties of the operators are described by equations (eq) and conditional equations (cq). It is possible to associate a name (label) with the equations. The name may then be used to refer to the respective equation during theorem proving.

```
th INTERVAL is

 protecting BOOL  .
 protecting FLOAT .
 protecting SET-FLOAT .

 sort Interval .

 subsort Interval < Set .

 op-as [_,_]   : Float Float -> Interval  for [x1,x2] if  x1 <= x2 .
 op _+_        : Interval Interval -> Interval [prec 6] [comm assoc id: [0,0]] .
 op -_         : Interval -> Interval [prec 4] .
 op _-_        : Interval Interval -> Interval [prec 6].
 op _*_        : Interval Interval -> Interval [prec 5] [comm assoc id: [1,1]] .
 op _ -1       : Interval -> Interval [prec 2] .
 op _/_        : Interval Interval -> Intervalo [prec 5].
 op w_         : Interval -> Float .
 op absi_      : Interval -> Float .
 op d          : Interval Interval -> Float .
 op _om_       : Interval Interval -> Bool .
 op _ok_       : Interval Interval -> Bool .
 op _inc_      : Interval Interval -> Bool .
 op _/\_       : Interval Interval -> Interval [comm assoc]
 op _\/_       : Interval Interval -> Interval [comm assoc] .
 op _pertains_ : Float Interval -> Bool .

 var A B         : Interval .
 var x1 x2 x3 x4 : Float .

 [add]  eq [x1,x2] + [x3,x4] = [x1 + x3, x2 + x4] .
 [sym]  eq - [x1,x2] = [- x2, - x1 ] .
 [dif]  eq A - B = A + - B .
 [eq]   eq [x1, x2] == [x3, x4] = (x1 == x3) and (x2 == x4) .
 [mult] eq [x1,x2] * [x3,x4] = [min({x1 * x3} U {x1 * x4} U {x2 * x3}
                                                      U {x2 * x4}),
                                max({x1 * x3} U {x1 * x4} U {x2 * x3}
                                                      U {x2 * x4})] .
 [rec]  eq [x1,x2] -1 = [1 / x2, 1 / x1] .
```

```
[div]  eq A / B = A  * B -1 .
[wid]  eq w [x1,x2] = x2 - x1 .
[abs]  eq absi [x1,x2] = max({abs(x1)} U {abs(x2)}) .
[dist] eq d([x1,x2] , [x3,x4]) = max({abs(x1 - x3)} U {abs(x2 - x4)}) .
[om]   cq [x1,x2] om [x3,x4] = true if x2 < x3 .
[okm]  cq [x1,x2] ok [x3,x4] = true if x1 <= x3 and  x2 <= x4 .
[inc]  eq [x1,x2] inc [x3,x4] = if x3 <= x1 and  x2 <= x4
                                   then true
                                   else false
                                fi .
[int]  eq [x1,x2] /\ [x3,x4] = if x1 > x4 or x2 < x3
                                   then {}
                                   else [max ({x1} U {x3}),
                                         min ({x2}  U{x4})]
                               fi .
[uni]  eq [x1,x2] \/ [x3,x4] = if ([x1,x2] /\ [x3,x4]) == {}
                                   then {}
                                   else [min ({x1} U {x3}),
                                         max ({x2} U{x4})]
                               fi .
[pert] cq x1 pertains [x2,x3] = true if x2 <= x1 and x1 <= x3 .

endth
```

Example 4.1 This example evaluate the negative of the interval [-4,-2] in the theory INTERVAL:

```
reduce - [-4,-2] .
```

The system returns the following:

```
==========================================
reduce in INTERVAL : - [- 4.0,- 2.0]
rewrites: 3
result Interval: [2.0,4.0]
==========================================
```

The command `reduce` or (`red`) is executed by the application of rewrite rules. In this example the system performs 5 rewriting steps. The result is the interval [2.0,4.0].

Example 4.2 Given the intervals X=[1,2], Y=[3,4], Z=[-3.5,0] and W=[-3,0], evaluate the expression

$$(\mathrm{X}^{-1} * (-\mathrm{Y})) \cap \mathrm{Z})$$

and compare the resulting interval with W, using the order proposed in [KM 81]. The command is

```
red (( [1,2] -1 * -[3,4]) /\ [-3.5,0]) ok [-3,0] .
```

The system returns:

```
==========================================
reduce in INTERVAL : ([1.0,2.0] -1 * - [3.0,4.0]) /\ [-3.5,0.0] ok
    [-3.0,0.0]
rewrites: 63
result Bool: true
==========================================
```

4.1 Proving Properties of the Interval Probability Using OBJ3

Probability theory [BUR 72][CHU 74], as we know today, was axiomatized by Kolmogorov [KOL 50] after the works of Lebesgue [RUD 76]. The formal background of probability rests on set theory and makes use of relations and functions of the real variable.

Computing probabilities in practice envolves floating-point numbers [GOL 91] and, in consequence, numerical problems. The discrete representation of the real numbers (a continous set) in a finite number of machine words gives birth most of the problems in scientific computation. The floating-point numbers as an algebraic system is extremely poor, if compared to the real numbers. Thus, calculating probabilities suffers from the same problems one faces in scientific computing.

The interval probability was defined by Campos [MAC 97]. This one is a useful innovation for validating probabilities since it contributes both to enhance its theoretical foundations as well as to facitate its applications in controlling numerical errors. Now the specification is applied to prove one of its properties. Let $P_{\mathbb{IR}}(\cdot)$ be the interval probability.

Proposition 4.1 If $P_{\mathbb{IR}}(\cup_{k=1}^{n} A_k) = \sum_{k=1}^{n} P_{\mathbb{IR}}(A_n)$, then

$$w(P_{\mathbb{IR}}(\cup_{k=1}^{n} A_k)) = \sum_{k=1}^{n} w(P_{\mathbb{IR}}(A_k)).$$

Proof. The proposition is proved by induction. For $n = 2$ one must proves that $w(X + Y) = w(X) + w(Y)$, where w is the width of the interval. Then

```
open .
ops 'x1 'x2 'x3 'x4 : -> Float .
eq 'x1 <= 'x2 = true .
eq 'x2 <= 'x1 = false .
eq 'x3 <= 'x4 = true .
eq 'x4 <= 'x3 = false .
red w(['x1, 'x2] + ['x3, 'x4]) == w ['x1, 'x2] + w ['x3, 'x4] .
close

OBJ>
in Proof
==========================================
open
==========================================
```

```
ops 'x1 'x2 'x3 'x4 : -> Float .
==========================================
reduce in INTERVAL0 : w ['x1,'x2] + ['x3,'x4] == w ['x1,'x2] + w ['x3,
    'x4]
rewrites: 15
result Bool: true
==========================================
close
OBJ> ev (dribble)
```

Now let $w(P(\cup_{k=1}^{n-1})) = \sum_{k=1}^{n-1} w(P(A_k))$ be. Then

$$\begin{aligned} w(P_{\mathbb{IR}}(\cup_{k=1}^{n})) &= w(P_{\mathbb{IR}}(\cup_{k=1}^{n-1})) + w(P_{\mathbb{IR}}(A_n)) \\ &= \sum_{k=1}^{n-1} w(P_{\mathbb{IR}}(A_k)) + w(P_{\mathbb{IR}}(A_n)) \\ &= \sum_{k=1}^{n} w(P_{\mathbb{IR}}(A_k)). \end{aligned}$$

■

5 Conclusions

OBJ3 was used with success as a theorem prover to verify properties of the interval arithmetic. Furthermore, given the natural dificulties of the interval methods and the high accuracy arithmetic this is an exciting and fruitful research area.

The dual nature of the type interval, was easily solved with the *subsort* declaration, i.e., `subsort Interval < Set` means that the set of elements having the first *sort* (`Interval`) is a subset, not necessarily proper, of the set of elements having the second *sort* (`Set`).

The module system stimulates an incremental presentation of the theory. It was also illustrated that modules may be parameterized by theories and instantiated to attend particular necessities. Such facility is more powerful than the ones offered by most of the specification/programming languages, where only the type of the arguments is considered; OBJ3 also specifies the properties that the real parameters must satisfy.

References

[AH 83] ALEFELD, G.; HERZBERGER, J. **Introduction to Interval Computation**. Academic Press, N. Y., 1983.

[BUR 72] BURRILL, C. W. **Measure, Integration and Probability**. McGraw-Hill Book Company, 1972.

[MAC 97] CAMPOS, M. A. **An Interval Extension to Probabilities** (Uma Extensão Intervalar para a Probabilidade Real). Doctoral Thesis, Universidade Federal de Pernambuco, Brazil, 1997. (in Portuguese)

[CHU 74] CHUNG, K. L. **A Course in Probability Theory**. Academic Press, Inc., second edition, 1974.

[CCL 95] CORREIA, M. de B., CAMPOS, M. A. and LINS, R. D. Paralelismo e Confiabilidade da Aritmética Computacional **XVI Congresso Ibero Latino Americano sobre Métodos Computacionais em Engenharia**, p. 1375-1385, Nov. 1995.

[CCCL 95] CLAUDIO, D. M., CORREIA, M. de B., CAMPOS, M. A. and LINS, R. D. **Linguagens de Programação e Computação Científica**. Jornada de Atualização em Informática, Ago. 1995.

[GOG 93] GOGUEN, J. A., WINKLER, T., MESEGUER, J., FUTATSIGI, K. and JOUANNAUD, J. P. Introduction to OBJ, SRI International, 1993.

[GOL 91] GOLGBERG, D. What Every Computer Scientist Should Know About Floating-Point Arithmetic. **ACM Computing Surveys**, v. 23, n. 1, p. 5-48, Mar. 1991.

[KOL 50] KOLMOGOROV, A. N. **Foundations of the Theory of Probability**. Chelsea, N. York, 1950.

[KM 81] KULISCH, U. W.; MIRANKER, W. L. **Computer Arithmetic in Theory and Practice**. Academic Press, 1981.

[LIN 95] LINS, R. D., CAMPOS, M. A. and CORREIA, M. de B. Functional Programming and Interval Arithmetic **International Journal of Reliable Computing**, p. 146-150, Feb. 1995.

[MOO 66] MOORE, R. E. **Interval Analysis**. Prentice Hall Inc., Englewood Clifs, N. J., 1966.

[MOO 79] MOORE, R. E. **Methods and Applications of Interval Analysis**. SIAM, Philadelphia, Pennsylvania, 1979.

[RUD 76] RUDIN, W. **Principles of Mathematical Analysis**. McGraw-Hill International Editions, 1976.