Interval Computations
No 3(5), 1992

ON INTERVAL EXTENSIONS OF COMPUTER ALGEBRA SYSTEMS

Nikolay M.Glazunov

In this paper a number of interval extensions of computer arithmetics and computer algebra systems are considered. A short description of peculiarities of implementations and applications are given.

ОБ ИНТЕРВАЛЬНЫХ РАСШИРЕНИЯХ СИСТЕМ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

Н.М.Глазунов

Рассмотрен ряд интервальных расширений компьютерных арифметик и систем компьютерной алгебры (СКА), кратко описаны особенности некоторых реализаций и указаны приложения.

Introduction

We call the interval extension of computer algebra system (CAS) an extension to interval data of definition domains and ranges of expression which can be represented in CAS as well as of operations over these expressions.

The implementation of interval arithmetics or, more generally, of methods of interval computations [1–5] in computer algebra systems allows to combine computer-algebraic transformations with the strict numerical results obtained during or on completion of the process of computer-algebraic transformations. At present a great number of interval arithmetics and interval computational schemes are proposed. They represent both the features of implementations of interval computations on a computer and the features of the object fields, for which interval computations are carried on. In turn, the known CAS are developed and the new ones appear (see, for example, [6] and bibliography in it), that causes the development and change of the architecture of CAS. The sphere of applications of interval and of computer-algebraic methods is continually extended.

In the paper a short review of interval arithmetics (IA) and of methods of interval computations is given, and IA used by the author for the extension of CAS (various versions of the Reduce system) are described in more detail, the description of the CAS architecture is adduced. The architecture determines to a large extent directions in implementation of interval extensions. Possible ways of integration of CAS and systems of interval computations are also described and some applications are shown.

Interval arithmetics and schemes of interval computations

A great number of interval arithmetics is proposed that reflects various aspects, theoretical and applied, of interval computations and defines various algebraic structures of interval data domain. These are the arithmetics of Moore, Kahan, Markov, Olwer, generalized arithmetic of Hansen, a posteriori interval analysis of Matiyasevich, and many others [1–5]. A great number of computer implementations for these arithmetics and methods is proposed. The survey and the analysis of these interesting and current works of different authors require a separate consideration.

The following modifications and extensions of Moore's interval arithmetics are implemented by the author: fixed-point interval arithmetic, floating-point interval arithmetic with account of roundoff error either in the arrest digit n of declared precision ($n \leq 24$) or in the least significant bit of binary notation (implementation on the IBM 360/370 — compatible computers in Turbo-C), interval arithmetic for a vector-pipeline computer in ASSEMBLER, rational arithmetic of infinite precision for various modifications of the Reduce system.

Computer arithmetics

Interval arithmetics can be constructed on the basis of computer arithmetics. The latter in turn are subdivided into standard computer arithmetics (s.c.a.) and arithmetics realizable on computers, but not standard. We understand the standard computer arithmetics as integral or fixed-point arithmetics (that are not considered below) and floating-point computer arithmetics satisfying, in particular, the JEEE-754 standard. We refer to remaining computer arithmetics as nonstandard (see also [7]).

Let $\mathbb{F}(\beta, r, e_1, e_2) = \{x \in \mathbb{R} \mid x = z.d_1 \ldots d_r \cdot \beta^e\}$, $z \in \{+1, -1, 0\}$, $d_i \in \{0, 1, \ldots, \beta - 1\}$, $i = 1, \ldots, r$; $d_1 \neq 0$, $e_1, e_2, e \in \mathbb{Z}$, $e_1 \leq e \leq e_2$ be the model of a floating-point arithmetic, with β being the base. In the implementation of nonstandard floating point arithmetics (see below) the point before d_1 can be moved to the end of the mantissa $d_1 \ldots d_r$, and the mantissa and exponent themselves will be the numbers of indefinite precision. In these cases sometimes the notation $\mathbb{F}(\beta, e)$ or simply $\mathbb{F}(\beta)$ is used.

For 80286 processor with 80287 coprocessor the numbers in the floating-point computer arithmetic have the following formats:

$$(-1)^{sgn}(1.d_1 \ldots d_{23}) \cdot 2^{E-127} \qquad \text{(single precision)}$$
$$(-1)^{sgn}(1.d_1 \ldots d_{52}) \cdot 2^{E-1023} \qquad \text{(double precision)}$$
$$(-1)^{sgn}(1.d_1 \ldots d_{64}) \cdot 2^{E-16383} \qquad \text{(extended precision)}$$

Here sgn is the value of sign bit, $d_1, d_2, \ldots$ are the binary digits of mantissa field, and E is the value of order field.

In arithmetic operations, the following types of roundings are supported: the rounding towards zero (truncation), the rounding to a nearest computer number, the rounding to $+\infty$, the rounding to $-\infty$.

For computers of Crey type or Elektronika SS-Bis the floating-point numbers are represented in the form

$$(-1)^{sgn} \cdot m \cdot 2^e,$$

where $1/2 < m < 1$ for normalized numbers, and $-8192 \leq e \leq 8192$. Rounding towards zero and rounding to a nearest computer number are supported by the program.

In the Reduce 3.1 system and in the later versions the arbitrary precision real arithmetic and the arbitrary precision rational arithmetic are realized on the basis of Sasaki's package. Thus, the arbitrary precision real number is represented as the element of $\mathbb{F}(10)$ in the form of the following LISP S-statement: (! : *BF*! :.(m.e.)) (for the sake of visualability here and below we use the simplified representation of the S-statement without QUOTE-function). Here ! : *BF*! : is an index, which distinguishes the big-float data type from another ones. A rational number is represented by the S-statement (! : *RATNUM*! : .($n.d$)) where ! : *RATNUM*! : is the index to a rational number and n, d are the numerator and the denominator respectively.

Computer interval arithmetics

For each type of rounding in the $\mathbb{F}(\beta, r, e_1, e_2)$ arithmetic and in it's modifications one can define a machine interval arithmetic with the operations of addition, subtraction, multiplication and division, in which this type (or combination of types) of rounding is realized. For example, if the truncation of the results of arithmetic operations is implemented on a processor, then let us introduce the following notation:

$(\underline{a}, \overline{a})$	is an interval with $\underline{a}$ and $\overline{a}$ bounds;
corr	is the addition or subtraction, depending on sign before it, of the unit to the less significant digit of mantissa of the result;
&	is the conjunction symbol,
ϵ	is the smallest number representable on computer, e.g. for the Elektronika SS-BiS $\epsilon = 0.5 \cdot 10^{-8192}$.

The algorithm for a multiplication, MLI, with account of truncation of the results of arithmetic operations has the following representation in Algol-like language:

```
if $\underline{x} \geq 0$ then do;
  if $\underline{y} \geq 0$ then $\underline{z} = \underline{x} * \underline{y}$; else $\underline{z} = \overline{x} \times \underline{y} - corr$;
       if $\overline{y} \geq 0$ then $\overline{z} = \overline{x} * \overline{y} + corr$; else $\overline{z} = \underline{x} * \overline{y}$;
            end;
  else if $(\underline{x} < 0)\&(\overline{x} \geq 0)$ then do;
   if $\underline{y} \geq 0$ then do; $\underline{z} = \underline{x} * \overline{y} - corr; \overline{z} = \overline{x} * \overline{y} + corr$; end;
     else if $\overline{y} < 0$ then do; $\underline{z} = \overline{x} * \underline{y} - corr; \overline{z} = \underline{x} * \underline{y} + corr$; end;
          else do; /*   $\underline{y} < 0 < \overline{y}$   */
     if $\underline{x} * \overline{y} \geq \overline{x} * \underline{y}$ then $\underline{z} = \overline{x} * \underline{y} - corr$; else $\underline{z} = \underline{x} * \overline{y} - corr$;
     if $\underline{x} * \underline{y} \geq \overline{x} * \overline{y}$ then $\overline{z} = \underline{x} * \underline{y} + corr$; else $\overline{z} = \overline{x} * \overline{y} + corr$;
           end;             end;
     else do;      /*   $\overline{x} < 0$   */
```

if $\overline{y} \geq 0$ **then** $\underline{z} = \underline{x} * \overline{y} - corr$; **else** $\underline{z} = \overline{x} * \overline{y}$;
if $\underline{y} \geq 0$ **then** $\overline{z} = \overline{x} * \underline{y}$; **else** $\overline{z} = \underline{x} * \underline{y} + corr$; **end**;

For the schemes of interval operations see, for example, [9].

For Reduce 3.3 system if the big-float data type is chosen then the interval number may be represented in the form of the S-statement

$$(!:INTNA!:.(!:BF!:.((\mathrm{m.e.}))(LB.RB))$$

where $!:INTNA!:$ indicates the type, and left LB and right RB ends of the interval are of big-float type.

In [10] the program package of interval arithmetic for the big-float numbers in the Reduce system is described. The author learned about this work after his own package in the Reduce 3.3 had already been implemented. Though this package possesses less possibilities for interval big-float numbers than the one described in [10], it allows, on the other hand, to process the interval rational numbers, which require no account of the roundoff error in the computational process, except, may be, for the last step, when the resulted interval rational number $(\underline{r}, \overline{r})$ is transformed to the decimal floating-point notation.[1]

Interval computations may be organized also in the Reduce 3.3 language without using of character mode, if we represent the interval rational number as a list: $JR := \{LB, RB\}$ and define the interval arithmetic operations on such represented rational numbers. E.g. if we rename $X1 = \underline{x}$, $X2 = \overline{x}$, $Y1 = \underline{y}$, $Y2 = \overline{y}$, $Z1 = \underline{z}$, $Z2 = \overline{z}$ removing the operation of correction $corr$ we can define the begining of $MLIRL$ procedure of multiplication of intervals in the list representation by the statements:

$$PROCEDURE\ MLIRL(L1, L2, R);$$
$$BEGIN\ SCALAR\ X1, X2, Y1, Y2, Z1, Z2;$$
$$X1 := FIRST(L1);\ X2 := SECOND(L1);$$
$$Y1 := FIRST(L2);\ Y2 := SECOND(L2);$$

Further, replacing the delimiters of compound operator (**do**; and **end**) by the double opening << and closing >> brackets and taking into account the syntax of the conditional statement in Reduce 3.3 we insert after these replacements the text of MLI into the $MLIRL$ procedure.

[1]As J. Wolff von Gudenberg reported to author the subroutine library of interval computations on the Reduce 3.X has been worked out also by J.Fitch.

The *INAR* library that contains the *MLIRL* modul also includes the moduls: *ADIRL*, *SBIRL*, *DVIRL*, *DGINR*, and some others.

It must be simplest, but not efficient implementation of interval arithmetic in Reduce 3.3.

Architecture and interval extensions of computer algebra systems

The directions of interval extension of CAS are defined to a large extent by their architecture. By analogy with the classification of database systems we will distinguish the CAS of closed type and the CAS with base programming language referring the latter CAS as the CAS of open type. Under the CAS of closed type we understand the systems that have the fixed set of acceptable data types (polynomials, rational functions, vectors, matrices and others) and the fixed set of statements for computations and for algebraic transformations with these data type, access to which is realized by a statement identifier or from a menu. The Derive system is an example of the system of closed type. Under the systems of open type we understand the systems possessing one or more programming languages that allow the extensions on the level of language facilities and are accessible for the user of the system. The Reduce 3.3 system is an example of the system of open type. It is implemented in R-LISP and provides the 3 interrelated program environments for the user: Reduce 3.3 environment, R-LISP environment and S-LISP environment.

The user is provided by the opportunity to work in two modes – character mode and algebraic mode, and by the opportunity to change the mode and to connect the modes.

Integration of computer algebra and of systems of interval computations

If the user is not hampered in memory resourses (e.g. on the IBM PC), has access to CAS and to the system of scientific computations, and can manage with these systems, then the following organization of computer algebraic and interval computations is possible with the use, e.g. of the Derive and Pascal-XSC[2] systems.

[2] The author is grateful to A.Davidenkoff, who has sent the information on a number

Notice, that the Pascal-XSC system [11] uses the Borland C++ and together they occupy about 16 Mbyte of hard disk memory. Computer-arithmetic transformations have been carried out in the Derive system and the obtained result has been transformed to the Pascal-XSC file (Derive has such opportunity). We edit this file according to the syntax of Pascal-XSC, define the interval value of constants and variables of the expression being computed, and transfer the formed program for execution to the Pascal-XSC system.

Applications

The implementation of interval-analytical computations on the CAS can be used for obtaining guaranteed results in the field of theoretical mathematics, for research (analysis and synthesis) of robust dynamic systems, for research of ill-conditioned problems and also for investigation of a range of applied problems whose parameters and coefficients change in the intervals.

References

1. Moore, R.E. *Methods and applications of interval analysis.* SIAM, Philadelphia, 1979.
2. Kalmykov, S.A., Shokin, Yu.I. and Yuldashev, Z.Kh. *Methods of interval analysis.* Nauka, Novosibirsk, 1986 (in Russian).
3. Alefeld, G. and Herzberger, J. *Introduction to interval computations.* Academic Press, New York, 1983.
4. Kulisch, U. and Stetter, H. (eds.) *Scientific computation with automatic result verification.* Springer-Verlag, 1988.
5. Matiyasevich, Yu.I. *Real numbers and computers.* Kibernetika i vychisl. tekhnika, **2** (1989) pp. 104–133 (in Russian).
6. Wall, S.M. (ed.) ISSAC'91. ACM Press, New York, 1991.
7. Glazunov, N. M. *Numerical-analytical computations and nonstandard arithmetics.* In: "Proc. International Workshop on Analytical Computations on Computers and Their Applications in Theoretical Physics, Dubna, Sept. 17–18, 1985", Joint Institute for Nuclear Research, Dubna, 1985, pp. 143–148 (in Russian).

of XSC-systems, and to D.Shiriaev, who has given to the author the demo-version of the Pascal-XSC system.

8. Sasaki, T. *An arbitrary precision real arithmetic package in Reduce.* Lecture Notes in Computer Sci. **72** (1979), pp. 358–368.

9. Glazunov, N. M. *An interval arithmetic on vector-pipeline computer.* In: "Voprosy kibernetiki **145**, Host software for a supercomputer". Moscow, 1990, pp. 91–101 (in Russian).

10. Bamberger, L. *Error validation package for Reduce.* In: "Esprit' 88. Part 1", 1988.

11. Kulisch, U., Hammer, R. and Neaga, M. *Pascal-XSC.* Inst. of Applied Math., Karlsruhe, 1992.