A software system for algorithm stabilization technique

Yuji KONDOH
Department of Control Engineering
Takuma National College of Technology
551 Koda, Takuma, Kagawa, Japan
kondoh@dc.takuma-ct.ac.jp

Matu-Tarow NODA
Department of Computer Science
Ehime University
3 Bunkyo-cho, Matsuyama, Ehime, Japan
noda@cs.ehime-u.ac.jp

Abstract

In symbolic-numeric computations, an algorithm stabilization technique proposed by Shirayanagi and Sweedler is used effectively. The technique is accomplished by the interval arithmetic and zero-rewriting for given algorithm. However, it is not easy to transform an algorithm to stabilized one. Thus, we develop a system which performs the transformation above automatically. Input of the system is a program which is one of way to represent the algorithm. Input program to our system is restricted to a program written in Asir language and is first converted to a program with the interval arithmetic. Then the program is translated into stabilized program. Examples are also shown.

1 Introduction

In symbolic-numeric computations, an algorithm stabilization technique proposed by Shirayanagi and Sweedler[7] is used very effectively. The technique is accomplished by the interval arithmetic[1] and zero-rewriting for given algorithm. They have already proven the algorithm stabilization theorem in [7] for the case that input algorithm is algebraic and input value is exact. However, the application range of the technique is unknown still and various applications are expected. One of authors applied the technique to some algorithms and reported the effectiveness of it[4, 6]. Nevertheless, it is not easy to transform an algorithm to stabilized one.

In this paper we develop a system which performs the transformation above automatically. Input of the system is a program which is one of way to represent the algorithm. Input program to our system is restricted to a program written in Asir language because we use a computer algebra system Risa/Asir[5]. First, we improve Risa/Asir to handle the interval arithmetic. Therefore two data types of interval numbers, double and bigfloat type, are added into basic data types of Risa/Asir. Their primitive arithmetic operations are also added. Second, we develop a converter which converts input programs to stabilized programs. This system first converts given program to a program with the interval arithmetic for improved Risa/Asir. Then the program is translated into stabilized program.

Furthermore we develop a library to be necessary to execute stabilized programs which are results of the above converter. It is written in Asir language.

2 Algorithm stabilization technique

Shirayanagi and Sweedler proposed algorithm stabilization technique[7, 8]. Their motivation was tha computations by symbolic algorithms waste a lot of memories by intermediate swell of coefficients. Thus, if the algorithm is combined with a numeric computation carefully, results may be accurate and stable, and furthermore computations may be done quickly. The result of executing a symbolic algorithm is thought of as being obtained by infinite precision numeric computation. Thus by limited precision computation, the result must be obtained in a convergent fashion. As a numeric computation, a concept of interval arithmetic is introduced. Coefficients are described by a circular interval number, i.e. a pair of midpoint and small deviation. It is called a bracket coefficient. The stabilized algorithm is executed by increasing precisions of inputs, and then the result converges to the true output obtained by symbolic computation. For the convergence to be successful, zero rewriting is introduces. Zero rewriting is a rule that if a bracket coefficient contains zero, then the bracket coefficient is rewritten to zero.

The stabilization technique has the following points.

(ST1) The syntactic structure of the algorithm is unchanged.

(ST2) The coefficients are converted to bracket coefficients in the data set.

(ST3) Zero rewriting is performed prior to predicate evaluation.

(ST4) Repeat the algorithm by increasing digits used for computations.

Bracket coefficients are coefficients which have the form of intervals from interval analysis. Therefore we use rectangular intervals.

In recent research it is reported that zero rewriting is performed not only prior to predicate evaluation, but also posterior operation of interval arithmetic with respect to **(ST3)**. In [6] they performed so. In our system to explain below, this case corresponds to zero rewrite mode.

3 Interval arithmetic in Risa/Asir

We improve Risa/Asir to add interval numbers into basic data types in a category of numbers. Two types of interval numbers to add are as follows.

- Double interval type whose infimum and supremum are double floatingpoint numbers.
- Bigfloat interval type whose infimum and supremum are bigfloat.

Risa/Asir use PARI library[2] for the bigfloat. Therefore we use also the same bigfloat. In algorithm stabilization technique, the bigfloat should be used. However, in interval analysis computations, the double interval is usually used.

Further, the following functions are built in.

- Input of an interval number$A = [a_1, a_2]$: `intval(`a_1`,`a_2`) (= itv(`a_1`,`a_2`))`
- Convert from a number a to the interval number: `intval(`a`) (= itv(`a`))`
- A midpoint of an interval number A: `mid(`A`)` $= (a_1 + a_2)/2$
- A width of an interval number A: `width(`A`)` $= a_2 - a_1$
- An absolute value of an interval number A: `absintval(`A`)` $= \max(|a_1|, |a_2|)$.
- A distance between two interval number A, B: `distance(`A`,`B`) =` $\max(|a_1 - b_1|, |a_2 - b_2|)$.
- An infimum $\underline{A}$ of an interval number A : `inf(`A`)` $= a_1$.
- A supremum $\overline{A}$ of an interval number A: `sup(`A`)` $= a_2$.
- An intersection of two interval number A and B: `cap(`A`,`B`)` $= [\max(a_1, b_1), \min(a_2, b_2)]$, while result is not empty.
- A concatenation of two interval number A and B: `cup(`A`,`B`)` $= [\min(a_1, b_1), \max(a_2, b_2)]$.
- Inclusion test of a number a in an interval number A: *if* $a \in A$ *then* 1 *else* 0: `inintval(`a`,`A`)`.
- Set the precision of bigfloat to x words: `setprecword(`x`)`

In PARI the precision of the bigfloat is able to set at the word unit. `setprecword()` performs so. In Maple and Mathematica we can set at the decimal unit. But concerning actual calculation the word unit is natural.

We implement the zero rewrite mode. To set this mode, `ctrl()` function is used as follows.

```
ctrl("zerorewrite", 1);
```

while `1` means set, so we use `0` if we unset the zero rewrite mode. Once we set the zero rewrite mode, Asir always converts the interval which contains zero to **0** after operation of interval arithmetic.

4 Converter to stabilized programs

In this section we develop the converter to stabilized programs automatically using algorithm stabilization technique. Input of the system is a program which is one of way to represent the algorithm. Input program to our system is restricted to a program written in Asir language and converted to a program with the interval arithmetic which is use in improved Asir above. Then the program is translated into stabilized program. In order to implement we regard the algorithm stabilization technique **(ST1)**$\sim$**(ST4)** as follows.

(ST1) The structure of programs is unchanged. Therefore comments and spaces in given program are put at the same places.

(ST2) Arguments of functions in given program are converted to their interval expansions.

(ST3) In a case to use zero rewrite mode the function `ctrl()` to set at head of the program. Other case `zerorewrite()` function written in Asir language is used.

(ST4) For given function `a()` not only a new function `a()` with interval arithmetic but also `stabled_a()` is created. `stabled_a()` is a program to compute by increasing precisions of inputs in which the new `a()` is used with interval arguments.

For portability a parser of the converter is generated from grammar file by **bison**[3] which is compatible with **yacc** and is very famous tools. A part of lexical analysis are written in C. We test the our system in FreeBSD on PC. However porting to other operating system is easy.

And we develop a library written in Asir language. It necessary to accomplish the algorithm stabilization technique such as `zerorewrite()` in above **(ST3)**, functions of support convergence defined in [7] and so on. The library is loaded at head of output program.

For example we convert the following program to compute Sturm sequences.

```
/* Sturm sequences */

def sturm(P) {
    V = var(P); N = deg(P,V); T = newvect(N+1);
    G1 = T[0] = P; G2 = T[1] = diff(P,V);
    for ( I = 1; ; ) { /* infinite loop */
        if ( !(R = srem(G1,G2)) ) break; /* is remainder 0? */
        T[++I] = R; G1 = G2; G2 = R;
        if ( type(R) == 1 ) break; /* 1 means a type of number. */
    }
    S = newvect(I+1);
    for ( J = 0; J <= I; J++ ) S[J] = T[J];
    return S;
}
```

Our system generate the following stabilized program in case of zero rewrite mode.

```
load("./stabilization.asr");
ctrl("bigfloat",1);
ctrl("zerorewrite",1);
#define    LOOP_STAB    100
#define    LOOPCHECK    3
/* Sturm sequences */

def stabled_sturm(P) {
    CurrentWordPREC=1;
    Loopcheck = LOOPCHECK;
    setprecword(CurrentWordPREC);
    R1 =  sturm(tointval(P));
    RT1 = getsupports(R1);
    for(I=0;I<LOOP_STAB;I++) {
        CurrentWordPREC++;
        setprecword(CurrentWordPREC);
        R2 =  sturm(tointval(P));
        RT2 = getsupports(R2);
        if ( checksupport(RT1,RT2) ) {
            if ( --Loopcheck == 1 ) return R2;
        }
        else Loopcheck = LOOPCHECK;
        R1 = R2;
        RT1 = RT2;
    }
    print("Need more Precision.");
    return R2;

}
/* Sturm sequences */

def sturm(P) {
    V = var(P); N = deg(P,V); T = newvect(N+1);
    G1 = T[0] = P; G2 = T[1] = diff(P,V);
    for ( I = 1; ; ) { /* infinite loop */
        if ( !(R = srem(G1,G2)) ) break; /* is remainder 0? */
        T[++I] = R; G1 = G2; G2 = R;
        if ( type(R) == 1 ) break; /* 1 means a type of number. */
    }
    S = newvect(I+1);
    for ( J = 0; J <= I; J++ ) S[J] = T[J];
    return S;
}
```

New two functions sturm() and stabled_sturm() are generated. A head of program the library for the algorithm stabilization technique is loaded and modes are set. LOOP_STAB is a maximum number of loop for the case that result do not converge. LOOPCHECK is a number of continuity of the same supports for support convergence. In this example using zero rewrite mode sturm() is the same input one. This point is that our system is very convenient since our system has the basic type of interval number. In result program sturm() is used with its argument of interval expansion in stabled_sturm(). Thus we can use stabilized stabled_sturm() automatically.

In case of non-zero rewrite mode Our system generate the following stabilized program.

```
load("./stabilization.asr");
ctrl("bigfloat",1);
ctrl("zerorewrite",0);
#define    LOOP_STAB    100
#define    LOOPCHECK    3
/* Sturm sequences */

def stabled_sturm(P) {
    CurrentWordPREC=1;
    Loopcheck = LOOPCHECK;
    setprecword(CurrentWordPREC);
    R1 =  sturm(tointval(P));
    RT1 = getsupports(zerorewrite(R1));
    for(I=0;I<LOOP_STAB;I++) {
        CurrentWordPREC++;
        setprecword(CurrentWordPREC);
        R2 =  sturm(tointval(P));
        RT2 = getsupports(zerorewrite(R2));
        if ( checksupport(RT1,RT2) ) {
            if ( --Loopcheck == 1 ) return R2;
        }
        else Loopcheck = LOOPCHECK;
        R1 = R2;
        RT1 = RT2;
    }
    print("Need more Precision.");
    return R2;

}
/* Sturm sequences */

def sturm(P) {
    V = var(zerorewrite(P)); N = deg(zerorewrite(P),V); T = newvect(N+1);
    G1 = T[0] = P; G2 = T[1] = diff(zerorewrite(P),V);
    for ( I = 1; ; ) { /* infinite loop */
```

```
        if ( !zerorewrite((R = srem(zerorewrite(G1),zerorewrite(G2)))) )
break; /* is remainder 0? */
        T[++I] = R; G1 = G2; G2 = R;
        if ( zerorewrite(type(R)) == zerorewrite(1) ) break; /* 1 means a
 type of number. */
    }
    S = newvect(I+1);
    for ( J = 0; zerorewrite(J) <= zerorewrite(I); J++ ) S[J] = T[J];
    return S;
}
```

Similarly, new two functions `sturm()` and `stabled_sturm()` are generated. In non-zero rewrite mode `zerorewrite()` function is used where relation operations and built-in functions stay. New `sturm()` is included in many places. While comments are in the same places of input program.

5 Conclusion

A software system of algorithm stabilization technique is developed in Risa/Asir. The followings are done.

- Risa/Asir is improved to add the interval arithmetic.
- A converter from given program to stabilized program is developed.
- A library that is necessary of executions of stabilization programs is developed.

Input program to our system is restricted to a program written in Asir language and is first converted to a program with the interval arithmetic. Then the program is translated into stabilized program. Furthermore it is useful for not only advanced researches of stabilization technique, but also engineering applications of high quality and self validated computations such as robotics, mathematical programming and so on.

As our future works, we should modify our system to convert input programs written arbitrary programming language such as C to resulting stabilized programs.

References

[1] G. Alefeld, J. Herzberger, *Introduction to Interval Computations*, Computer Science and Applied Mathematics, New York, Academic Press, 1983.

[2] C. Batut, K. Belabas, D. Bernardi, H. Cohen, M. Olivier, User's Guide to PARI / GP, November, 2000.

[3] C. Donnelly, R.M. Stallman, Bison The YACC-compatible Parser Generator, Bison Version 1.25, Free Software Foundation, 1995.

[4] H. Minakuchi, H. Kai, K. Shirayanagi, M.-T. Noda, Algorithm stabilization techniques and their application to symbolic computation of generalized inverses, *Electronic Proceedings of the 3rd International IMACS Conference on Applications of Computer Algebra*, 1997.

[5] M. Noro, T. Takeshima, Risa/Asir — A Computer Algebra System, *Proceedings of ISSAC'92*, pp.387–396, 1992.

[6] K. Shiraishi, H. Kai, M.-T. Noda, Symbolic-numeric computation of Wu's method using stabilizing algorithm, *Proceedings of ATCM2001*, ATCM Inc., USA, pp.444–451, 2001.

[7] K. Shirayanagi, M. Sweedler, A Theory of Stabilizing Algebraic Algorithms, *Tech.Rep.95-28*, Cornell Univ., pp.1–92, 1995.

[8] K. Shirayanagi, M. Sweedler, Remarks on Automatic Algorithm Stabilization, *J. Symbolic Computation*, **26**, pp.761–765, 1998.