A program converter 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 combined 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. They already proved the algorithm stabilization theorem for the case that input algorithms are algebraic and input values are exact. However, applications of the technique are limited, today.

Therefore, we developed a system which performs to transform an input program to stabilized one automatically. Input program to the system is restricted to a program written in Asir language.

In this paper, we discuss the system which performs to transform a numerical computational program in C language to stabilized one in Asir language automatically. Inputs of the system are programs of numerical computations and written in C language. Outputs of the system are stabilized programs which are executed in compute algebra system Risa/Asir. Thus, the system is seemed as a program converter from a numeric program to a stabilized program.

1 Introduction

In symbolic-numeric combined computations, an algorithm stabilization technique proposed by Shirayanagi and Sweedler[8] is used very effectively. The technique is accomplished by the interval arithmetic[1] and zero-rewriting for given algorithm. They already proved the algorithm stabilization theorem in [8] for the case that input algorithms are algebraic and

input values are exact. However, applications of the technique are limited, today. One of authors applied the technique to some algorithms and showed the effectiveness of it[4, 6]. Nevertheless, it is not easy to transform an algorithm to stabilized one.

Therefore, we developed a system which performs to transform an input program to stabilized one automatically[3]. Input programs of the system are restricted to programs written in Asir language. Sekigawa and Shirayanagi developed automatic algorithm stabilization system whose target symbolic computation program in Maple[7]. Inputs of [7] are restricted to algebraic programs of computer algebra algorithms.

In this paper, we consider that inputs are numerical computational programs written in C language. Our new system performs to transform a program in C language to stabilized one in Asir language automatically. Inputs of the system are programs of numerical computations and written in C language. Outputs of the system are stabilized programs which are executed in compute algebra system Risa/Asir[5]. Thus, the system is seemed as a program converter from a numeric program to a stabilized program.

2 Algorithm stabilization technique

Shirayanagi and Sweedler proposed algorithm stabilization technique[8, 9]. 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.

1. The syntactic structure of the algorithm is unchanged.
2. The coefficients are converted to bracket coefficients in the data set.
3. Zero rewriting is performed prior to predicate evaluation.
4. 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. For further details and more general theory see[8].

3 The converter C language to stabled program in Asir language

In this section, we will develop a new converter to stabilized programs automatically using algorithm stabilization technique. Input to the system is a program which is one of way to represent the algorithm. The input to our system is a program written in C language. The program is translated in Asir language. Then the program are converted to a stabilized program with the interval arithmetic[3]. It seems that grammar of C language and Asir language are very similar. However, the details are different. Therefore, the converter carries out rewriting for the difference between grammar of C language and Asir language and for stabilization as follows.

- Identifiers (a variable name and a function name)

  In Asir, the head of a variable name must be a capital letter. And the head of a function name must be a small letter. Additionally, scope of an identifier is different between in C language and Asir language. Therefore, the variable name is add `V##_` in the head. The function name is add `f_` in the head. `##` means a number which is used for the difference of scope.

- Constants

  - Integer constants

    Asir don't accept octal notation. Therefore, octal notation is converted to acceptable expression. And Asir don't accept an integer with a suffix. The suffix is removed.

  - Character strings

    Separating character strings are automatically concatenated in C language. However it is not carried out in Asir. So separating caracter strings are converted to a concatenated string.

  - Floating-point number constants

    In the stabilization technique, the computation is repeated by raising the accuracy. Because of specification of a bigfloat number in Asir, it is necessary that the number is made into a rational number and expanded to interval number. If the expression of the number has any suffix, it will be removed. For example, `1.23` is converted to the following result.

    ```
    tointval(123*10^(-2))
    ```

    Function `tointval()` is a function for expansion to the interval number.

- Variable declarations

  There is no variable declaration in Asir. Therefore, it may be removed except the global variable. However, it is necessary to rewrite as an assignment statement, when there is an initializer.

- Extern declaration

  Since the effective scope of local and global variable is different between C language and Asir language, it is necessary to process such as addition of the extern declaration and change of the variable name above.

- Array declarations

  An array type corresponds with a vector type and a matrix type in Asir. A dimension of the vector type is 1. A dimension of the matrix type is 2. 3 or more dimension array must be represented by the vector. To convert simply, `array()` function is generated. It carries out generating arbitrary dimension array using the vector type. Additionally, when array declaration has the initializer, `array()` is assigned initial values to generated array. For example,

  ```
  double d[3] =  {1, 2, 3};
  ```

  is converted to the following result.

  ```
  V0_d=array([3], [1, 2, 3]);
  ```

- Pointers

  Since there is no pointer in Asir, it is very difficult to convert the pointer to the program without a pointer. Only if pointer is used in argument of a function and the argument is used as array in the function, it is possible to rewrite the program. In this case, pointer dereference operator(`*`) is removed. For example,

  ```
  double f(double *x) {
  ...
      x[i] = 0.0;
  ...
  }
  ```

  is converted to the following result.

  ```
  def f_f( V0_x) {
  ...
      V0_x[V0_i] = 0;
  ...
  }
  ```

  At present, it is impossible that a program has a pointer of function, an address operator(`&`) and more complex pointers. This is a future work.

- Predicates

  In stabilization technique, predicates are rewritten with the comparison with 0. And zero rewriting is performed prior to predicate evaluation. In C language, predicates

correspond to relational operation(<, <=,>,>=), equality operation(==,!=), logical complement operator(!), conditional operation(? :) and if statement. Therefore, when predicate is not the comparison with 0 except it is original 0, the transposition is done, and it is rewritten for the comparison with 0. And, it is necessary to change the rewriting processing of the program by whether the automatic zero rewriting function is set in Asir[3]. For example, when the automatic zero rewriting function is unset, they are converted to such as follows:

```
a<b     →     zerorewrite(V0_a-(V0_b))<0,
a>=b    →     zerorewrite(V0_a-(V0_b))>=0,
a==0    →     zerorewrite(V0_a)==0,
! a     →     ! zerorewrite(V0_a),
```

where `zerorewrite()` carries out zero rewriting.

- Control structure

  Asir have same control structures: `if` statement, `for` statement, `while` statement and `do-while` statement in C language. Therefore it is very easy to convert these structure. However, Asir don't have `switch` statement. It is necessary to convert `switch` statement to a complex structure using `if-else` statement. It is very difficult. At present, it is possible to convert simple `switch` statement whose `case` and `break` are correspondent to one-to-one. It is a future work that other kinds of `switch` statement are processed.

- `typedef` declaration

  In Asir language, there is no variable declaration. Therefore, it is only used in order to judge whether it is a new variable declaration or not. And it is removed from an output.

- Bitwise operations

  Asir don't have bitwise operators(AND operator `&`, exclusive OR operator `^` and inclusive OR operator `|`), bitwise complement operator `~`, shift operators(`<<` and `>>`). Therefore, they are converted to the follows:

```
a&b     →     iand(V0_a,V0_b)
a^b     →     ixor(V0_a,V0_b)
a|b     →     ior(V0_a,V0_b)
~a      →     ixor(0xffffffff, V0_a)
a<<b    →     ishift(V0_a,V0_b)
a<<b    →     ishift(V0_a,-V0_b)
```

  where `iand()`, `ixor()`, `ixor()` and `ishift()` are built-in functions in Asir, and `0xffffffff` is depended on a type of CPU.

At present, the following items are not implemented yet. These are only rarely used in the numerical computation.

- Standard functions

  It is necessary to individually prepare the standard functions in C language such as `printf()` and `scanf()`.

- Enumeration constants

  It should be global variables with assignments of the initial value.

- Structures

  It should be converted to a structure in Asir.

- Comma expressions

  It is impossible to use A comma expression except for inside of the parenthesis of `while` and `for` statements. Therefor, it is necessary to rewrite to statements separately.

- C Preprocessor (CPP).

  At present, an input to our system is a program after processing of the preprocessor. In future, it should be a program before the processing.

- A union type and `goto` statement

  It must be impossible to rewrite these.

4 Implementation and experiments

4.1 Implementation

For portability, a parser of the converter is generated from grammar file by **bison**[2] 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 5.1-RELEASE on PC. However, porting to other operating system is very easy.

4.2 An example

Our system can not carry out a program which has a pointer of function, an address operator and more complex pointers. The program of the Gauss-Jordan iteration method in [10] is taken up as a standard numerical computation program which satisfies our constraint. It was confirmed that the result program normally operated for a numerical stabled input. The details of the conversion result is shown below.

4.2.1 Conversion for the difference of the languages

Here, rewritten part for the difference between C language and Asir language is shown. A following is a part of Gauss-Jordan method program in C language[10].

```
void gaussj(float **a, int n, float **b, int m)
/* Linear equation solution by Gauss-Jordan elimination, ... */
{
    int *indxc,*indxr,*ipiv;
    int i,icol,irow,j,k,l,ll;
    float big,dum,pivinv,temp;

    indxc=ivector(1,n); /* The integer arrays ipiv, indxr, and indxc */
    indxr=ivector(1,n); /* are used for bookkeeping on the pivoting */
    ipiv=ivector(1,n);

...

    free_ivector(ipiv,1,n);
    free_ivector(indxr,1,n);
    free_ivector(indxc,1,n);
}
```

`ivector()` and `free_ivector()` are functions of the memory management commonly used in the literature[10]. Our system generate the following part of stabilized program in Asir language.

```
def f_gaussj( V0_a,  V0_n, V0_b,  V0_m)
/* Linear equation solution by Gauss-Jordan elimination, ... */
{

    V0_indxc=f_ivector(1,V0_n); /* The integer arrays ipiv, indxr, and
 indxc */
    V0_indxr=f_ivector(1,V0_n); /* are used for bookkeeping on the piv
oting */
    V0_ipiv=f_ivector(1,V0_n);

...

    f_free_ivector(V0_ipiv,1,V0_n);
    f_free_ivector(V0_indxr,1,V0_n);
    f_free_ivector(V0_indxc,1,V0_n);
}
```

In this result, it is rewritten so that the function definition uses `def` statement in the Asir language. Variable declarations are removed. `f_` is added to the function name and `V0_` is added to the variable name. The comments are output as they are.

4.2.2 Rewriting for the stabilization technique.

Here, rewritten part for the stabilization technique is shown. A following is an other part of Gauss-Jordan method program[10].

```
...
                if (fabs(a[j][k]) >= big) {
                    big=fabs(a[j][k]);
                    irow=j;
                    icol=k;
                }
...
if (a[icol][icol] == 0.0) nrerror("gaussj: Singular Matrix-2");
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
```

`nrerror()` is a functions of the error process. Our system generate the following part of stabilized program in Asir language.

```
...
                if (zerorewrite(f_fabs(V0_a[V0_j][V0_k])-( V0_big)) >=0)
 {
                    V0_big=f_fabs(V0_a[V0_j][V0_k]);
                    V0_irow=V0_j;
                    V0_icol=V0_k;
                }
...

if (zerorewrite(V0_a[V0_icol][V0_icol]) ==0) f_nrerror("gaussj: Singular
 Matrix-2");
V0_pivinv=tointval(10*10^(-1))/V0_a[V0_icol][V0_icol];
V0_a[V0_icol][V0_icol]=tointval(10*10^(-1));
```

In the first `if` statement, it is rewritten such that the transposition is done and it is converted to the comparison with 0. In the second `if` statement, since it was the comparison with 0, there is no necessity of such rewriting. And the floating-point data is made the rational number and expanded the interval number using `tointval()` function. In Risa/Asir which set the interval arithmetic function up, if the data is the interval number, it is automatically calculated in interval arithmetic.

4.3 Experiments

We attempt to convert the numerical computation programs in [10]. There are about 350 programs in [10]. About 52% of all the programs are satisfied with our constraint: there aren't in a pointer of function, an address operator and more complex pointers input programs. We success to convert 97% of the satisfied programs to stabilized one. There are standard function such as `scanf()` in failed programs.

5 Conclusion

A new program converter for algorithm stabilization technique is developed. Inputs to the converter system are numeric programs written in C language and outputs of it are stabilized programs. The inputs are first converted to programs in Asir language with the interval arithmetic. Then the program is translated into stabilized programs. 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. Inputs of our system are restricted to numeric programs without a pointer of function and an address operator in C language.

As our future works, we should refine and modify our system to convert input programs written other programming language for numerical computation such as FORTRAN 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. Donnelly, R.M. Stallman, Bison The YACC-compatible Parser Generator, Bison Version 1.25, Free Software Foundation, 1995.

[3] Y. Kondoh, M.-T. Noda, A software system for algorithm stabilization technique, *Proceedings of the 7th Asian Technology Conference in Mathematics*, ATCM Inc., pp.360–367, 2002.

[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] H. Sekigawa and K. Shirayanagi, Automatic Algorithm Stabilization System, Josai Mathematical Monographs 2, NLA'99 Computer Algebra, pp. 159–168, 2000.

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

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

[10] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988.