数式処理 *J.JSSAC (2005)*
Vol. 11, No. 3,4, pp. 29 - 34

Noda2005 論文

数値計算プログラムの自動安定化変換システムの構築

近藤 祐史* 野田 松太郎†

詫間電波工業高等専門学校 愛媛大学 工学部

概 要

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 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. In this paper, we improve the system to check a type of variables and constants and to convert only for the floatingpoint number type.

1 歴史的経緯

白柳らは，代数的アルゴリズムを近似計算で行ったときに起きる不安定な場合に対し，アルゴリズムの安定化手法を提案し理論の証明を行った [5][6]．この手法を用いることは，数値数式計算を行う一つの有効な方法である．安定化手法の基本は，

(STAB1) アルゴリズムの構造は変えない
(STAB2) データ領域において普通の係数を区間数に変える
(STAB3) 条件文等の述語の直前において 0 を含む区間数を 0 と見なすゼロ書換えを行う

である．これらにより変換されたアルゴリズムを，精度を上げながら再計算することにより，真の解を得るかまたはそれに近づく解を得ることができる．しかし，この技術の適用範囲は未だ不十分である．そこで，既存のプログラムに対し，安定化手法が適用可能であるかどうかを検討することは非常に重要である．既存のプログラムを安定化手法を用いたものへ書換えることは，必ずしも容易ではない．そのため，自動的に安定化手法を用いたプログラムへ書換えるシステムの開発が必要である．そのようなシステムとして，関川らは数式処理システム Maple を用いて，Maple で書かれたプログラムの自動安定化を行うシステムを開発した [4]．また，筆者らは

*kondoh@dc.takuma-ct.ac.jp
†noda@cs.ehime-u.ac.jp

数式処理システム Risa/Asir によって，Asir 言語で書かれたプログラムに対して自動安定化を行うシステムを開発した [1]．これらは，数式処理のプログラムを自動安定化することを目的としている．

2 数値計算プログラムの安定化

筆者らは，安定化手法の適用範囲の対象を従来の数式処理のアルゴリズムではなく数値計算とした場合の研究を遂行するために，数値計算プログラムを自動安定化するシステムの構築を目指した [2][3]．[3] で提案したシステムは，実行環境を区間演算機能を有効にした Risa/Asir として，C 言語で書かれたプログラムを Asir 言語へ変換することにより実現したが開発途中での報告であり不十分であった．本論文では，これらをまとめるとともにシステムの改良を行ったので報告する．

3 プログラムの自動安定化変換技法

安定化手法を施したプログラムの実行環境は，区間演算機能を有効にしている Risa/Asir[1] とする．区間演算機能を有効にしている Risa/Asir では，**(STAB3)** を実現するために，区間演算の直後にゼロ書換えを自動的に行う自動ゼロ書換え機能を実装している．そのため，この機能を使用するか否かによって対応する必要がある．

自動安定化には，安定化手法を用いたプログラムに書換えるトランスレータが必要である．このトランスレータの入力は C 言語とし，出力は安定化手法を用いた Asir 言語のプログラムとする．書換えは，アルゴリズムの安定化手法に従い，

- **(STAB1)** に関しては，プログラムの構造は変えない．そのため，コメント等も変えない
- **(STAB2)** に関しては，浮動小数点数データ入力は区間数変換する
- **(STAB3)** に関しては，計算直後にゼロ書換えを行うときは自動ゼロ書換え機能を有効にし，述語の直前に行うときには自動ゼロ書換え機能を無効にする．両方ともにそれぞれ対応した書換えをする

という方針で行う．具体的には，C 言語と Asir 言語の違いを変換するため，また安定化のため次のような処理を行う．

- 識別子（変数名，関数名など）
  Asir では，変数名の先頭は大文字アルファベット，関数名の先頭は小文字アルファベットでなければならない制約がある．C 言語では，どちらでもいいため変数名と関数名を書換える必要がある．変数名には先頭に `V##_` を付加し，関数名には先頭に `f_` を付加する．ここで，`##` はローカル変数名が重複しないように自動生成した数字を表す．
- 定数

  整数（整定数） 8 進数への対応，L や U などの接尾子の削除

  文字，文字列 C 言語において行われているソースコード中の文字列の連結は行われないので，連結されたものに書換える．

浮動小数点数 安定化手法のために精度を上げて再計算するときの精度指定のために，有理数化しかつ区間数化する．ここでも，接尾子は削除する．例えば，1.23 は，

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

と書換える．ここで，tointval は区間数化するための Asir 内部関数である．

- 外部宣言
  局所変数と大域変数の有効範囲が C 言語とは異なるため，extern 宣言の追加や変数名の変更などを行い書換える．
- 変数の宣言と初期化
  Asir 言語には，型の概念がないため変数の宣言は必要ない．そのため，基本的には削除すれば良い．しかし，初期化がある場合には代入文として書換える必要がある．また，配列については，Asir のベクトルへ書換える．1 次元の配列の場合は容易であるが，2 次元以上になると繁雑になるため，配列関係を処理する array 関数を作成し，array 関数を使用するように書換える．
  例
  ```
  double d[3] = {1, 2, 3};  ⟹  V0_d=array([3], [1, 2, 3]);
  ```
- 述語（関係演算子など）
  C 言語では，関係演算子（<,<=,>,>=），等値演算子（==,!=），否定（!）が，データの比較のために使われている．これらは，安定化手法において，0 との比較に変更し，ゼロ書換えを行った上で 0 との比較を行う必要がある．そのため，元のプログラムが 0 との比較であった場合は書換える必要はないが，0 との比較ではない場合には移項し 0 との比較に書換える．また，区間演算を有効にした Asir では，自動ゼロ書換え機能が有効かどうかによって，プログラムの書換え処理を変える必要がある．
  例

  自動ゼロ書換え機能有効時
  ```
  a<b   →  V0_a-(V0_b)<0
  a<0   →  V0_a<0
  a==0  →  V0_a==0
  !a    →  ! V0_a
  ```
  自動ゼロ書換え機能無効時
  ```
  a<b   →  zerorewrite(V0_a-(V0_b))<0
  !a    →  ! zerorewrite(V0_a)
  ```

  [3] ではすべての述語に対し書換えを行っていたが，浮動小数型数のデータを用いている場合のみにする必要がある．そのため，次節で改良する．
- 制御構造
  if 文，for 文，while 文，do-while 文といった制御構造は，同じ仕様であるため変更する必要はない．しかし，switch 文は Asir 言語にはない．そのため，if-else 文を用いた記述に書き換える必要がある．現在，switch 文中の case と break が 1 対 1 に対応

するもののみ書き換え可能である．多対 1 のものは今後の課題である．

- 型定義 typedef
  Asir 言語には，型の概念がないので必要ないが，新しい型名での型宣言を有効にするための処理を行っている．
- ビット演算子
  C 言語のビット演算子 (&,^,|, ~, <<, >>) は Asir 言語にはないが, 組込み関数として iand(), ior(), ixor(), ishift() などがあるため，それらへ変換する．
  例

```
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)
```

  ただし，0xffffffff は使用する計算機の CPU に依存する．
- ポインタ
  Asir にはポインタがないため，ポインタを用いたプログラムをすべて扱うことはできない．しかし，変数をポインタとして宣言していても配列参照の形式で記述されている場合のみ書き換えできる．この書換えは，配列を関数へ引数として渡し，関数内では配列参照として記述されている場合を仮定している．
  例

```
double f(double *x) {                  def f_f( V0_x) {
...                                    ...
    x[i] = 0.0;            ⟹              V0_x[V0_i] = 0;
...                                    ...
}                                      }
```

  ポインタ参照で値を参照したり代入したりしている場合は，動作が保証されない．これは，今後の課題である．

現在，次のものは未対応となっているが，対象としている数値計算プログラムではあまり使われない．

- 列挙定数
- for 文，while 文の括弧中以外のコンマ式
- 構造体
- プリプロセッサ（CPP）
- 共用体
- goto 文
- 多種多様な標準関数

表 1: Timing data（単位は秒）

次元	改良前			改良後		
	CPU	GC	Total	CPU	GC	Total
10	0.0097	0.0040	0.0137	0.0047	0.0020	0.0067
20	0.0668	0.0274	0.0942	0.0324	0.0140	0.0464
30	0.2177	0.0863	0.3040	0.1060	0.0374	0.1434
40	0.5079	0.2006	0.7085	0.2528	0.0833	0.3361
50	0.9781	0.4346	1.4127	0.4966	0.1794	0.6760

4 改良点

文献 [3] では，プログラム中のすべての述語を対象に **(STAB3)** に関する書換えを行った．数式処理のプログラムなど型宣言を持ったない言語で記述されたプログラムに対し安定化する場合には，すべての述語に対し書換えを行う方が安全である．しかし，`for` 文等の繰り返しで用いる制御変数や配列の要素番号を表すための変数は整数でしかありえず，それら比較で構成される述語は，**(STAB3)** に関する書換えを行う必要はない．また，数値計算のプログラムは，技巧的なものを除くと，整数型で宣言されている変数や整数定数のみの演算は正確計算を目的としていると考えて差し支えない．対象としている C 言語のプログラムでは，変数は型を宣言する必要があり，変数宣言と計算式の記述から浮動小数点数型を用いているか否かの判定が容易である．そのため，[3] では余分な書換えを行っていたことになる．そこで，述語において各辺が整数型間の演算の場合は書換えをしないように改良を行った．

実行例の入力として [2] で用いた [7] のガウス・ジョルダン法のプログラムを取り上げる．出力は，自動ゼロ書換えを行わないつまり明示的に `zerorewrite` 関数を用いるモードで行った．結果，改良前述語の書換えがプログラム中 21 箇所であったのが，改良後 2 箇所となった．また，出力されたプログラムを Asir 上で実行したのタイミングデータを表 1 に示す．データは，乱数により生成した 100 個の連立 1 次方程式を double 型の区間演算を用いて解いたときの時間の平均である．これらの方程式では安定して解を計算できた．また，使用した計算機の CPU は AMD Athlon XP 3000+，OS は FreeBSD 5.3 である．表 1 より余分な処理をさせないようにプログラムを書換えることで計算時間が半分になったことがわかる．

5 まとめ

文献 [7] にある標準的な数値計算プログラムについては変換できるような，C 言語プログラムの自動安定化変換システムの開発を行っている．本論文では，不必要な述語の書換えをしないように改良を行った．型のチェックを行い浮動小数点数型の変数やデータを含む述語のみ書換えを行うことにより変換後のプログラムの実行速度を改善することができた．

参 考 文 献

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

[2] 近藤 祐史，野田 松太郎，プログラムの自動安定化変換について，京都大学数理解析研究所講究録 1335, 2003, 181–187.

[3] Y. Kondoh, M.-T. Noda, A program converter for algorithm stabilization technique, *Proceedings of the 8th Asian Technology Conference in Mathmatics*, **1**, ATCM Inc., 2003, 134–143.

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

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

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

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