2V-06　並行論理プログラム自動修正系 *kima* における誤り検出率の向上とその効果

網代育大† 上田和紀‡

† 早稲田大学大学院 理工学研究科　‡ 早稲田大学 理工学部

1 はじめに

kima 第2版は、強モード／型体系の下で、並行論理プログラムにおける変数の少数個の誤りを静的に修正することができる。誤りの検出および修正の際、*kima* 第2版は、プログラムテキストから得られるモードや型に関する制約の冗長性だけを利用しており、プログラムに関する明示的な仕様や宣言を何ら必要としない。だがそのために、誤りの検出率は、モードと型情報を併用しても 70 % 程度であった。

しかし、完全な仕様や宣言を与えるのは、プログラマにとって負担が大きいため、新たに *kima* 第2.1版を実装し、モード／型情報の他に、構文上の簡単な規則やヒューリスティクスを、誤り検出時や修正案探索時に利用するようにした結果、いくつかの実験では、検出率が 90 % 以上に向上することが分かった。ここでは、その詳細を説明するとともに、修正案探索の効率化についても述べる。

2 検出レベルの導入

kima 第2.1版が、モード / 型以外に利用している情報は、大きく以下の2つに分けられる。

i) 処理系が検査しない構文上の規則 (**検出レベル 1**)

(1) ガードで検査している変数がヘッドになくてはならない

(2) ユニフィケーションの両側に同一の変数が出現してはならない (部分的な occur-check)

ii) 統計的にどういったプログラムがもっともらしいかを規定したもの (ヒューリスティクス)

(1) プロセス (述語) 同士の1対1通信がもっともらしい (基本的である)

(2) プログラム中のあるパスとその car とに同一の変数が出現している場合はもっともらしくない

kima は、誤りを含んだプログラムから得られるモード / 型制約集合の中から、矛盾する極小部分集合を求めることによって、誤りを検出、およびその場所を特定し、その周辺の変数を書換えた結果、制約集合が充足可能となるかどうかを調べる generate&test 方式で、修正案を探索する [2]。

修正案は一般に多数求まるが、ii のヒューリスティクスを使って、求めた修正案に優先度をつけることで、修正案の品質を向上させることができる [1]。つまり、ii の「もっともらしい (らしくない)」は、「プログラム中の変数を少数個だけ書換えた (モード / 型の正しい) プログラムの集合というものを考えたとき、それを極力守っているものの方が、意図したプログラムに近い (遠い) であろう」ということを意味している。

並行論理型言語では、節中の変数がプロセスの通信路に対応しているため、ii-(1) は、「変数が linear に出現 [4](ヘッドとボディに1回ずつか、あるいはボディに2回出現) しており、ボディに2回の場合は、異なる述語呼び出しの引数に出現している方がもっともらしい」と言い換えることもできる。

ii-(2) は、それがリダクションされるたびに、あるリスト構造の深さが深くなっていくような節を意味しており、そのようなプログラムは現実にはほとんど見られない。

一方、i の2つの規則への抵触は、永久中断や無限ループの原因となるため、プログラムとして明らかに誤りであり、誤りの検出に常に利用できる。また、ii-(1) で特に、変数が singleton (1回のみ) 出現している場合は、その変数が持っているデータがそこで破棄されることを意味しており、Prolog の主な処理系は、そのような変数の名前が "_" で始まっていない場合に警告を発する。そこで新たに、次の規則を誤り検出に利用することを考える。

(ii-1′) 下線 "_" で始まっていない変数の singleton 出現は誤り (**検出レベル 2**)

つまり、データを捨てることの意思をプログラマに確認することで、それが誤りかどうかを識別するが、その負担は軽いと言える。

kima 第2.1版では、新たに検出レベルの概念を導入し、誤りを検出する際に、あるレベルまでの規則を、選択的に利用できるようにしている。これらの検出規則を利用した場合の検出率の向上と求まる修正案の数を表1に示す[1]。ただし、変数名を "_" と誤るような間違いについ

Improvement of the Detection Rate and its Effectiveness by Means of Heuristics in *kima*, an Automated Error-Correction System for Concurrent Logic Programs.
Yasuhiro AJIRO, Kazunori UEDA
Department of Information and Computer Science, Waseda University

[1] 文献 [1] における同種の実験では、ガードの書き誤りをカウントしていないために実験総数が異なっているので、比較の際は注意されたい。

表 1: 変数の 1 箇所の書き間違いに対する検出率と修正案の数

プログラム	検出レベル	優先度づけ	実験総数	検出成功	検出率 (%)	求まる修正案の数 1	2	3	4	5	6	7	≥8
append	0	なし	58	36	62.1	1	3	8	3	6	5	3	7
	1	なし	–	40	69.0	2	12	4	3	5	6	4	4
	2	なし	–	58	100	21	12	3	3	9	7	3	0
	2	あり	–	–	–	39	19	0	0	0	0	0	0
fibonacci	0	なし	124	74	59.7	18	13	4	15	9	0	6	9
	1	なし	–	94	75.8	34	20	13	15	7	2	3	0
	2	なし	–	105	84.7	62	6	11	11	5	5	1	4
	2	あり	–	–	–	75	20	8	0	2	0	0	0
quicksort	0	なし	304	225	74.0	53	76	8	59	0	9	18	2
	1	なし	–	240	78.9	80	98	20	19	0	7	16	0
	2	なし	–	290	95.4	146	101	21	16	3	0	2	1
	2	あり	–	–	–	203	84	2	1	0	0	0	0

ては、考慮していない。また、検出レベル 0 では、検出にモード / 型情報だけを使い、レベル 1 では、それに加えて検出レベル 1 の規則、レベル 2 では、レベル 1 と 2 両方の規則を使っている。

3 修正案探索の最適化

検出レベルを利用した誤りの検出は、対象とする節だけを調べれば良いため、節中の記号数に比例する手間で行なうことができる。それに対し、モード / 型の検査には、プログラム全体の解析が必要であり、プログラムサイズに (ほぼ) 比例する手間がかかる [3]。

そのため、修正案探索時に、generate した書換え案を test する際、モード / 型解析の前に、検出規則 (レベル 1, 2) を使った検査を行うことで、モード / 型解析の回数を減らし、探索の効率を向上させることができる。

また、前節 ii のヒューリスティクスによる修正案への優先度づけに関しても同様で、優先度の高い修正案だけを求めたい場合は、モード / 型解析による検査の前に優先度づけを行なって、優先度の高いものだけをモード / 型解析するようにすれば、同じく探索の効率を向上させることができる。節のサイズは、プログラムサイズに関係なくほぼ一定なため、プログラムサイズが大きいほど、向上率も大きくなるはずである。

これらの最適化を使った自動修正アルゴリズムを図 1 に示す。修正案の探索は、書換え個数に関する深さ漸増探索であり、アルゴリズム中の *depth* はその探索の深さを表している。

```
モード制約の矛盾する極小部分集合を計算;
そこから疑わしい節および記号を抽出;
検出規則 (レベル 1, 2) に抵触する節と記号の検出;
depth ← 1;
while 解が見つかっていない do
  while 指定された優先度以上の depth 個の変数の
        書換え方がまだある do
    その depth 個の記号を書換える;
    if 書換えたプログラムが検出規則に抵触しない
      then
      if 書換えたプログラムが well-moded/typed
        then その書換えを修正解として出力
  end while;
  depth ← depth + 1;
end while
```

図 1: 自動修正アルゴリズム (改良版)

参考文献

[1] 網代育大, 上田和紀, 並行論理プログラム静的解析系 Kima の実装. 情報処理学会第 58 回全国大会論文集, 1999, pp. 397–398.

[2] Ajiro, Y., Ueda and K., Cho, K., Error-Correcting Source Code. In *Proc. Fourth Int. Conf. on Principles and Practice of Constraint Programming (CP'98)*, LNCS 1520, Springer, 1998, pp. 40–54.

[3] Ueda, K. and Morita, M., Moded Flat GHC and Its Message-Oriented Implementation Technique. *New Generation Computing*, Vol. 13, No. 1 (1994), pp. 3–43.

[4] Kazunori Ueda, Linearity Analysis of Concurrent Logic Programs. To appear in Proc. International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications, World Scientific Publishing, 2000.

[5] 網代育大, *kima* — an Automated Error-Correction System for KL1 Programs. Available from `http://www.ueda.info.waseda.ac.jp/~ajiro/study-j.html`, 2000.