1E3-02

反復深化 A* 探索によるもっともらしいプログラムの効率的な生成

Generating Plausible Programs Efficiently by Iterative-Deepening A* Search

網代育大　　上田和紀
Yasuhiro Ajiro　Kazunori Ueda

早稲田大学理工学部情報学科
Dept. of Information and Computer Science, Waseda University

We have implemented the Kima system, which automatically corrects wrong occurrences of variable symbols in concurrent logic programs under strong moding and typing in the absence of the explicit declarations of programs. Automated error correction is attempted basically by generate-and-test search, namely the generation of possible rewritings and the computation of their principal modes and types. Search space is kept small because the locations of bugs can be limited to small regions of program text by computing the minimal inconsistent subsets of mode and type constraints. In terms of the semantics of the mode system, we have found there are syntactical constraints which *plausible* programs should observe. The syntactical constraints we have presented as heuristic rules turned out to be quite effective not only for multiple alternatives proposed by Kima to be reduced but also for the optimization of searching alternatives to errors. We explain this optimization technique by showing the algorithms of an iterative-deepening A* search with respect to the plausibility.

1. はじめに

現在、現実的なプログラム開発支援システムとしては、ステップ実行等の機能を備えたデバッガやエディタなどの統合開発環境があるのみであり、プログラミング作業の知的支援の観点からは、多くの技術課題を残している。一方で、計算機性能の大幅な向上によって、我々の身の回りの計算機の多くは、その計算能力を持て余している。この余剰計算能力とプログラム解析技術とを組み合わせることにより、誤りを含むプログラムや一部の欠落したプログラムから正しいプログラムを計算機が予測し提示してくれるようなシステムを作れないか、というのが本研究の基本的な動機である。

計算機の支援を借りてプログラムの正しさや安全性を保証するための代表的な手法に、定理証明に基づくプログラム検証技術が挙げられる。しかし、プログラムの完全な検証には、現実サイズのプログラムに対して適用するのが難しい、プログラムの (正しい) 仕様を記述するするための指針が明確でないといった問題点があり、実用的かつ汎用的な検証システムを構築するのは非常に困難な状況である。

現在、プログラムの正しさを部分的に保証するための技術としてもっとも成功し普及しているのは型体系である。強い型づけ (strong typing) を備えた言語では、型を静的に検査ないしは推論することによって、型に関するプログラムの誤りを事前に検出することが可能である。並行論理型言語においては、情報の流れを表現するモード体系が提案されているが、これもまた広い意味での型体系の一種である。並行論理型言語における強モード体系は、プログラムテキストから得られるモード制約式を制約充足問題として解くことで、ユーザがモード宣言やプログラムの仕様を与えることなく、プロセス間通信プロトコルの整合性 (ないしは変数に対する読み書きのケイパビリティ) を静的に判定することを可能にする [5]。並行論理型言語においては、データ型についても同様の解析が可能である。

Moded Flat GHC プログラム中に誤りが存在する場合、誤った記号出現によって課せられるモード制約が、他の (正しい) 制約集合と矛盾を起こすことが多い。制約集合全体の中から矛盾する極小の部分集合を求める効率的なアルゴリズムが、我々のグループによってすでに開発されている。モード制約はそれぞれプログラムテキスト中の記号出現によって課せられるため、矛盾する極小集合は、モード制約の矛盾に対する極小の「説明」になっていると同時に、誤り箇所の候補を指し示している。この技法を用いることで、複数の独立な誤りを一度に検出することも可能である [4]。

本研究では、対象言語として高速な実装が入手可能な並行論理型言語 KL1 と、プログラム解析の枠組として Moded Flat GHC 言語の備えるモード/型体系を利用した。両言語はいずれも GHC をベースにしており、Moded Flat GHC のモード/型体系は KL1 に対しても適用可能である。このモード/型体系の下で、前述の誤り同定技術をさらに発展させ、発見した誤りを自動的に修正するシステム Kima を開発した [1, 2, 3]。

Kima は基本的に生成検査法 (generate-and-test) によって修正案を探索する。Kima の持つ 3 つの大きな特徴を以下に示す。

1. プログラムに関する明示的な仕様や宣言をプログラマが与えることなく、プログラム中に出現する論理変数の少数個の書誤りを自動的に修正する。
2. 点在する複数個の誤りを一度の解析で修正するばかりでなく、近接箇所における k 個の誤りを、深さ k の探索によって修正する。
3. あるモジュールに含まれる述語群など、プログラムの一部に対して適用できる。

Kima の備える自動修正の機能はまた、プログラム中の誤り部分を対象とした、小規模な自動プログラミング技術であるとみなすこともできる。

本研究では、「もっともらしい」並行論理プログラムが満たす構文上の制約が存在することを突き止め、ヒューリスティクスとして示した。モードおよび型情報のみを用いた場合、誤りに対する修正案は一般に複数求まるが、ヒューリスティクスを併用することで修正案を 1 つあるいはごく少数に絞り込むことができる。また、示したヒューリスティクスは、修正案の品

連絡先: 網代育大, 早稲田大学理工学部 (現所属: 日本電気 (株)),
〒 169-8555 新宿区大久保 3-4-1, Tel/Fax: 03-5285-7882,
{ajiro,ueda}@ueda.info.waseda.ac.jp

表 1: もっともらしくない変数出現に対するペナルティ

変数出現	ペナルティ
(i). singleton 出現	
- 変数名が “_” で始まっていない	
* 変数が頭部に節の頭部に出現	2
* 変数が頭部に節のボディに出現	3
- 変数名が “_” で始まっている	1
(ii). ヘッドで 2 回出現	2
出現数が 1 増えるごとに	1
(iii). ヘッドまたはボディに 3 回出現	1
出現数が 1 増えるごとに	1
(iv). ある述語呼び出しの引数に 2 回出現	2
出現数が 1 増えるごとに	2

質の向上に役立つばかりでなく、修正案の探索の最適化にも有効に作用することを、反復深化 A* 探索アルゴリズムの提示によって示す。同一節中に 2 箇所の誤りを含んだクイックソートプログラムのある例では、最適化によって 20 倍の高速化が観察された。

2. 自動修正アルゴリズムの最適化

2.1 優先度づけと検出規則による最適化

Kima は、プログラムに関する明示的な仕様や宣言を必要としないことを特徴としており、モードや型情報を用いて修正案を求めるため、修正案は一般に複数求まる。そこで Kima は、以下の**ヒューリスティクス**を用いて修正案に優先順位をつける。

ヒューリスティクス. 節中において、ある変数が (i) singleton (1 回のみ) 出現、(ii) ヘッドで 2 回以上出現、(iii) ヘッドまたはボディに 3 回以上出現、(iv) ある述語呼び出しの引数に 2 回以上出現しているようなものは、もっともらしくない。

Kima は、修正案に含まれるこれらのもっともらしくない部分に表 1 に示すペナルティを課す。ペナルティの総和の小さい修正案が優先度の高い修正案となる。

また Kima は、以下の**検出規則**を設けて検出力の強化を図っている。

検出規則 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

検出規則 2. singleton 出現している変数の名前は、下線 “_” で始まっていなければならない

Kima では、検出規則 1 は常に使用されるが、検出規則 2 に関しては、コマンドラインオプションによってユーザが選択的に利用できる。

ヒューリスティクスによる優先度づけや検出規則による検査は、節中の変数の出現を調べるだけで済み、修正案の探索の際に書き換えられた節だけを調べることができる。一方、モードや型の検査には、書き換えた節がプログラムテキスト中の一部であっても、プログラム全体の解析が必要である。ヒューリスティクスや検出規則の詳細および合理性については、文献 [1, 2, 3] を参照されたい。

```
w ← misv(L);  L_1 ← cls(w);
L_0 ← L \ L_1;
for k ← 1 to k_MAX do
  S_k ← {}
end for;
h ← 1;  k ← 1;
while k ≤ k_MAX ∧ h ≤ MAX_h do
  h ← k_MAX − k + h;
  Q_1, ..., Q_h ← high_priority(L_1, h, d_MAX);
  for j ← 1 to h do
    for each q ∈ Q_j do
      if detection_rule2_ok(q) then
        if mcs(L_0 ∪ q) is consistent
          ∧ tcs(L_0 ∪ q) is consistent then
          S_k ← S_k ∪ {q} fi fi
    end for;
    if S_k ≠ {} then k ← k + 1 fi
  end for
end while
```

図 1: 自動修正アルゴリズム (最適化版)

2.2 改良版アルゴリズムの概要

準備としていくつかの記号や関数を定義する。プログラム L を節の集合 $\{l_1, \ldots, l_n\}$ とし、W_L を $W_L = \{(i, v) \mid 1 \leq i \leq n,\ v \in V_{l_i}\}$ と定義する。ここで V_{l_i} は節 l_i の中に現れる変数記号の集合を表す。W_L の要素は節の番号と節中に現れる変数記号のペアである。誤りを含むプログラム L からモード/型制約の矛盾する極小部分集合が求まったとき、これによって指摘される記号出現の中で変数に関するものの集合を $misv(L)$ で表すことにすると、$misv(L)$ は W_L の部分集合となる。モード/型制約は、変数出現の他に、関数 (定数を含む) 記号出現や単一化ゴールなどによっても課されるが、Kima は変数の誤りを修正の対象にしているため、誤りの原因として変数出現だけを考慮する。

$cls(w)$ $(w \subseteq W_L)$ によって w に含まれる節の集合を表す。節の集合 L が与えられたとき、その中に含まれる変数出現を d 個書き換えた節の集合を考える。そのようにして L を元に作り出すことのできるすべての「節の集合」の集合を Q_L^d で表す。Q_L^d の要素をすべて求めるには非常に大きなコストがかかるため、Kima は、優先度に関する反復深化 A* 探索によって、優先度の高い書換えだけを効率的に生成する。L 中の変数出現を最大 d 個書き換えることによって得られる書換え案の中で、優先度の高いものの集合 $\mathcal{Q}_1, \ldots, \mathcal{Q}_h$ を返す関数 $high\_priority(L, h, d)$ を考える。$\mathcal{Q}_1$ がもっとも優先度の高い書換え案の集合である。

このとき、自動修正アルゴリズムの概要は図 1 のようになる。このアルゴリズムは、変数に関する誤りを含んだプログラム L が与えられると、最大 d_{MAX} 個の変数を書き換えることで、優先度を基に分類された修正案の集合 S_k $(1 \leq k \leq k_{MAX})$ を計算する。$k = 1$ がもっとも高い優先度を意味し、k_{MAX} はユーザによって与えられる定数である (デフォルトは 1、すなわちもっとも優先度の高い修正案だけを求める)。図中で、MAX_h は、停止性を保証するための (Kima 側で用意している) 定数である。関数 $mcs(L)$ と $tcs(L)$ は、プログラム L から得られるモードと型制約の集合をそれぞれ表す。関数 $detection\_rule2\_ok(L)$ は、検出規則 2 の使用を指定され、かつ L が検出規則 2 に抵触しない場合に $true$ を返す。

2.3 優先度の高い書換え案の生成

節中に誤りが存在する時、その節のペナルティは悪化することが多いため、$high\_priority(L,h,d)$ は L 中のそれぞれの節に対して優先度の高い書換えを生成し、それらを組み合わせることで、書換え案の集合 $\mathcal{Q}_1,\ldots,\mathcal{Q}_h$ を生成する。

$L=\{l_1,l_2,\ldots,l_n\}$ のとき、節 $l_i\in L$ 中の変数出現を d 個書換えることによって得られる優先度 h の節の集合を $\mathcal{L}^i_{h,d}$ とする。和集合 $\bigcup_{d=0}^{d_{MAX}}\mathcal{L}^i_{1,d}$ は $\bigcup_{d=0}^{d_{MAX}}Q^d_{\{l_i\}}$ の中でもっともペナルティの小さい節の集合であり、{} であってはならない。一方、$h\geq 2$ のとき、$\bigcup_{d=0}^{d_{MAX}}\mathcal{L}^i_{h,d}$ はもっとも小さいものよりも $h-1$ だけ大きなペナルティを持った節の集合であり、{} であってもよい。

このとき、$\mathcal{Q}_i$ は以下のように定義される:

$$\mathcal{Q}_i=\{\{l_1,\ldots,l_n\}\quad|\quad l_1\in\mathcal{L}^1_{h_1,d_1},\ldots,l_n\in\mathcal{L}^n_{h_n,d_n},\ \sum_{j=1}^{n}h_j=n+i-1,\ \sum_{j=1}^{n}d_j\leq d_{MAX}\}$$

ここで、書換え数 0 の節は (書換え前の) 元の節を表す。

$\mathcal{L}^i_{h,d}$ の生成自体も探索問題の一種であり、$\mathcal{L}^i_{h,d}$ 中の要素に書き変わる可能性のないノード (すなわち節) を枝刈りすることによって探索効率を向上できる。具体的には、枝刈りの際の評価基準 (評価関数) として、プライオリティ(ペナルティポイント) と、検出規則 1 への抵触の改善可能性を調べる。1 箇所の変数出現の書換えは、pickup と putdown の操作の組で表現できる。節 l 中の変数出現のうちの 1 つを pickup (無視) した節の集合を U_l とする。$u\in U_l$ に対して、変数出現が pickup されている場所に元とは異なる変数を putdown した節の集合を D_u とする。そして、節 l_i が与えられたとき、優先度 h $(1\leq h\leq h_{MAX})$ と深さ d $(0\leq d\leq d_{MAX})$ に対して $\mathcal{L}^i_{h,d}$ を求めるアルゴリズムを図 2 に示す。

アルゴリズムは、定数 MAX_{pp} よりもペナルティの大きな節を生成しない。簡単のため、MAX_{pp} と図 1 の MAX_h は、同じ値 (デフォルトは 20) に設定されている。$\mathcal{L}^i_{h,d}$ $(1\leq i\leq n)$ 中のそれぞれの節を組み合わせることで、関数 $high\_priority(L,h_{MAX},d_{MAX})$ は $\mathcal{Q}_1,\ldots,\mathcal{Q}_{h_{MAX}}$ を生成する。

ペナルティポイントと検出規則 1 への抵触の改善可能性は、pickup の後と putdown の後に毎回調べられる。関数 $reachable\_mid(l,d,pp)$ と $reachable(l,d,pp)$ は、それぞれ pickup と putdown の後に、現在の深さ d とペナルティ pp に対して節 l が両方の改善可能性を持つ場合に $true$ を返す。関数 $found(l,pp)$ は、節 l が検出規則 1 に抵触せず、かつペナルティ pp を持つとき $true$ を返す。

関数 $num\_violate\_dr11(l)$ と $num\_violate\_dr12(l)$ はそれぞれ、節 l 中で検出規則 1.1 と 1.2 に抵触している (変数出現ではなく) 変数の数を返す。関数 $penalty\_point(l)$ は節 l のペナルティを表す。これら 3 つの関数は節中の変数出現の数を数えるだけなので、D_u の要素だけでなく、U_l の要素に対しても適用可能である。

まず、関数 $found(l,pp)$ は次のように定義される。

$found(l,pp)=$
$num\_violate\_dr11(l)=0\wedge num\_violate\_dr12(l)=0$
$\wedge\, penalty\_point(l)=pp$

図 2 のアルゴリズムは優先度 (ペナルティ) に関する反復深化 A*探索となっており、現在版の Kima では、評価関数 $reachable(l,d,pp)$ と $reachable\_mid(l,d,pp)$ を次のように定義している。

```
h ← 1; pp ← 0;
while h ≤ h_MAX ∧ pp ≤ MAX_pp do
  d ← 0; T ← {};
  if reachable(l_i, d, pp) then
    T ← {l_i};
    if found(l_i, pp) then
      L^i_{h,d} ← {l_i} fi fi;
  for d ← 1 to d_MAX do
    T' ← {}; L^i_{h,d} ← {};
    for each l ∈ T do
      for each u ∈ U_l do
        if reachable_mid(u, d, pp) then
          for each l' ∈ D_u do
            if reachable(l', d, pp) then
              T' ← T' ∪ {l'};
              if found(l', pp) then
                L^i_{h,d} ← L^i_{h,d} ∪ {l'} fi fi
          end for   fi
      end for
    end for
  end for;
  if h > 1 then
    h ← h + 1
  else if ⋃_{d=0}^{d_MAX} L^i_{h,d} ≠ {}
    h ← 2 fi;
  pp ← pp + 1
end while
```

図 2: 優先度の高い節を生成するアルゴリズム

$reachable(l,d,pp)=$
$(num\_violate\_dr11(l)+num\_violate\_dr12(l)\leq d_{rest})$
$\wedge(penalty\_point(l)\leq pp+2\cdot 3\cdot d_{rest})$

$reachable\_mid(u,d,pp)=$
($u\in U_l$ に対し、pickup された変数出現が頭部のもので、かつ $num\_violate\_dr11(u)\geq 1$ の場合)
$(num\_violate\_dr11(u)+num\_violate\_dr12(u)\leq d_{rest}+1)$
$\wedge(penalty\_point(u)\leq pp+2\cdot 3\cdot d_{rest}+3)$
(それ以外の場合)
$(num\_violate\_dr11(u)+num\_violate\_dr12(u)\leq d_{rest})$
$\wedge(penalty\_point(u)\leq pp+2\cdot 3\cdot d_{rest}+3)$

ここで、$d_{rest}=d_{MAX}-d$ である。

これらの定義におけるパラメータは、次の 2 つの理由により設定されている。第 1 に、検出規則 1 への抵触に関して、抵触している変数の数を改善し得る場合は次の 3 つである。(1) 節中の頭部またはガードにおける変数出現の pickup が、$num\_violate\_dr11(l)$ を 1 だけ減少させ得る。(2) ボディにおける pickup が、$num\_violate\_dr12(l)$ を 1 減少させ得る。(3) 頭部における putdown が $num\_violate\_dr11(l)$ を 1 減少させ得る。したがって、深さ d の探索 (d 個の書換え) は、$num\_violate\_dr11(l)$ と $num\_violate\_dr12(l)$ の合計を最大 d だけ改善できる可能性がある。

第 2 に、ペナルティに関して、pickup と putdown は両者とも、節へのペナルティを 3 だけ減少させ得る (表 1)。1 つの変数出現はヒューリスティクス (ii) と (iii)、あるいは (iii) と (iv) の状況に同時に陥る可能性があるが、その場合でも課せられるペナルティは最大 3 である (またそのようにペナルティ

表 2: N 箇所の誤りの修正にかかる Kima の平均応答時間

プログラム	節の数	節中の変数出現数の平均	節中の変数の種類数の平均	N	検出に成功した事例	解析を中止した事例	提示された修正案の数	応答時間 (秒)
nqueen	34	10.4	5.12	1	96	0	1.10	0.439
				2	96	1	2.17	19.1
gen_test	77	11.6	5.84	1	99	0	1.67	2.04
				2	83	3	5.74	16.5
tgraph	79	17.6	8.84	1	100	0	1.21	10.9
				2	100	37	2.35	60.4
graph	155	16.8	8.46	1	98	1	1.15	21.7
				2	97	39	2.28	87.4

の値が定められている)。したがって、深さ d の探索はペナルティを最大 $2 \cdot 3 \cdot d$ だけ改善できる可能性がある。

3. 実験: Kima の性能

実験には次の 4 つのサンプルプログラムを用いた。並列版 N-queen プログラム nqueen.kl1 (67 行)、Kima のモジュールの 1 つで、書換え案の生成を行なうモジュールと検査を行なうモジュールの仲介を行なう gen_test.kl1 (200 行)、型制約のソルバ tgraph.kl1 (218 行)、そしてモード制約のソルバ graph.kl1 (390 行) である。

これらのサンプルプログラムに対して、同一節中に 1 箇所および 2 箇所の変数出現に関する誤りをランダムに挿入した例をそれぞれ 100 例ずつ生成し、深さ 1 および 2 の探索を行なって、もっとも優先度の高い修正案だけを求めた。ただし、変数名を "_" と書き間違えるような誤りについては、人間の犯す誤りとしてほとんどないことから考慮していない。このとき、最終的に求まった修正案数と応答時間の平均を表 2 に示す。表の値は、修正に成功したものに関する平均値である。誤り検出には、モードおよび型情報のほか、検出規則 1, 2 を用いた。実験環境としては、Linux 2.4.3 + PC/AT 互換機 (PentiumIII 733 MHz + 768 MB memory) を用いた。Kima のモジュールは、他のモジュールの述語を 5 つ程度呼び出しているが、呼び出されている側の情報は用いていない。しかし、修正の品質そのものに大きな影響はなかった。

表中の "解析を中止した事例" は、誤りの検出には成功したが、メモリスワップの発生、あるいは修正に 20 分以上の時間がかかるために、修正を中止した事例の数を表す。"提示された修正案の数" や "応答時間" は、解析を中止した事例を除いて求めた値である。

大多数の場合、Kima によって提示される修正案の数は 1 つかごく少数である。2 箇所の誤りに対する修正では、プログラムの等価性 [1, 2, 3] のために、必ずしも深さ 2 の探索が必要となるわけではない。`gen_test` の $N = 2$ の場合の誤り検出例が他と比較して少ないのは、これが原因である。プログラム中の変数の書換えは、元のプログラムと等価なプログラムを作り出すことがある。特に `gen_test` には、最大 4 つしか変数出現を持たない節が 10 以上も存在しており、節中の 2 箇所の変数の書換えが、等価なプログラムを作り出す可能性が高いと。また、修正の際の余分な書換えが、修正案の数を増大させてしまう場合がある。`gen_test` の $N = 2$ の場合では、節中の 25 の変数出現のうちの 2 つの書き間違えに対して、184 の修正案がたまたま見つかった事例があった。この事例を除けば、該当の "提示された修正案の数" は 5.74 から 3.44 となる。

4. 考察と今後の課題

"解析を中止した事例" において、計算量の増大から解析が不可能となった原因は、モード/型制約の矛盾する極小集合が誤りの候補として指し示す節の数の増大である。求まる極小集合のサイズは、節中の変数出現数と相関があるが、深さ 2 の探索の場合、10 から 12 以上の節が極小集合によって指摘されると (この実験の基準では) 解析不能となることがわかった。

しかしながら、実際には、(単一の) 誤りに対して、候補となる節を指し示すのは、1 つの極小集合ではなく、グループ化された複数の極小集合である [1, 2, 3]。この実験の例では、多くの場合、サイズの大きな極小集合と同時に、7 以下の節を指し示すサイズの小さな極小集合が発見されていた。現在版の Kima では、グループ内の極小集合が指し示す節すべてを修正の対象とみなすが、複数の極小集合が指し示す節の共通部分をより怪しい節とみなすことで、この実験の範囲内における誤りは修正可能となる。誤り箇所の候補が増大したときに、どのようにして合理的に誤りを特定するかは、今後の課題である。

参考文献

[1] 網代育大, 上田和紀, Kima: 並行論理プログラム自動修正系. コンピュータソフトウェア別冊ソフトウェア発展, Vol. 18, No. 0, 2001, pp. 122–137.

[2] Ajiro, Y., Ueda, K., Kima: an Automated Error Correction System for Concurrent Logic Programs. *Automated Software Engineering*, Vol. 9, No. 1, Kluwer Academic Publishers, 2002, pp. 67–94.

[3] 網代育大, 制約充足に基づく静的解析手法を用いたプログラミングの自動化に関する研究, 早稲田大学博士論文, 2002.

[4] Cho, K. and Ueda, K., Diagnosing Non-Well-Moded Concurrent Logic Programs. In *Proc. 1996 Joint Int. Conf. and Symp. on Logic Programming (JICSLP'96)*, The MIT Press, 1996, pp. 215–229.

[5] 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.