並行論理プログラム静的解析系 Kima の実装

4N−4

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

並行論理プログラミングにおいて、強モード／型体系の下で、プログラム中の簡単な誤りを自動的に修正する、制約ベースの静的解析系 Kima を実装した。Kima は、KL1 プログラムの変数や定数の少数個の書き誤りを静的に検出・修正することができる。

強い型体系や強いモード体系を備えた言語では，型やモードを静的に検査ないしは推論することによって、それら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。

ML などの関数型言語における型解析と同様に、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、ユーザによって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 [3]。

並行論理プログラムに誤りが存在する場合、そこから得られるモード制約の集合が充足不可能になることが極めて多く、その制約集合から矛盾する極小部分集合を探索することによって、プログラムの誤りを局所的に特定することが可能である [2]。

プログラムは一般に、数多くの条件分岐をもち、また、同一の手続きが複数回呼ばれることも多い。このような場合、モードや型に関する制約集合は冗長性をもつことになる。Kima は、この冗長性を利用することで、たとえ誤ったプログラムであっても、モードや型などの抽象情報に関する正しい仕様を推定し、発見した軽微な誤りを自動的に修正することができる。

Kima では、先ほど述べたモード制約の矛盾する極小部分集合を利用することで、修正案の探索空間及び計算時間の増加を抑え、さらにデータ型情報や、いくつかのヒューリスティクスを併用して、修正案の品質を向上させている。

Kima — an Automated Error Correction System for Concurrent Logic Programs.

†Yasuhiro AJIRO, ‡Kazunori UEDA

†Graduate School of Science and Engineering, Waseda University

‡School of Science and Engineering, Waseda University

2 並行論理型言語における強モード / 型体系

並行論理プログラミングでは、モード体系が型体系よりも基本的な役割を演じる。ここでは Moded Flat GHC を例にとり、モード体系の大まかな概要を述べる。

Moded Flat GHC のモード体系の目的は、ゴールのふるまいを定義する述語の引数に、極性構造 (データ構造の各部の情報の流れの向きを定めたもの) を与えることであり、モードとはこの極性構造のことである。

プログラムは、h :- G | B (h は原子論理式、G および B は原子論理式のマルチ集合) の形の節の集合である。これが課するモード制約の制約規則や、解析に必要な計算量の詳細などについては [3] を参照してほしい。

型体系については、関数記号 (定数を含む) の集合 *Func* を、重なりをもたないいくつかの集合 $F_1, \ldots, F_n$ に分類することによって定式化でき、モード解析と同様の方法で計算できる。

3 Kima の動作

例として、変数を 1 箇所書き間違えている append プログラムを考える。

R_1 : append([], Y,Z) :- true | Y=Z.

R_2 : append([A|Y],Y,Z0) :- true |
Z0=[A|Z], append(X,Y,Z).

(頭部は append([A|X],Y,Z0) が正しい)

これに対し Kima は、節 R_2 に関する次のような 6 つの修正案を発見して、提示する。このうち、Priority 1 の 2 つ目の修正案が意図通りの修正である。また、同じ Priority の 1 つ目の修正案は、2 つのリストを交互にマージするプログラムになっている [1]。

```
========== Priority 1 ==========
append([A|Y],X,Z0):-true|Z0=[A|Z],append(X,Y,Z).
append([A|X],Y,Z0):-true|Z0=[A|Z],append(X,Y,Z).
========== Priority 2 ==========
append([A|Y],Y,Z0):-true|Z0=[A|Z],append(Z,Y,Z).
append([A|Y],Y,Z0):-true|Z0=[A|Z],append(Z0,Y,Z).
append([A|Y],Y,Z0):-true|Z0=[A|Z],append(Y,Y,Z).
========== Priority 3 ==========
append([A|Y],Y,Z0):-true|Z0=[A|Z],append(A,Y,Z)
```

4 自動修正アルゴリズム

Kima に実装されている自動修正アルゴリズムは、次のようなものである。

表 1: 変数の 1 箇所の書き間違いに対する修正案の数 (ヒューリスティクスあり vs. なし)

プログラム	ヒューリスティクス	実験の総数	検出されたもの	求まった修正案の数 1	2	3	4	5	6	7	≥8
append	あり	58	36	20	13	3	0	0	0	0	0
	なし	−	−	1	3	7	3	8	4	3	7
fibonacci	あり	104	51	42	6	3	0	0	0	0	0
	なし	−	−	25	11	3	1	8	1	0	2
quicksort	あり	279	169	116	50	1	2	0	0	0	0
	なし	−	−	65	69	2	20	1	2	8	2

```
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;
depth ← 1;
while 解が見つかっていない do
  while depth 個の記号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 well-moded かつ
      well-typed になった
    then その書換えを修正解として出力
  end while;
  depth ← depth + 1
end while;
```

このアルゴリズムは、モード制約の矛盾する極小部分集合から疑わしい記号を抽出する誤り同定部分と、それらをもとに修正案を探索する部分の 2 つに大別することができる。また修正案の探索は、書換え個数に関する深さ漸増探索であり、*depth* がその探索の深さを表している。

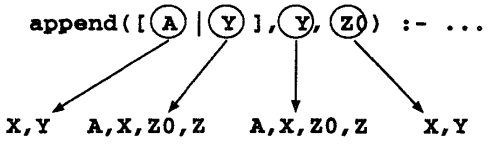
5 プログラム中の誤りの同定

並行論理プログラムに誤りが存在する場合、そのプログラムは通信プロトコルの一貫性を失い、モード制約集合が充足不可能 (ill-moded) となることが多い。このとき、モード制約集合の矛盾する極小部分集合を求めることで、矛盾の原因となっているプログラム中の節と記号を絞り込むことが可能である [2]。

モード制約は、プログラム中の記号出現と「記号出現 ⇒ モード制約」の形のモード規則によって課されるため、この極小部分集合に含まれるモード制約を課した記号出現を、誤りの原因として特定することができる。

6 修正案の探索

第 3 節の例の場合、矛盾する極小部分集合を求めることで、節 R_2 の変数記号 X と Y が怪しい変数として特定される。このとき、深さ (書換え個数) 1 の修正案の探索は次のように行われる [1]。



7 修正案への優先度の付加

Kima はさらに、求めた修正案に対し

i) モード制約の強さの総和が弱い方がもっともらしい。

ii) リストの car と、もとのリストのデータ型が単一化されているものは、誤りである可能性が高い。

というヒューリスティクスを使って、優先度をつける。ii は 3 節の例の `Priority 3` のような修正案を排除するのに利用できる。表 1は、このヒューリスティクスを使って、最も優先度の高いものだけを修正解として求めた場合と、ヒューリスティクスを利用しなかった場合の、提示される修正案の数の比較である。

8 今後の課題

現在 Kima では、誤りの検出にはモード情報だけを利用しているが、型情報を併用して誤りの検出率を向上させ、さらに修正案の探索空間を縮小することが今後の課題である。

参考文献

[1] Ajiro, Y., Ueda, 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.

[2] 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.

[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.