公聴会発表資料

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

網代 育大

早稲田大学大学院 理工学研究科
情報科学専攻 上田研究室

ajiro@ueda.info.waseda.ac.jp

概要

1. はじめに
2. 並行論理型言語の簡単な解説
3. 並行論理プログラム自動修正系 Kima
   - デモ、動作原理、実験結果
4. ヒューリスティクスによる修正品質の向上
5. 修正案探索の最適化技術
6. まとめ

研究の背景と動機

- 強型体系 (strong typing) はデータ型の一貫性を保証し、多くの簡単な誤りを静的に検出することができる
- 並行論理プログラミングにおける強モード体系 (strong moding) はデータフローの一貫性を保証
- CPU 性能の向上によって、個人の所有する計算機であっても、計算能力を持て余している

- 余剰計算能力と、これまで「保証に」使われていた型の体系を「攻撃的に」利用することにより、プログラムの正しい形状を計算機に予測、提示させることを目指す

研究の成果

- プログラムに関する明示的な仕様や宣言を与えることなく KL1 プログラムの軽微な誤り (変数の少数個の書き誤り) を自動的に修正するシステム Kima を開発した
- 「もっともらしい」プログラムが満たす構文上の制約があることを突き止め、ヒューリスティクスとして示した
- 修正案の探索の最適化技術を開発して実装した
  - 「もっともらしさ」に関する深さ漸増 A* 探索
- 実験により、変数の 1 箇所の誤りに対してはその 90% 以上を自動的に修正できることを示した (2 箇所以上の誤りに対してはそれ以上)

近い将来: 自動プログラミングへの応用

入出力例を与えることによって、変数の記述を自動化

```
:- instance qsort(+[5,2,7,3],-[2,3,5,7],+[]).
qsort/3 :- X=[]    |=/2.
qsort/3 :- X=[X|X] |
  part/4,qsort/3,X=[X|X],qsort/3.
part(_,[],    S, L ):- true | S=[],L=[].
part(A,[X|Xs],S0,L ):- A>=X | S0=[X|S],part(A,Xs,S,L).
part(A,[X|Xs],S, L0):- A< X | L0=[X|L],part(A,Xs,S,L).
```

⇓ メタインタプリタの援用

```
qsort([],    Ys0,Ys ):- true | Ys=Ys0.
qsort([X|Xs],Ys0,Ys3):- true |
  part(X,Xs,S,L),qsort(S,Ys0,Ys1),Ys1=[X|Ys2],qsort(L,Ys2,Ys3).
```

研究の特徴

並行論理型言語と強モード/型体系を用いた。モード/型解析は制約充足の一般的な枠組に基づいており、以下の利点がある:

- 静的な解析が可能
- プログラムの一部に対して適用可能
  - 大規模なプログラムであっても分割解析が可能
  - 未完成のプログラム（述語定義単位）に対して適用可能
- プログラムの変更や追加、変形に対する解析を制約に対する操作として数学的に扱うことができる
  - プログラム中の誤りによってモード/型制約集合に矛盾が発生したとき、矛盾する極小の部分集合を計算することで、誤り箇所を局所的に特定することができる

関連技術

- **仕様記述** (e.g., Z, CafeOBJ)
- **抽象解釈**
- **型解析**
  - Strong typing (e.g., ML, Haskell) — **静的、厳格**
  - Soft typing (e.g., Lisp, Scheme) — **動的、インタラクティブ、柔軟**
- **入力予測 UI** (e.g., Web browser, PO BOX, T9)
- **遺伝的プログラミング**
- **帰納論理プログラミング** (programming by examples)

並行論理プログラミング

並行論理型言語の特徴

- セマンティクスのレベルで並行性を扱っていることにより、並行・並列・分散計算の記述に適している。
- 論理型言語 (Prolog) からバックトラックを除いた言語。
- 計算は (並列計算/論理プログラミングにおける) リダクションによって進む。
- プロセス間通信は、論理変数を通じて暗黙のうちに行なわれる。

KL1 プログラムは以下のような節の集合：

```
p(X,Z) :- true | q(X,Y)@node(0), r(Y,Z)@node(1).
```

ここでデモ

1. Append プログラムの1箇所(深さ1)の誤り
2. 組合せプログラムの独立した2箇所(深さ1)の誤り
3. クイックソートプログラムの2箇所(深さ2)の誤り
4. N-queen プログラムの2箇所(深さ2)の誤り

Kima の動作原理

基本アルゴリズム

1. モード/型制約の矛盾する極小部分集合の計算による、誤り箇所候補の特定
2. 矛盾する極小部分集合のグループ化による誤りの分類
3. 以下の手続きの繰返しによる generate-and-test
   (a) 特定した場所の周辺の機械的な書換え
   (b) 書換えたプログラムのモード/型の再計算

- 計算量 = 可能な書換え × モード(型)解析の手間
- モード(型)解析の手間はプログラムサイズに比例

実験1: 1箇所の誤りに対する修正

Program	Level	Prioritizing	Total cases	Detected	Proposed Alternatives 1	2	3	4	5	6	≥7
append	0	no	58	36	1	3	8	3	6	5	10
	0	yes	58	36	27	9	0	0	0	0	0
	2	yes	58	58	39	19	0	0	0	0	0
fibonacci	0	no	118	72	18	13	2	15	9	0	15
	0	yes	118	72	54	11	1	6	0	0	0
	2	yes	118	99	71	18	8	0	2	0	0
quicksort	0	no	300	221	49	76	8	59	0	9	20
	0	yes	300	221	164	41	16	0	0	0	0
	2	yes	300	286	199	84	2	1	0	0	0

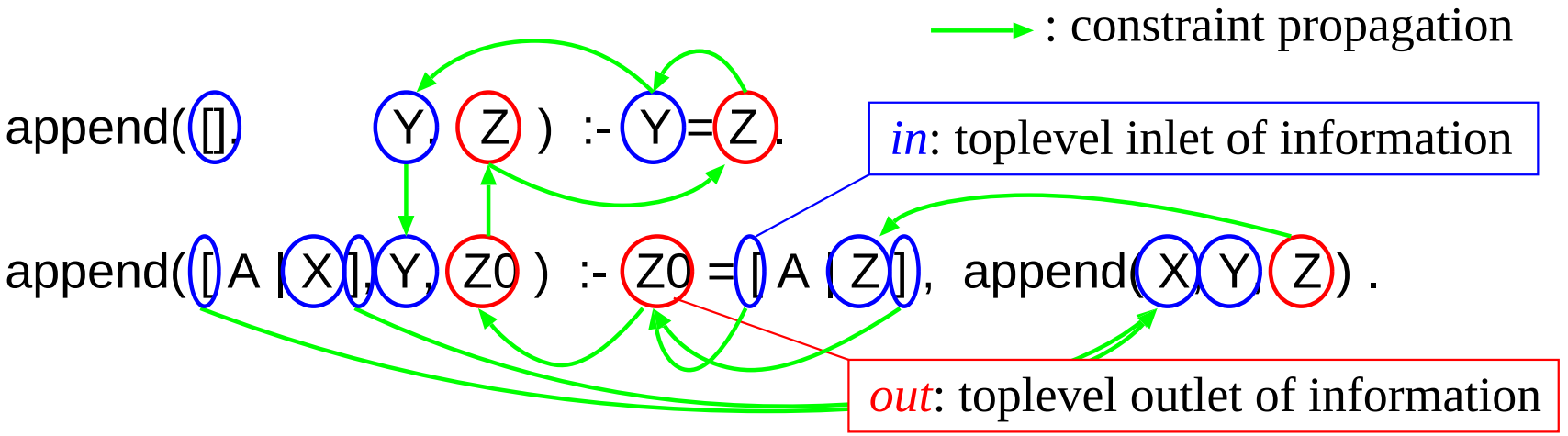
- 平均 90% (443/476) 以上の誤りを検出および修正可能
- 提示される修正案は1つかごく少数

実験2: 同一節における 2, 3 箇所の誤りの検出率

Program	N	Level	Total cases	Detected cases	Detection rate (%)
append	2	0	1200	937	78.1
	2	2	1200	1141	95.1
	3	0	16980	14597	86.0
	3	2	16980	16674	98.2
fibonacci	2	0	4668	3982	85.3
	2	2	4668	4489	96.2
	3	0	133045	125300	94.2
	3	2	133045	131810	99.1
quicksort	2	0	12102	11263	93.1
	2	2	12102	12005	99.2
	3	0	337455	330769	98.0
	3	2	337455	336943	99.8

- 検出規則の併用により、95% 以上の誤りを検出可能

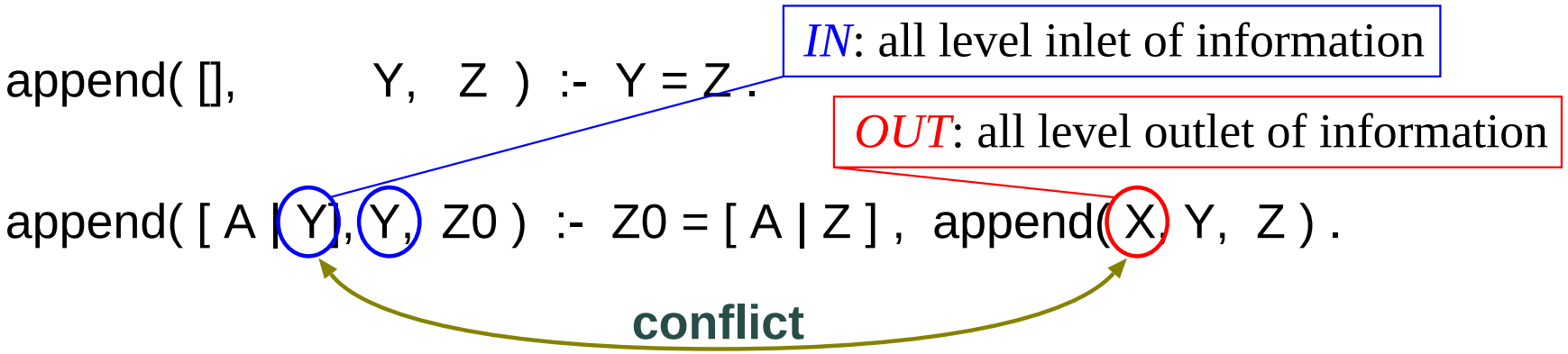
モード解析による情報の流れの解析



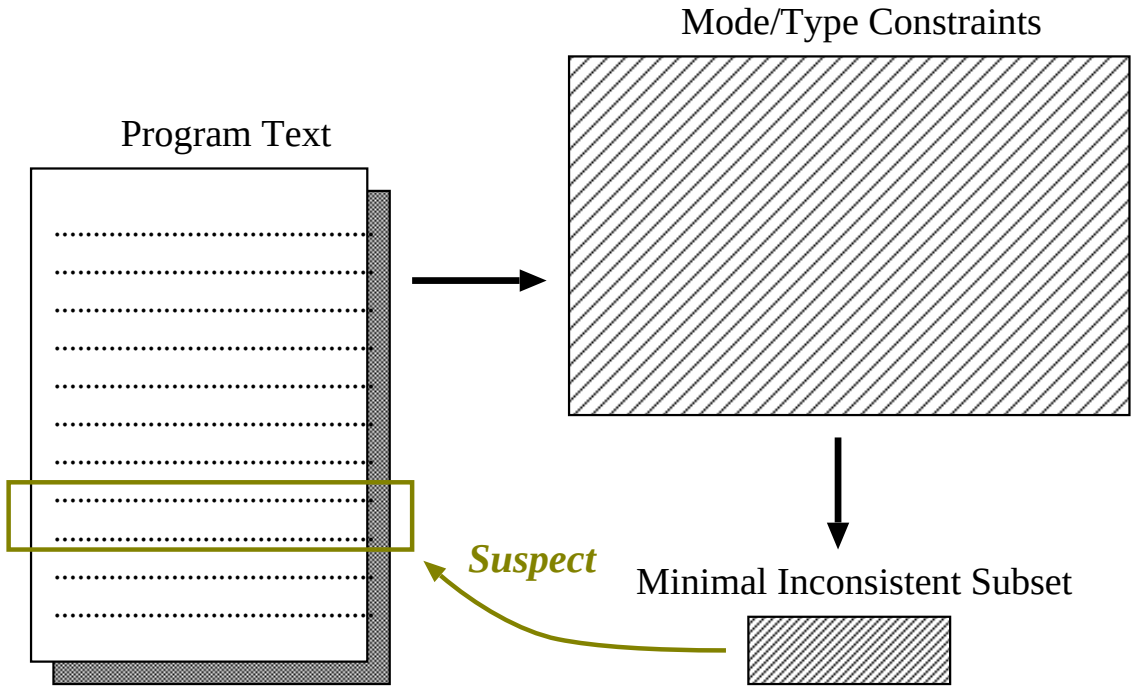
- モード解析は多数の簡単なモード制約の制約充足問題である。
- モード制約は
  - プログラムテキスト中の記号出現と
  - モードづけ規則によって課される。
- データ型に関しても同様の方法で解析可能

モード解析による誤りの検出

- 正しいプログラムはモード/型制約の集合が充足可能 (well-moded/typed) である。
- 誤りを含んだプログラムは、制約集合が充足不可能 (non-well-moded/typed) となることが多い。



矛盾する極小部分集合による誤り箇所の特定



- 矛盾する極小部分集合は間違った制約を含む。
- 極小部分集合サイズの増加は、プログラムサイズの増加に比べて小さい（多くの場合、集合の要素数は 10 以下)。

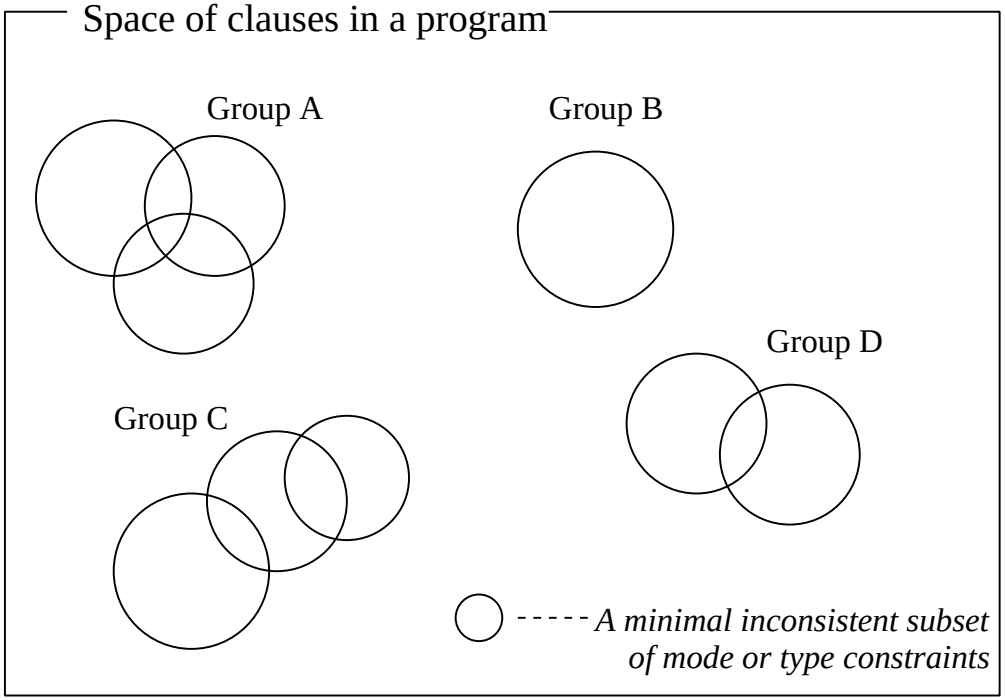
矛盾する極小部分集合を求めるアルゴリズム

$\{c_1, \ldots, c_n\}$: a multiset of constraints (input),
S: a minimal inconsistent subset (output).

$c_{n+1} \leftarrow \mathit{false}$;
$S \leftarrow \{\}$;
while S is consistent **do**
  $D \leftarrow S$; $i \leftarrow 0$;
  while D is consistent **do**
    $i \leftarrow i + 1$; $D \leftarrow D \cup \{c_i\}$
  end while;
  $S \leftarrow S \cup \{c_i\}$
end while;
if $i = n + 1$ **then** $S \leftarrow \{\}$ **fi**

- **計算量はプログラムサイズに対してほぼ線形オーダー。**

誤り箇所候補のグループ化



- 同一の節を指す極小集合を同一グループに分類
- 修正案の探索はそれぞれのグループに対して独立に行う

修正案探索の最適化

Kima に実装されている2種類の最適化技術：

1. 「プログラムのもっともらしさ（優先度）」に関する深さ漸増 A* 探索によって、もっともらしい書換え案だけを効率的に生成
2. プログラムテキストにおいて、矛盾が発生している場所の周辺だけを最初に解析 (局所的な解析) することで、検査の際に矛盾を早く検出

2種類の最適化を利用したアルゴリズム

最適化版アルゴリズム

1. モード/型制約の矛盾する極小部分集合の計算による、誤り箇所候補の特定
2. 以下の手続きの繰返しによる generate-and-test
   - (a) 特定した場所の周辺の機械的な書換えの中で優先度の高いものだけを生成
   - (b') モード/型の局所的な検査
   - (b) プログラム全体のモード/型の検査

節中の変数の出現の仕方に基づく最適化

優先度づけ Kima は節中における変数の出現の仕方によって修正案に優先度をつける。

- 変数は一対一通信に使われるのがもっともらしい。
- もっともらしい 2回 $>$ 3回以上 $\gg$ 1回もっともらしくない

検出規則 Kima は節中に1回しか現れない変数の変数名が下線 “_” で始まっていないとき、これを誤りとして検出できる。

修正案探索の際、書き換えていない節に関しては優先度づけや検出規則による再検査を行なう必要がない。

モード/型の再計算には、プログラム全体の解析が必要。

実験3: 優先度の高いプログラムの分布

Program	N	Total cases	Plausible programs	With caller info	Related to guard
append	1	58	0	0	0
	2	1200	7	1	0
	3	16980	14	3	0
	4	167842	29	5	0
fibonacci	1	118	11	9	4
	2	4668	66	3	3
	3	133045	309	5	5
quicksort	1	300	9	4	4
	2	12102	33	17	4
	3	337455	76	48	16

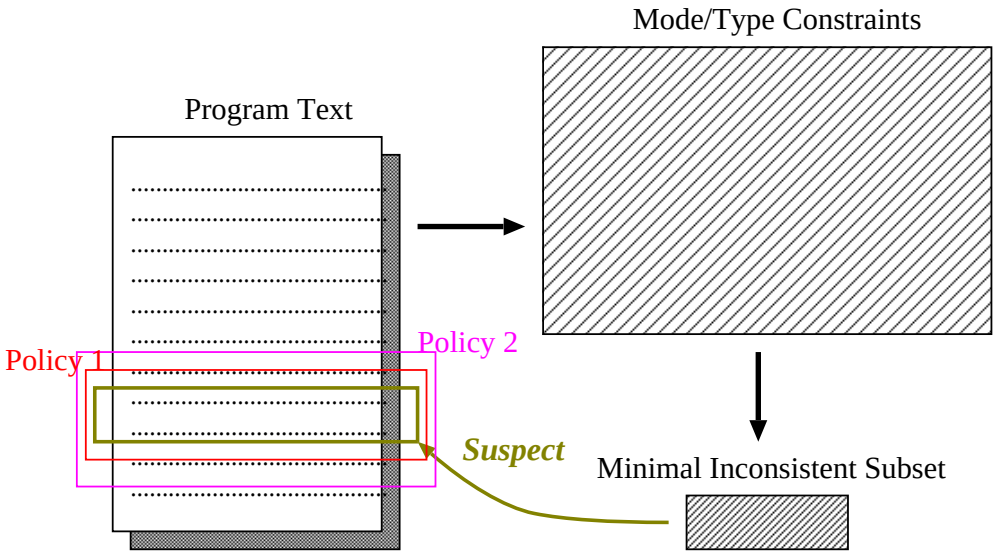
- オリジナルの正しいプログラムの周辺に存在するもっともらしいプログラムの数は極めて少ない。

局所的な解析による最適化

プログラム全体のモード/型解析の前に、矛盾する極小部分集合によって指摘された節の周辺だけを解析する。

方針1 極小部分集合によって指摘された節を含む述語 (第一述語と呼ぶ) の述語定義節

方針2 第一述語に加えて、第一述語を呼び出している述語と第一述語が呼び出している述語の述語定義節



実験4: 最適化による探索空間の縮小効果

プログラム	節の数	節中の変数出現数の平均	深さ	すべての書換え案の数	優先度の高い書換え案の数	局所的な検査に通った書換え案の数 方針1	局所的な検査に通った書換え案の数 方針2	最終検査に通った書換え案の数
nqueen	34	10.4	1	36.8	2.55	2.05	1.76	1.43
(67行)			2	8069	199	84.8	78.2	6.48
gen_test	73	10.8	1	66.9	2.45	2.01	1.89	1.61
(183行)			2	26700	21.5	8.78	7.90	4.25
tgraph	79	17.6	1	53.2	2.59	1.78	1.74	1.39
(218行)			2	16900	8.10	4.70	4.70	3.40
graph	155	16.8	1	86.0	2.54	1.80	1.77	1.35
(390行)			2	44000	9.40	4.20	3.90	3.10

- 優先度づけと検出規則によって、書換え案を大幅に絞り込める。
- 局所的な解析によって、最終的に求まる修正案に近いところまで、書換え案をさらに絞り込める。
- 方針1と2では効果にあまり差がない。

まとめ

- Kima の枠組は自動プログラミングのための要素技術として有望である。
- Kima はそれ自身 KL1 で記述されており、現在の大きさは約 5,200 行。
- Kima は以下のサイトからフリーソフトとして利用可能

  http://www.ueda.info.waseda.ac.jp/~ajiro/study-j.html