実用的な型エラースライサーの実装と評価

理学専攻・情報科学コース　脇川奈穂　(指導教員：浅井 健一)

1　はじめに

本学では、主に情報科学科の 2 年を対象とする授業「関数型言語」が開講されている。この授業の目的は OCaml の習得であり、受講者は学習にデバッグ支援システム**型エラーデバッガ** [2] を利用している。

型エラーデバッガでは、対話的に型の意図を尋ねる質問を繰り返すことによって、プログラム作成者の意図に合わせた型エラーの原因を特定している [1]。型が付いている式が質問の対象となり得るが、それら全てが型エラーの原因になるとは限らず、プログラムの規模が大きくなるほど不必要な質問数が多くなっていた。

この問題点を解決するべく、我々は型エラーデバッガと同様に OCaml の型推論器を用いて**型エラースライサー** [3] を実装した。具体的には、あらかじめ不要な式の型や構文の抽象化によって型エラースライスを求めて、質問の対象となる式を削減した。本研究では、型エラースライサーの実用性の向上を目的に、高速化を重視したアルゴリズムの改良および実証実験による評価を行った。

2　型エラースライサー

実用的なアルゴリズムは以下の通りである。プログラム中の式が木構造のデータを持ち、単純なアルゴリズムでは抽象化できる式の数が 1 つのみだったが、複数のノードを同時に抽象化できるように改良した。

1. ノード数が式 S 全体の半数程度になる部分木 s（**ピボットに相当**）を選ぶ。
2. S を含まないノードを全て抽象化したプログラム $P_2(s)$ を対象に型推論を実行する。

   `illtyped:` s の型エラースライス s' を求めて、$P_2(s')$ を返す。

   `welltyped:` 次に進む
   (抽象化できるノードがなかった場合も含む)
3. Sのみを抽象化したプログラム $P_1(s)$ を対象に型推論を実行する

   `illtyped:` $P_1(s)$ の型エラースライスを求める。

   `welltyped:` s の型エラースライス s′を求めて、$P_1(s')$ の型エラースライスを求める。

例として、以下のプログラムの型エラースライスが求められるまでの過程を説明する。

```
# let pow x n = x ^ n in pow (1 + 2) 3
```

ピボット（下線部分）毎のプログラムの状態は

```
# let pow x n = x ^ n in pow (1 + 2) 3
# let pow x n = x ^ n in pow □ 3
                    ⋮
```

となり、最終的に以下の状態が得られる。

```
# let pow x n = □ ^ n in pow □ 3
```

3　実験方法

3.1　実験データ

本研究では、型エラースライサーの実行時間や型推論の回数を記録する。本研究では、個々の型推論にかかった時間も計測する。さらに、高速化の手掛かりとして、実行の対象となったプログラム中の式の構造に関するデータの採集も行った。

実験によって得られるデータを表 1 に示す。上 4 行は型エラースライサーのアルゴリズムの中で採取され、下 3 行は OCaml のプログラムを対象に、アルゴリズムが呼び出される前に採取される。

表 1: 実験データ

項目	種類	補足
root_size	整数	式全体のノード数
total_time	実数	型エラースライサーの実行時間
infer_num	整数	型推論の回数
infer_time	実数	1 回の型推論にかかる時間
size	整数	式のノード数
depth	整数	式の深さ
type	文字列	型の種類

実験には、規則性がある複数のプログラムを用いる。それらは、下記のいずれかに該当する。

- 算術演算を中心に構成された式 (arith)

  `# 1 + ... + 2.0 + ... + 1`
- 比較演算子を用いた式 (comp)

  `# 1 + ... + 1 = 2.0 +. ... +. 2.0`
- リスト構造の式 (list)

  `# 1 :: ... :: 2.0 :: ...`

3.2　分析と評価

データ分析によって実験データのグラフを作成し、それをもとに実験結果の評価を行った。いずれの場合にも、プログラムに関するデータ *size*、*depth*、*type* を記録し、必要に応じて分析に用いる。プログラム arith, comp, list では、*root_size* 毎に他項目がどのように変化するか確認し、提案手法の妥当性や効果について考察した。

4　実験結果

4.1　アルゴリズムに対する分析

まず、型エラースライサーの実行に関するデータを分析した。*root_size* 毎の *infer_num*, *infer_avg*, *total_time* の相関およびばらつきを表すグラフを図 1、図 2、図 3 に示す。

ノード数 n のプログラムを型エラースライサーで実行すると、*infer_num* は $O(\log n)$、*infer_avg* は $O(n)$

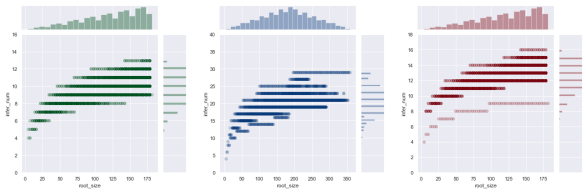

図 1: infer_num per root_size

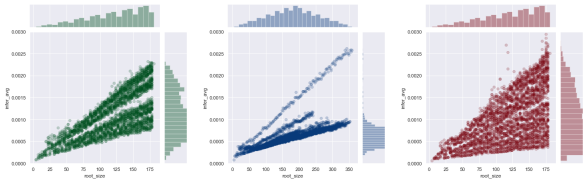

図 2: infer_avg per root_size

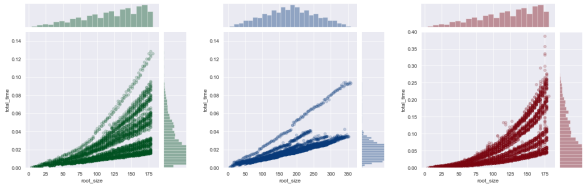

図 3: total_time per root_size

に比例すると考えられる。*total_time* はアルゴリズムの計算量 $O(n \log n)$ と予想し、期待と異なる結果が得られる場合はあったが、単純なアルゴリズムより十分に実用的である。また、プログラム毎の傾向としては、comp や list のように多相型を含む場合に極端に時間がかかったり、ばらつきが大きかったりすることがわかった。arith は全ての式が単相型だが、comp や list には多相型の式を含む。list が他のプログラムに比べて実行時間が長くなっており、実際に「関数型言語」で要素数が非常に多いリスト構造にて、型エラースライサーの実行時間が数時間かかるケースがある。

実験によって、リスト構造を含むプログラムでは、1 回あたりの型推論にかかる時間はあまり変わらないが、型推論の回数が減ることによる効果が大いに期待できることがわかった。

4.2 アルゴリズムの比較

単純なアルゴリズムと実用的なアルゴリズムでは、型推論にかかる時間にあまり差は見られなかった。しかし、図 4 からわかるように、型推論の回数が $O(n)$ から $O(\log n)$ に減少したため、プログラムが大きくなるほど効果を期待できる。

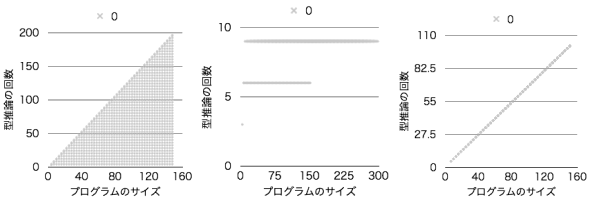

図 4: 単純なアルゴリズムにおける型推論の回数

4.3 プログラムに対する分析

次に、プログラムの構造を分析する、演算子以外は同様の構造を持っており、構文として定数、関数適用、演算子、リストの構成子を含む。各構文の数や割合は図 5 のようになった。横軸はいずれもプログラムの規模に対応しており、右に行くほどノード数が多くなる。グラフの縦軸は、構文の数を意味する。

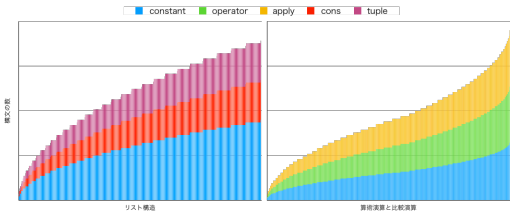

図 5: プログラム中の式の構文に関するデータ

arith と comp は構文の割合がほぼ同じだった。さらに、ノードの偏りについて分析したところ、arith と list の状態が近く、特定のノードのみ大きくなっていることがわかった。

arith : 閾値

arith の infer_avg は、型エラーの原因となるノードの位置によって値が変化し、当該ノードが頂点に近いほど高速になる。おおよそ全体の半分程度の深さが境界となっていた。型推論の時間はサイズに比例しており、同一サイズでは型エラーの原因となるノードの位置の影響を受ける。

comp : 外れ値

comp の infer_avg は、時々極端に遅い場合があった。そのようなプログラムの規則性を調べたところ、比較演算の右辺と左辺のサイズがほぼ同じになっていることがわかった。この事象の原因は多相型にある。関数部分が抽象化されることによって制約が減り、各々の引数の推論が必要になるからである。

list : 構文の抽象化

list は実験の対象としたプログラムの中で最も時間がかかり、total_time の計算量が単純なアルゴリズムとほぼ同じになっている。型推論以外にかかる時間は長いが、型エラーデバッガの質問の回数は減少する。抽象化の回数が多いことが影響していると考えられる。

5 まとめ

実証実験によって、型エラースライサーの計算量ががわかった。ノードを 1 つずつ抽象化する単純なアルゴリズムに比べて、実用的なアルゴリズムはかなり短い時間で結果が得られるようになったため、十分に実用的な効果が得られたといえる。

一方で、多相型を含むと型推論に時間がかかることがわかった。この点を考慮できれば、更なる高速化の余地があると考えられる。

参考文献

[1] E. Y. Shapiro. *Algorithmic Program Debugging*. Cambridge: MIT Press, 1983.

[2] 対馬かなえ, 浅井健一. コンパイラの型推論を利用した型デバッグの手法の提案. コンピュータソフトウェア, Vol. 30, No. 1, pp. 180–186, 2013.

[3] 脇川奈穂, 対馬かなえ. 実用的な型エラースライサーの提案と評価. In PPL 2018 : 第 20 回プログラミングおよびプログラミング言語ワークショップ, 2018.