5L-1

Ruby向け動的コンパイラに対するクラスの推定手法の実装と評価

小泉　有輝[†]　　千葉　雄司[‡]　　久保田 光一[§]

中央大学大学院 理工学研究科 情報工学専攻[†‡§]

要約: 本論文ではクラス推定の精度を改善する手法として, 呼出しに成功したメソッドの名前を利用する方法を提案する. 提案手法では, レシーバの取りうるクラスの集合を限定するために, レシーバがメソッドの呼出しに成功するごとに, レシーバが取りうるクラスの集合を縮小することで推定の精度を改善した. 提案手法の効果を評価するために, Ruby 向け動的コンパイラに提案手法を適用し, Ruby Benchmark Suite のマクロベンチマークを実行して脱仮想化を適用できる確率の変化を調査した. その結果, 適用可能な確率が最大で 38.1%, 相乗平均で 6.06% 増加することがわかった.

キーワード: クラス推定, Ruby, オブジェクト指向プログラミング言語

1　はじめに

本論文では, クラス推定の精度を改善する手段として, 呼出しに成功したメソッドの名前を利用する方法を提案する. 提案技法の適用対象は Ruby など動的な型付けをおこなうオブジェクト指向プログラミング言語である. 提案手法を Ruby 向けの動的コンパイラ上に実装し, その有効性を評価した.

2　関連研究

クラスの推定手法には, データフロー解析による推定手法やクラス階層を解析することによる推定手法, プロファイル情報を用いてクラスを予測する手法がある [2][3][4].

データフロー解析による従来の推定手法は, 処理中のメソッドのプログラム中に含まれるリテラル, 現在のレシーバを表す変数（C++ や Java の this, Ruby の self にあたる変数）, インスタンス生成文の返戻値, クラス検査などからクラスに関する情報を取得し, それをデータフローに沿って伝播することでレシーバのクラスを推定する.

例として図 1（a）に示すプログラムをデータフロー解析による従来の推定手法で解析した結果を図 1（b）に示す.

従来の推定手法では, インスタンス生成文の返戻値を情報源にすることによってメソッド c1 のレシーバがクラス A だと推定することができる（図 1（b））. しかし, 図 1（c）に示すプログラムにおいては, レシーバのクラスの推定に使用できる情報源が存在しないため, メソッド c2 のレシーバのクラスを推定することができない（図 1（d））.

3　提案手法

提案手法では, データフロー解析による推定手法を拡張し, レシーバが呼出しに成功したメソッドにより, レシーバの取りうるクラスの情報を縮小することで精度の改善をはかる.

まず, ローカル変数ごとに, 変数が指し示すインスタンスが取りうるクラスの集合を保持する領域を作成し, すべてのクラスを取りうる状態（ここでは ANY とする）に初期化する. また, 現在のレシーバを表す変数に対しても, クラスの集合を保持する領域を作成し, クラスの集合を現在のメソッドを定義するクラスと is-a な関係のクラスの集合に初期化する. 加えて, ローカル変数と現在のレシーバを表す変数それぞれに対して,

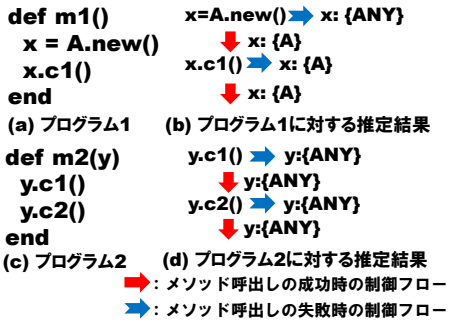

図 1　従来手法による推定結果

同じインスタンスを指し示している変数を管理するための別名（must-alias）の集合を保持する領域を作成し, 空集合で初期化する.

次に, 解析対象のプログラムの次の箇所において, それぞれ対応した処理を行いながらフローの解析を行う.

リテラル, インスタンス生成文の返戻値　代入された変数に対応するクラスの集合を, 情報源から得られたクラスのみを含む集合にする. 代入された変数に対応する別名の集合に含まれている変数に対して, それぞれの変数に対応する別名の集合から代入された変数を取り除き, その後, 代入された変数に対応する別名の集合を空集合にする（以下, この処理を別名の集合をリセットするという）.

メソッド呼出し　メソッド呼出しに成功した場合, レシーバを指し示している変数に対応するクラスの集合から呼出しに成功したメソッドを呼出すことができないクラスを取り除く. レシーバを指し示している変数に対応する別名の集合に含まれている変数に対しても, それぞれの変数に対応するクラスの集合から呼出しに成功したメソッドを呼出すことができないクラスを取り除く.

変数同士の代入　右辺の変数に対応するクラスの集合を, 左辺の変数に対応するクラスの集合にする. 右辺の変数に対応する別名の集合に含まれている変数に対して, それぞれの変数に対応する別名の集合に左辺の変数を含める. その後, 右辺の変数に対応する別名の集合に左辺の変数を含める. 最後に, 左辺の変数に対応する別名の集合をリセットした後, 右辺の変数と右辺の変数に対応する別名の集合に含まれている変数それぞれを, 左辺の変数の別名の集合に含める.

メソッド呼出しの返戻値の代入　メソッドの返戻値のクラスが特定できるなら, 代入された変数に対応しているクラスの集合をメソッドの返戻値のクラスのみを含む集合にし, メソッドの返戻値のクラスが特定できないなら, 代入された変数に対応するクラスの集合を ANY にする. いずれの場合においても, 代入された変数に対応する別名の集合をリセットする.

クラス検査　検査された変数に対応するクラスの集合を, 検査したクラスのみを含む集合にする. 検査された変数に対応する別名の集合に含まれている変数に対しても, それぞれの変数に対応するクラスの集合を, 検査したクラスのみを

An implementation of class inference on a dynamic compiler for Ruby

† Yuki Koizumi, Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University

‡ Yuji Chiba, Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University

§ Koichi KUBOTA, Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University

含む集合にする．

上記以外の代入　代入された変数に対応するクラスの集合を，ANY にし，代入された変数に対応する別名の集合をリセットする．

データフローのマージ時には，クラスの集合については集合同士の和をとり，別名の集合については集合同士の積をとる．

図 1（b）に示すプログラムに対して提案手法を使用し，解析した結果を図 2 に示す．

図 2 から，呼出しに成功したメソッド名により取りうるクラスを限定しているため，メソッド c2 のレシーバのクラスがメソッド c1 を呼出せるクラスであることがわかる（図 2）．

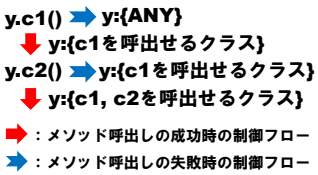

図 2　プログラム 2 の提案手法による推定結果

4　評価

Ruby 向け動的コンパイラ [1] に提案手法を実装し，Ruby Benchmark Suite のマクロベンチマークを使って，従来のクラス推定を使用した場合と提案手法によるクラス推定を使用した場合での脱仮想化の適用率の増分を評価した．

ここで脱仮想化の適用率とは，脱仮想化の適用数を動的コンパイルしたメソッドの中に含まれていたメソッド呼出しの箇所数で除した値である．脱仮想化の適用先は呼出すメソッドを唯一に推定できたメソッド呼出しとした．なお，同じメソッドを複数回コンパイルする際に前と同じ仮想呼出しを脱仮想化する場合があるが，適用の成否は静的なメソッド呼出しの箇所ごとに集計し，1 度でも脱仮想化を適用した箇所を適用に成功した箇所とみなした．

評価に使用した計算機は Dell Precision T5400（Xeon E5420 2.50GHz, RAM 3.2GByte），OS は Ubuntu9.10 である．

脱仮想化の適用率の増分の評価結果を表 1 に示す．また，実行速度の比較結果を図 3 に示す．

表 1 は次に示す 4 つの条件で実行を行った場合の脱仮想化適用数を示す．

A　従来のクラス推定を使用した場合の脱仮想化適用数
B　提案手法のクラス推定を使用した場合の脱仮想化適用数
C　動的コンパイルしたメソッドの中に含まれていたメソッド呼出しの箇所数
D　脱仮想化の適用率の増分 (単位：%)

図 3 において，動的コンパイル済みコードの速度は，実行プロファイル取得後の動的コンパイルが終了してから計測したものなので，動的コンパイルや実行プロファイルの採取に伴うオーバーヘッドを含まない．

図 3 は次に示す 3 つの条件で実行を行った場合の実行速度がインタプリタ実行に対して何倍速くなっているかを示す．

（1）　動的コンパイルあり
（2）　（1）に加え従来のクラス推定を適用
（3）　（1）に加え提案手法のクラス推定を適用

測定値は，個々のベンチマークを 20 回実行し，そのうち最小の実行時間を測定結果として採用した．

評価の結果，脱仮想化の適用率は最大で 38.1%，相乗平均で 6.06% 増加することがわかった．

表 1　脱仮想化の適用数と適用率の増分

ベンチマーク名	A	B	C	D
list	4	12	21	38.1
gzip	4	7	13	23.1
observ	4	6	18	11.1
cal	55	56	87	1.15
hilbert_matrix	38	38	90	0
norvig_spelling	14	14	33	0
rcs	2	2	12	0
pi	1	1	1	0
parse_log	2	2	16	0
mpart	3	3	18	0
sudoku	4	4	12	0

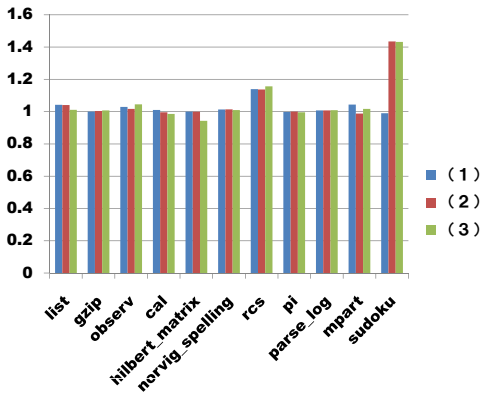

図 3　実行速度比較

図 3 より，表 1 の中で適用率の増分が上位である list, cal ベンチマークの性能は明確には上がっていないことがわかる．その原因を調査したところ，list はガーベジコレクションに実行時間の多くを割いており，インタプリタの実行時間が少ないためだとわかった．cal は従来の推定手法に比べて 1 箇所脱仮想化の適用先が増加しているが，その増加した適用先は呼出される回数の少ない仮想呼出しだとわかった．

5　結論

クラス推定の精度を改善する手段として，呼出しに成功したメソッドの名前を利用する方法を提案した．Ruby 向け動的コンパイラ [1] に提案手法を実装し，Ruby Benchmark Suite のマクロベンチマークのプログラムを使い，従来のクラス推定と提案手法のクラス推定とで，脱仮想化の適用率の増分を評価した．その結果，適用可能な確率が最大で 38.1%，相乗平均で 6.06% 増加することがわかった．

参考文献

[1] 石井直也，村田俊哉，千葉雄司，土居範久：Ruby 向け動的コンパイラの実装，情報処理学会論文誌 プログラミング，to appear.
[2] 小野寺民也：オブジェクト指向言語におけるメッセージ送信の高速化技法，情報処理 38(4), 301–310, 1997.
[3] Craig Chambers and David Ungar：Interactive Type Analysis and Extended Message Splitting: Optimizing Dynamically-Typed Object-Oriented Programs., In Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation, 25(6):150–164, 1990.
[4] Jeffrey Dean, David Grove, Craig Chambers ：Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis. , In Proceedings of the European Conference on Object-Oriented Programming, 1995.