5M-07

FFIをもつSchemeインタプリタのための部分コンパイル方式コンパイラの構成法

寺澤哉門† 小宮常康†

†電気通信大学 大学院情報理工学研究科

1 背景

プログラミング言語 Scheme の実行速度向上のため，より実行速度が速い C にコンパイルする方法がある．C へのコンパイラを作成するには Scheme の実行モデルを C で模倣しなければならない（図 1）．Scheme プログラムの中には模倣せずに済むものもあり，これらは C の素直なコードへ変換する最適化が可能である．実行モデルの模倣や最適化には高度な技術が必要とされる．

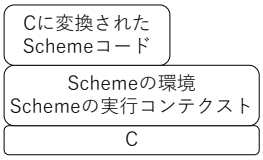

図 1: C で Scheme の実行モデルを模倣する様子

本研究では作成が容易でありながら比較的高性能なコンパイラを実現するため，Scheme プログラムのうち，比較的容易に見つけ出せる素直な C にコンパイルできる部分のみをコンパイル対象とするコンパイラを提案する．それ以外の部分の実行はインタプリタに任せ，コンパイル後の Scheme プログラムと C プログラムはインタプリタの FFI (Foreign Function Interface) を利用して組み合わせる．FFI とは，あるプログラミング言語から他のプログラミング言語の関数にアクセスする機能である．

2 関連研究

Bartlett の Scheme->C [1] はいわゆるクロージャ変換によってクロージャを実現している．変数束縛は可能なら C の変数束縛として実現するが，無限の寿命を持つ束縛を必要とするクロージャや第一級継続のために，束縛をヒープに置くこともある．一方我々のコンパイラは，後者の束縛を必要とする場合はインタプリタで実行する．Bartlett の Scheme->C は Scheme のほとんどの機能を C にマッピングしているが，真正末尾再帰の一部を諦めている．

Scheme インタプリタ SCM 用の Scheme-to-C コンパイラ Hobbit [2] は，コンパイル時にすべてのソースプログラムが得られると仮定して大域的な解析を行うことで，クロージャの生成を抑制し素直な C プログラムへ変換できるような最適化を行っている．Hobbit でも末尾再帰の除去は完全ではない．

3 提案するコンパイラの概要

提案するコンパイラとインタプリタの組み合わせ方を図 2 に示す．以下ではコンパイル対象の抽出方法の概略について述べる．基本方針としては素直な C で直接表現できるものをコンパイルの対象とする．生成時の環境を必要とするクロージャの生成やその本体はインタプリタに実行を任せる．実行モデルの模倣が不要な Scheme の関数は以下の条件で検出することにした．

> box 化が必要な変数を含まず，関数の各変数が C の束縛で直接的に表現できること

変数の box 化は変数をクロージャに閉じ込める場合や（第一級継続のキャプチャに備えるため）代入が行われる場合に必要となる．クロージャを生成する必要がなければ，入れ子の関数定義は lambda-lifting によってトップレベルに移動する．以下の関数 foo の内部の関数 bar は foo のスコープを外れて使用されることは

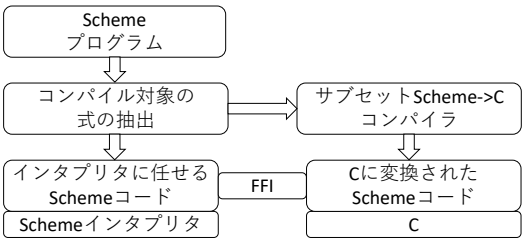

図 2: 提案するコンパイラの概要

A partial Scheme->C compiler for Scheme interpreters with FFI
Saimon Terazawa† and Tsuneyasu Komiya†
†Graduate School of Informatics and Engineering, The University of Electro-Communications

ないためクロージャを生成する必要はなくCへコンパイル可能である．

```
(define foo (lambda (x)
  (define bar (lambda (y) (+ x y)))
  (bar x)))
```

以下の関数fooの内部の関数barはfooの外で使用される可能性があり，barはfooが呼び出されたときの環境を保持するクロージャとする必要がある．このような関数fooの実行はインタプリタに任せる．

```
(define foo (lambda (x)
  (define bar (lambda (y) (+ x y)))
  bar))
```

以下の関数fooのxはfooのlambda式だけで使用され，他のlambda式で使用されない．これはCのローカル変数と同じスコープと寿命で済むため，Cの代入で表現できる．

```
(define foo (lambda (x) (set! x 0)))
```

以下の関数fooの内部では関数barに閉じ込められたxへ代入を行っている．関数barのxと代入式のxは同一の変数であり，xが書き換えられた時点でbarに閉じ込められたxも書き換えられなければならない（そのためbox化する）．これはCの代入では直接的に表現できないため，インタプリタに実行を任せる．

```
(define foo
  (lambda (x)
    (define bar (lambda () x))
    (set! x 0)
    bar))
```

以上のような検査でSchemeの実行モデルの模倣が不要な部分を見つけ出し，Cにコンパイルする．

4 性能評価

提案する手法で構成したコンパイラの性能を評価するためにSchemeの純粋なインタプリタであるSIOD (version 3.5)を使用した．ベンチマークにはLisp系のGabrielベンチマークを使用し，以下のScheme処理系と比較した．

- ChezScheme (version 9.5.1)：Schemeから機械語にコンパイルする最も高速なコンパイラ．
- CHICKEN (version4.13.0)：末尾呼び出しの除去を実現するために継続渡しスタイルのC [3]へ変換して実行するScheme to Cコンパイラ．すぐれた最適化器をもつ．
- SCM (version 5f2)：高速なインタプリタ．更にHobbitと呼ばれるScheme to Cコンパイラを持つ．

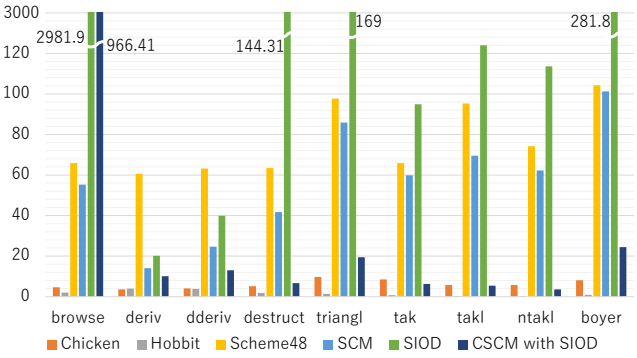

図3: ChezSchemeの実行時間を1とした実行時間比

- Scheme48 (version 1.9.2)：バイトコードインタプリタ方式の処理系．

図3はChezSchemeの実行時間を1としたときの各処理系の実行時間比である．ChezSchemeはグラフにほとんど現れないほど高速なため省略している．「CSCM with SIOD」が提案するコンパイラCSCMとSIODを組み合わせた結果である．自己末尾再帰の最適化が未実装なためbrowseは大差をつけられている．browse以外のベンチマーク結果はChezSchemeの3.5~25.3倍の実行時間となった．ほとんどのベンチマーク結果において優れた最適化器をもつScheme to Cコンパイラには及ばないが，バイトコードインタプリタより高速となった．takのような単純な関数呼び出しであれば最適化器をもつコンパイラに匹敵する結果となった．

5 まとめと今後の予定

Schemeから実行モデルが異なるCへのコンパイルには実行モデルの模倣や高度な最適化が必要になる．提案手法はこれらが不要なため比較的容易にScheme to Cコンパイラを実装できる．優れた最適化器を持つScheme to Cコンパイラには及ばないもののバイトコードインタプリタを上回る性能を実現できた．

Schemeの実行モデルの模倣が不要なScheme関数の現在の検出方法ではCにコンパイルできるSchemeのプログラムが限られる．より多くのSchemeコードをコンパイル対象化するようなSchemeコードの変形を検討している．

参考文献

[1] Joel F. Bartlett: Scheme->C a Portable Scheme-to-C Compiler, WRL Research Report 89/1 (1989).
[2] Tanel Tammet: Lambda-lifting as an optimization for compiling Scheme to C, Chalmers Univ. of Tech., Dept. of Computer Sciences, Goteborg, Sweden (1996).
[3] Baker, H. G.: CONS Should Not CONS Its Arguments, Part II: Cheney on the M.T.A. *SIGPLAN Notices,* Vol.30, No.9, pp.17–20 (1995).