6L-02

インタプリタとコンパイラ生成コードの対比によるコンパイラ実行時環境理解支援ツール

李 志弘† 小宮 常康‡

†,‡ 電気通信大学 大学院情報理工学研究科

1 はじめに

コンピュータ教育においてプログラム実行を可視化することが有用であることは、いくつかの研究により証明されている。そのため、多くのアルゴリズム可視化ツールが開発されている。これらのツールは、コンピュータプログラミングに初めて挑戦する学習者を対象としている。一方で、言語処理系の初学者にとっては、コンパイラやインタプリタの動きを理解するのを助ける可視化ツールはあまりない。

類推的思考は、新しい知識の習得効率を改善することができる。例えば、C++言語を学んだプログラマが Java 言語を学ぶとき、C++言語の知識を類推することで、Java にすばやく取り組める。これは処理系を学ぶ学習者にも当てはまる。学習者はまずインタプリタの構造を学び理解し、その後コンパイラや仮想マシン(VM)を学ぶケースがよくある[1]。インタプリタとコンパイラを類推すると、インタプリタの構造を理解することができる学習者は、コンパイラの対応する部分の原理もすばやく理解できると期待される。しかし、実際の実行では、インタプリタと VM の動作はかけ離れており、学習者には難しいと感じることがある。その具体的要因の一つは，関数呼出しや環境が VM 上でどう実現されているかの理解不足（実行時環境の理解不足）にあると考える。

これを踏まえて、本研究では、Scheme 言語の処理系の初学者が実行時環境を理解するのを助けるために、ソースプログラムをコンパイルして生成されるコードの VM 上での実行と，同一ソースプログラムのインタプリタによる実行を，同時並行・同期的にステップ実行しつつ，可視化した双方の環境等を並べて見せるツールを実装した。さらに、今回用いた対象処理系以外の処理系にも可視化機能を容易に組み込めるようにツールの構成部品を設計した。可視化機能組み込みマクロで簡単に自分の処理系が視覚化できる。

A Tool for Understanding Compiler Runtime Environment by Contrasting Executions of Both Interpreter and Compiler Generated Code

†Zhihong Li

‡Tsuneyasu Komiya

†,‡ Graduate School of Informatics and Engineering, The University of Electro-Communications

2 提案手法

2.1 ツールの全体像

図 1 は、ツールのスクリーンショットを示しており、実行したい Scheme プログラムは左上に表示されている（図の 1）。その下のボタンは Scheme プログラムの実行を制御するものであり、ブレークポイントのオンオフやステップ実行などができる。実行中の式や命令列は右上に表示されている（図の 2）。図中の 3,4,5 はそれぞれインタプリタにおける環境、スタックとヒープ中のデータが表示されている。画面の下部には、インタプリタが実行している操作の名前や VM が実行している命令など、実行に関する特定の情報が表示されている（図の 6）。最後に、右下のボックスはナレーションであり、実行に関する詳しい説明を提供する。ナレーションはカスタマイズ可能で、教師はコースの内容に応じて実行中のプログラムの詳細な説明を提供することができる。

本ツールの対象処理系には，文献[2]による Scheme インタプリタとコンパイラ，VM（いずれも Scheme で実装）を用いた（一部拡張している）．文献[1]による処理系にも対応できるはずであるがその確認は今後の課題とする．ツールは Web ブラウザ上に実装した．また，対象処理系を動作させるための処理系には BiwaScheme[3]を用いた．

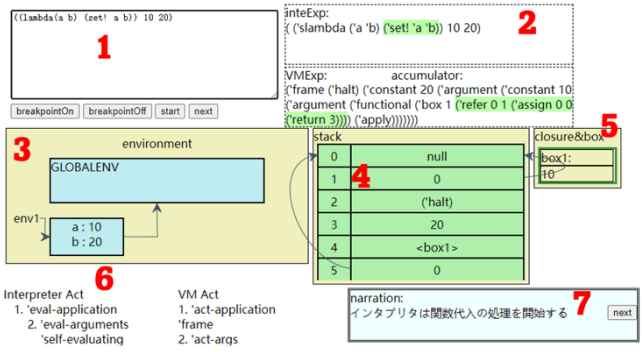

図 1 ツールのスクリーンショット

2.2 インタプリタと VM の同期実行方法

本ツールにおけるインタプリタと VM の並行実行は，Scheme の継続機能によるコルーチン（イ

ンタプリタコルーチンと VM コルーチン）により実現した。しかし両者の実行のステップの粒度には違いがあり（eval/apply によるビッグステップ vs. 機械語のスモールステップ）、両者の実行の足並みを揃える手がかりが不足する。

幸い，文献[1]や[2]などのインタプリタ/コンパイラの両方を扱う文献において、コンパイル処理の粒度は（eval-arguments に対する compile-arguments などのように）インタプリタの実行ステップと同じであることが多い。そこで、コンパイラによる生成コード中に、ビッグステップの処理の境目を示す疑似命令を挿入することとし，基本的にはビッグステップで同期しながら並行実行し、VM 命令列にしかない処理の間はインタプリタの実行は進めない、という形態をとることにした（図 1 の 6 において 1 や 2 の処理はビッグステップの処理を示しており、番号の付いていない処理(frame)は VM 命令を示している）。

2.3 可視化機能組み込みマクロ

本ツールを特定の処理系を視覚化するようにデザインすると、本ソフトウェアの有用性が限定される。また、処理系の一部を修正すると視覚化ツールが動かなくなるようでは非常に面倒である。そこで、本ツールのうち処理系にかかわる部分は様々な処理系に迅速に対応できるマクロ定義として設計した。これらのマクロ定義は、処理系の改造対象となる関数名をパラメータとして渡すだけで、ブレークポイント処理の追加、アニメーションの生成、ナレーションの追加機能などを組み込める。

```
01. (define-macro embed-draw-vm-argument
02.   (lambda (fun)
03.     `(let* ((org-fun ,fun))
04.        (set! ,fun
05.              (lambda (a x f c s)
06.                ;描画プログラム
07.                (view-stack-push a 'argument)
08.                (org-fun a x f c s))))))
09. ;argument処理の関数を再定義
10. (embed-draw-vm-argument VM-argument)
```

図 2 可視化機能組み込みマクロと使用例

例えば、図 2 の 10 行目は組み込みマクロの使用例である。文献[2]の処理系のアーギュメント処理関数名（VM-argument）だけをパラメータとして渡すとスタックに引数をプシュするアニメーションの機能の組み込みが完了する。

3. 使用例

変数アクセスは言語処理系において重要な部分である。本ツールでは、その部分はアニメーションを用いて表すことで、学習者に対しわかりやすい形式で概念を伝えることができる。

```
01. ((lambda(a b)
02.    ((lambda(c d)
03.       b) 3 4) 1 2))
```

図 3　変数アクセスの例

図 4 はインタプリタによって図 3 の変数 b にアクセスする際のツールの描画を示している。まず、env2 で変数リストを探索するがそのフレームでは見つからず、静的リンクを辿って次のフレームに入る。そこで 2 回の比較を行った後で、変数 b の束縛を発見し、合計 4 回の操作を行った。

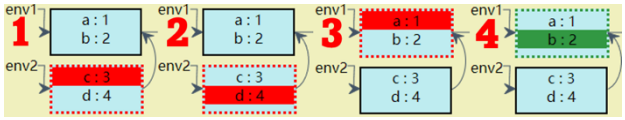

図 4 インタプリタでの変数アクセス

図 5 は、VM 命令列による実行を示している。最初に、コンパイラによって生成された「refer 1 1」命令を使用し、現在のフレームから一回静的リンクを辿ったフレーム内のオフセット 1 の場所（変数 b の場所）にアクセスする。

また、図 4 のフレームあるいは変数束縛をマウスで選択すると、図 5 における対応するフレームあるいは束縛がハイライトされる。

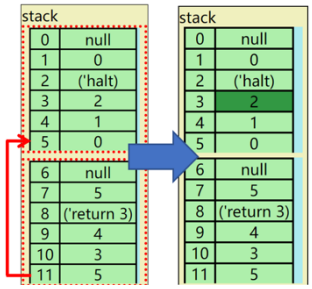

図 5 VM での変数アクセス

4 関連研究

Mieru Compiler[4]では、ソースコードとアセンブリコードの対応関係を対話的に可視化するハイライト機能を使用している。本研究では類似の方法を使用している。

Tango[5]と呼ばれるアルゴリズム可視化ツールでは、ソースコードを大幅に変更せずに、アルゴリズムのアニメーション化を実現できる。本研究では、可視化機能組み込みマクロを使用し、ソースプログラムの修正を最小限に抑えることを実現した。

参考文献

[1] Abelson, H., Sussman, G. J., Sussman, J., & Perlis, A. J. (1996). *Structure and interpretation of computer programs, 2nd ed.* The MIT Press.
[2] R. K. Dybvig. 1987. Three implementation models for scheme. Ph.D. Dissertation. University of North Carolina at Chapel Hill, USA. Order Number: UMI Order No. GAX87-22287.
[3] BiwaScheme, https://www.biwascheme.org/
[4] 権藤克彦 野崎広弥 教育用コンパイラ XCC とその可視化ツール Mieru compiler: 電子情報通信学会論文誌 D. vol. 108, no. 444, 2008, pp. 35-42.
[5] Stasko, J.T., Profile, V. and Metrics, O.M.V.A. (1990) Tango: A Framework and system for algorithm animation, ACM SIGCHI Bulletin. vol. 21, issue 3.