4ZC-06

置き換えモデルによるScalaの実行過程の可視化*

竹内 俊貴†[1] 酒井 三四郎‡[2]

[1] 静岡大学大学院総合科学技術研究科
[2] 静岡大学情報学部

1 研究の背景・目的

近年、関数型プログラミングのスタイルの利点が見直されつつある。その理由として、簡潔な記述ができること、副作用がなく厄介なバグが起きにくいことなどが挙げられる。ただし、C言語やJavaなど、別のプログラミングパラダイムを習得してきたプログラマにとって、高階関数や再帰呼び出しを多様する関数型プログラミングのスタイルはギャップが大きく、さらに、パターンマッチやラムダ式のように高度に抽象化された式が多いため習得が難しい問題がある。そのため、新たに関数型プログラミング言語を学習する初学者には何らかの補助が必要だと考える。

プログラムの理解を補助する手法として、実行過程や実行状態の可視化はよく行われる。ただし、関数型プログラミングは近年見直されてきたパラダイムであるため、適切な可視化ツールが存在していない問題がある。そこで本研究の目的は、関数型プログラミングに対応した適切な可視化ツールを提案し、問題の解決を図ることである。

2 関連研究

プログラミング言語の実行過程の可視化を行うツールでは、Javaを対象としたLevyらのJeliotがある[1]。Jeliotはクラスベースのオブジェクト指向を対象としているため、インスタンスへの操作を中心に、状態変更や関数呼び出しのアニメーションを表示している。しかし、関数型プログラミングでは状態を持たず、関数の組み合わせで結果を作ることが多い。そのため、関数型プログラミングの可視化には別のアプローチが必要となると考える。また、Jeliotを用いた対照実験の結果では、Jeliotを使用した学生の成績を向上させた。そのため、プログラムの適切な可視化は学習の助けになると考えられる。

同様に、Javaの実行過程を可視化するツールとしてJamesらによるjGRASPがある[2]。jGRASPは制御構造やデータ構造を中心としたアニメーションを表示する。しかし、関数型プログラミングには、パターンマッチやif式などJavaには無い制御構造があるという違いがある。データ構造に大きな違いはないと考えるが、関数型プログラミングでは特にListを対象とした抽象度の高い演算が多くあるため、Listへの演算に対する補助が新たに必要だと考える。

3 本研究のアプローチ

本研究では、以下の2点の特徴を持つアニメーションを提案し目的の達成を図る。可視化対象のプログラミング言語には、Javaの後継として今後使用者が増えると予想されるScalaを選択した。

(1) 置き換えモデルで表現

副作用のないプログラムは、置き換えモデルで表現することができる。置き換えモデルはプログラムを段階的に評価していくため、学習者にとって分かりやすいモデルだと考える。

置き換えモデルには作用順序の評価と正規順序の評価の二種類が存在し、実際のプログラムの実行過程は作用順序の評価である。そのため、学習者に正しいメンタルモデルを構築できると考え、作用順序の評価を採用した。

(2) 演算過程の一部を省略可能

学習者の理解度に応じ、可視化の過程を省略可能にすることで、学習効率を上げることができると考えた。対象の構文要素は、パターンマッチング式とリストに対する演算の2つである。その理由は、これらの式は置き換えモデルでは単純に置き換えられてしまうが、抽象度が高く過程が複雑な場合がある。そのような時に、学習者の理解度次第で結果のみ見たい場合と過程も見たい場合があると考えたからである。

4 可視化ツールの提案

4.1 アニメーション方法の提案

置き換えモデルによるアニメーションの様子を図1に示す。置き換えモデルによるアニメーションでは、置

*Visualization of Scala's execution process by replacement model

†Toshiki Takeuchi・Graduate School of Integrated Science and Technology, Shizuoka University

‡Sakai Sanshiro・Faculty of Informatics, Shizuoka University

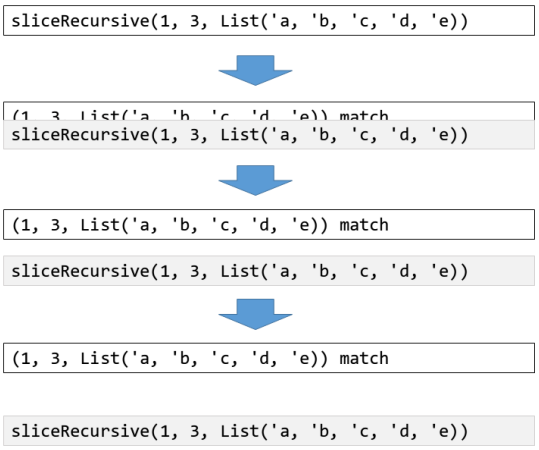

図 1: 置き換えモデルによるアニメーション

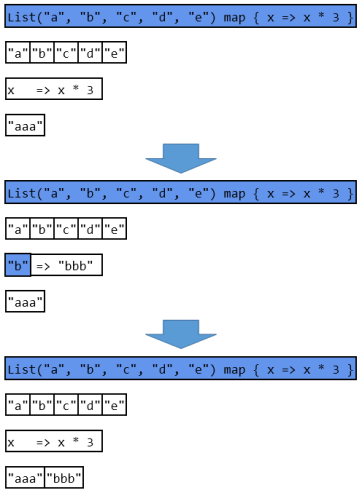

図 2: List への演算に対するアニメーション

き換え前後の式が紙芝居のように前後に重なっており、手前にある置き換え前の式が紙芝居のように下にスライドし、背後にある置き換え後の式となるアニメーションを採用した。スライドした式を最大 2 つまで画面に表示することで、置き換え前後の値を比較可能にする。

List への演算は過程が追いきれない場合があり、置き換えモデルだけでは不十分だと考えたため別途専用のアニメーションを用意した。アニメーションの例を図 2 に示す。アニメーションでは、最初に、対象の式から List の要素と高階関数を展開する。次に、List の各要素に対し関数を適用した結果と中間結果を逐次表示する。最後に関数の適用結果を表示し、対象の式が結果に置き換えられる。

4.2 可視化ツールのインターフェース

本ツールは、大きく分けて次の 4 つの要素から構成される。ツールの外観を図 3 に示す。①はソースコードビューで、可視化対象となるプログラムを表示する。

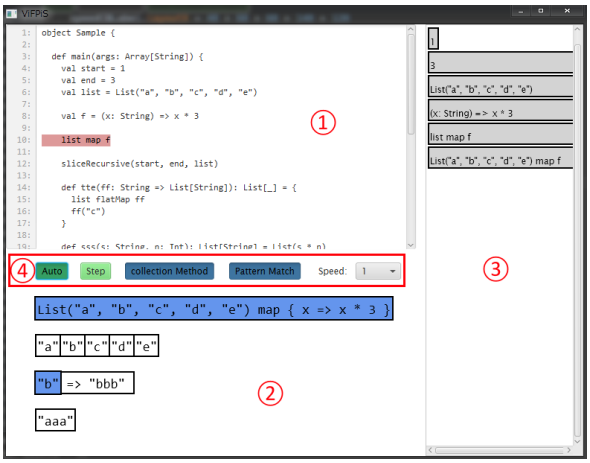

図 3: システムの外観

実行中の式に対応する行がハイライトされる。②はアニメーションキャンバスで、プログラムの実行過程をアニメーションとして表示する。③は履歴キューで、置き換えられた式が貯まり、選択した任意の状態へ戻ることができる。コールスタックはインデントされ、呼び出しの深さを視覚的に示す。④はコントロールパネルで、アニメーション速度の変更や、ステップ実行を制御するボタンがある。3 章で挙げた演算過程の一部を省略する機能を実行するボタンもここに配置されている。

5 まとめ

本研究では、関数型プログラミング言語に対応する適切な可視化ツールがない問題に対し、置き換えモデルを用いた新たな可視化ツールを提案した。今後の課題として、提案ツールの評価を行う必要がある。評価方法は関数型プログラミング言語初学者の情報学部生を対象に、ツールの有無による対照実験を考えている。その他、対応する構文の幅を広げること、大きく、複雑なプログラム対して可視化をする際のデザインの検討などが求められる。

参考文献

[1] R. Ben-Bassat Levy, M. Ben-Ari, P. A. Uronen: The Jeliot 2000 program animation system, Computers & Education, 40 (1), pp.1-15 (2003).

[2] James H. Cross, II, T. Dean Hendrix: jGRASP: an integrated development environment with visualizations for teaching Java in CS1, CS2, and beyond, Journal of Computing Sciences in Colleges, vol.23, no.2, pp.170-172 (2007).