Elixir におけるリスト処理の変換および比較ツールの開発と評価

1250370　松尾　秀　【 ソフトウェア検証・解析学研究室 】

1　はじめに

Elixir は，Erlang VM 上で動作する関数型プログラミング言語である．この言語は，リストをはじめとしたコレクションに対する処理の記法が複数用意されている．例として，リストの各要素に対して関数を適用する map 処理には，再帰を用いた記法や，組み込みライブラリである Enum モジュールの関数 Enum.map を用いた記法などがある．しかし，どの記法が適しているかをコーディングの段階で自力で判断するのは，経験の浅い開発者には困難であるし，書き換えるのも手間である．

本研究では，数値型のリストに対する map，filter，reduce の 3 種類の処理を記述する方法として，再帰を用いた記法と Enum を用いた記法の 2 種類に着目する．片方の記法が採用されたコードの抽象構文木（AST）から，もう片方の記法へ変換可能な箇所を検出し，変換前後の性能比較を行うツールを提案する．

2　関連研究

Vegi らによる Elixir のリファクタリングに関する研究により作成されたリファクタリングカタログの中には，再帰関数を高階関数を用いた関数に変換するリファクタリング手法が提案されており，まさに再帰関数から Enum モジュールの関数への変換が例として挙げられている．このリファクタリングを行う動機として，再帰は抽象度が高く難解であるため，再帰の詳細を隠しつつ反復処理が行える高階関数を利用すべきであると述べられている [1][2].

また，提案ツールに類似する静的解析ツールとして，Elixir には Credo[1]，同じ関数型言語である Haskell には Hlint[2]がある．いずれのツールにも，より簡潔で望ましいコーディングスタイルへの変換を提案する機能があるが，変換前後の性能比較を行う機能は備わっていない．

3　提案手法

提案ツールは，大きく分けて 2 つのモジュールに分けられる．1 つ目は，Elixir の AST を解析し，変換可能な箇所を検出し変換を行うモジュールである．AST の中から再帰（または Enum）を用いた記法を検出し，もう一方の記法に変換した AST を返す．

2 つ目は，変換前後のコードを比較するモジュールである．変換前後のコードをそれぞれコンパイルし，Elixir のベンチマークライブラリ Benchee を用いて性能比較を行う．

[1] https://hexdocs.pm/credo/overview.html
[2] https://github.com/ndmitchell/hlint

4　実行結果

以下は，リスト中の値の和を計算する関数 sum の 2 つのコード例である．これらは提案ツールによって相互に変換可能であり，一方を入力すればもう一方のコードが生成される．

再帰
```
defmodule Sample do
  def sum([]), do: 0
  def sum([head | tail]), do: head + sum(tail)
end
```

Enum
```
defmodule Sample do
  def sum(list) do
    list |> Enum.reduce(0, &(&1 + &2))
  end
end
```

生成後，両者の性能比較を自動的に行い，例として表 1 のような結果を得ることができる．これは，それぞれの sum 関数に対し「1〜100 までの乱数からなる 長さ 10000 のリストを引数として 100 回呼び出す」処理を実行した際の，秒当たりの実行数 (IPS) と平均実行時間を示している．この表からは，sum の処理は再帰を用いた記法の方が Enum.reduce を用いるより実行速度が約 1.38 倍速いことが読み取れる．

表 1　関数 sum の記法によるベンチマーク比較例

	秒あたり実行数（IPS）	平均実行時間
再帰	110.41	9.06 ms
Enum	80.21	12.47 ms

これらの結果から，可読性や性能に基づいてどちらの記法を選択すべきかを判断することができる．

5　まとめ

本研究では，Elixir におけるリスト処理の 2 種類の記法に対する変換，および比較を行うツールを提案した．結果，提案ツールによってそれぞれの記法を自動で変換する機能を実現でき，また変換前後の性能比較を行うことができた．今後は，提案ツールをより幅広いケースに対応させ，実際の Elixir プロジェクトに適用できるようにすることが課題である．また，コードメトリクスによる定量的な可読性の比較や，実行時間以外の指標での性能比較を実装することも検討している．

参考文献

[1] L. Vegi, M. Valente, “Towards a Catalog of Refactorings for Elixir”, IEEE ICSME, 2023.

[2] L. Vegi. “Elixir-Refactorings: Catalog of Elixir Refactorings”, GitHub repository, https://github.com/lucasvegi/Elixir-Refactorings. 2025 年 1 月閲覧.