C 既習者の Rust 学習における列挙型の所有権の可視化による理解支援方法の提案

M2022SE003 堀江傳貴
指導教員：蜂巣吉成

1 はじめに

プログラミング言語 Rust は C や C++ と同様にシステムレベルのプログラミングが可能な言語である．Rust は所有権によって，メモリリークやメモリの 2 重解放といったメモリ関連のエラーをコンパイル時に検証可能である．Rust はメモリ安全性の高さから，モジュールを C から置換えるプロジェクトが Linux や android で進行している．

C 言語を学んだ開発者が，既存の C プログラムを書き換える形で Rust の学習を行うことが想定される．しかし，所有権や C と異なる概念の列挙型の存在によって，C と同様の記述ではコンパイルエラーとなり，解決策がわからず Rust への理解が進まないことがある．行き詰まりの原因として，所有権のムーブが発生するオブジェクトを列挙型から取り出す際に，所有権のムーブによって，所有権のルールに関する制約条件が満たされなくなることによるエラーが発生し，学習者がそれを理解できない問題がある．

本研究では，列挙型からのオブジェクト取り出しにおいて学習者が理解できない問題を解消するための，Rust の理解支援方法を提案する．C と同様の記述からどの列挙型のメソッド呼出しを追加するか，正しい記述となるメソッド候補とエラーとなるメソッド候補を抽出し，メソッド呼出しを追加した場合の所有状況の可視化図を作成するための要素を定義する．抽出したメソッド候補と定義した要素で作成した可視化図を組み合せたメソッド候補提示グラフを学習者に提示する．学習者は正しいメソッド候補を確認し，コンパイルエラーを解消できる．また，正しいメソッド候補とエラーとなるメソッド候補の適用による所有状況を可視化図によって確認し，所有権のムーブによって制約条件を満たさなくなる状況を理解できる．

2 関連研究

Almeida ら [1] の研究では，学習教材として登場するコンパイル可能な Rust プログラムを対象として，変数の所有権と借用イベントをタイムラインとして可視化するツール RustViz を提案している．所有状況の可視化が本研究と同様である．RustViz はコンパイル可能なサンプルプログラムレベルの問題を対象とし，可視化についてはタイムライン形式で所有権のムーブ状況のみを表示しているが，本研究ではコンパイルーエラーの場合も対象とし，可視化については所有されているオブジェクトを型情報を含めて，所有状況として表示する点が挙げられる．

Coblenz ら [2] の研究では，ライブラリベースのガベージコレクタ Bronze を提案している．ライブラリベースのガベージコレクタを学習者に利用させることで，所有権の煩雑さを一旦簡潔にした上で，再度ガベージコレクタを使用せずに演習問題に取り組ませることで，学習支援としている．Rust の理解を段階的にするアプローチに関しては，本研究の C での理解を Rust の理解に活かすという点が同様である．本研究の Coblenz ら [2] の研究との差別化点については，Almeida ら [1] の研究と同様にコンパイルエラー時の理解支援も対象とすることが挙げられる．

3 問題分析

C 既習者が書換え学習を行うことを想定し，C の単方向リストプログラムの Rust への素朴な書換えを行う際に，コンパイルエラーとなり，エラーを解消できない箇所を行き詰まり箇所として問題分析を行った．想定する C 既習者は次のとおりである．

- 構造体やポインタを含めた C 言語の文法を一通り学び，リストや木などのデータ構造を理解している
- Rust の参考書を一通り読み，基本的な文法は理解している

分析の結果，列挙型と所有権に関連して，次の点で行き詰まりが発生することがわかった．

(1) 列挙型のメソッドがわからない
(2) ムーブ不可の制約条件下でムーブが発生している状況が理解できない

ノード走査処理を行うリスト操作関数に関するソースコード 1，2 を用いて，それぞれ説明する．

ソースコード 1 リスト走査の代入文 (C)

```
1  for (i = 0; i < index; i++) {
2    prev_node = prev_node->next;
3  }
```

ソースコード 2 リスト走査の代入文 (Rust)

```
1  struct Node { // 構造体宣言
2    value: i32,
3    next: Option<Box<Node>>,
4  }
5  /* -- 省略 -- */
6  for _i in 0..index {
7    // 型不一致エラー
8    // prev_node = prev_node.next;
9    // 型不一致エラー
10   // prev_node = prev_node.next.unwrap();
11   prev_node = prev_node.next.as_mut().
         unwrap();
12 }
```

(1) はソースコード 1 の 2 行目と同様な記述としたソースコード 2 の 8 行目や，取り出し処理のみを追加した 10

行目の代入文における行き詰まりである．エラー発生時のコンパイラのメッセージを図 1 に示す．

```
error[E0308]: mismatched types
  --> scenarioEx.rs:68:21
   |
64 |     let mut prev_node: &mut Node = list;
   |                        --------- expected due to this type
...
68 |         prev_node = prev_node.next;
   |                     ^^^^^^^^^^^^^^ expected `&mut Node`, found enum `Option
`
   |
   = note: expected mutable reference `&mut Node`
                        found enum `Option<Box<Node>>`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0308`.
```

図 1　ソースコード 2 の 8 行目のエラーメッセージ

エラーメッセージは型不一致エラーが発生したことを報告し，代入の左辺と右辺の型情報を表示する．しかし，具体的な解決方法は提示されず，学習者はどのメソッドを呼び出せば型を合わせられるかわからない．10 行目の代入文は，列挙型の中身の型を利用するには取り出し処理が必要という前提知識で，簡潔な記述の unwrap メソッドを呼出している．しかしこの代入文でも図 1 と同様の型不一致エラーが発生し，学習はどのメソッドを呼び出せば型を合わせられるかわからない．

(2) はコンパイラのメッセージからムーブ不可の制約条件下でムーブが発生している状況が読み取れない行き詰まりである．列挙型からのオブジェクト取り出しはムーブが発生するが，Rust では参照，借用されているオブジェクトをムーブすることや，構造体のメンバをムーブすることは所有権のルールを満たさなくなるために禁止されている．この制約条件の違反は 8 行目や 10 行目で発生しているが，図 1 のエラーメッセージは型不一致エラーのみ指摘しており，制約条件の違反は読み取れない．

4　提案方法

4.1　概略

問題分析の結果，学習者が列挙型のメソッドがわからないことと，ムーブ不可の制約条件下でムーブが発生している状況が理解できないことが行き詰まりの原因とわかった．本研究では問題 (1)，(2) を解決する方法としてそれぞれ次を検討する．

(1) エラーとなるメソッドを含めた，列挙型のメソッド候補の提示
(2) メソッド候補を適用した代入文における，オブジェクトの所有状況の可視化

ノードをオブジェクトの所有状況の可視化図，エッジを代入文で追加する列挙型のメソッド候補とした，メソッド候補提示グラフを学習者に提示する．正しい場合の所有状況とエラーの場合の所有状況を学習者が比較可能にすることで，所有権ルールの理解を目指す．

4.2　所有状況の可視化図

問題 (2) 解決のために，メソッド候補提示グラフのノードで表示する可視化を定義する．可視化に必要な要素の一覧を図 2 に示す．

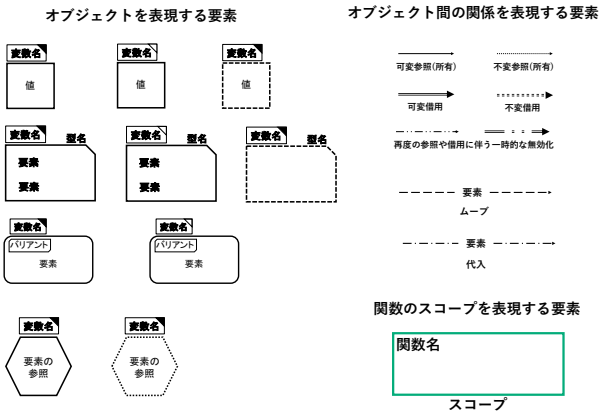

図 2　可視化図の要素一覧

可視化図の要素はオブジェクトを表現する要素と，オブジェクト間の関係を表現する要素，その他の要素の 3 種類を定義した．オブジェクトを表現する要素は次で表現する．

- 型を図形の形で表現
- 変数がオブジェクトを所有する場合は変数名を左上に記述
- 制約条件を変数名右上の塗りつぶしで表現

図形の違いで型を表現し，列挙型でラップされた型の取り出しを表現する．変数名右上の塗りつぶしはオブジェクトの参照，借用に伴いオブジェクトを変更する権利が凍結される場合に白に変更し，制約条件を表現する．

オブジェクト間の関係を表現する要素は．関係の種類ごとに異なる矢印を用いて次の要素で表現する．

- 矢印の始点を参照するオブジェクト，終点を参照されるオブジェクトとして所有関係を表現
- 参照と借用はそれぞれ 1 重線，2 重線で表現
- 参照と借用の可変，非可変の違いは実線，破線で表現
- ムーブと代入は矢印の間にムーブ，代入されるオブジェクトの要素を記述
- ムーブを破線，代入を点を含む破線で表現
- ムーブが発生する代入はムーブの矢印を記述し，ムーブが発生しない場合は代入の矢印を記述

ムーブ，代入の矢印はオブジェクトが代入によってどの型のオブジェクトが移動するか記述し，ムーブによる所有者の変更を表現する．

関数のスコープを表現する要素は，関数呼び出しでそれぞれの関数のスコープと，スタック領域にあるオブジェクトとヒープ領域にあるオブジェクトの区別を表現する．

定義した要素でソースコード 2 の 8 行目のエラーとなる代入文と，11 行目の正しい代入文の所有状況を可視化した例をそれぞれ図 3 と図 4 に示す．

図 3 では型不一致と，ムーブ不可の制約条件下でムーブ

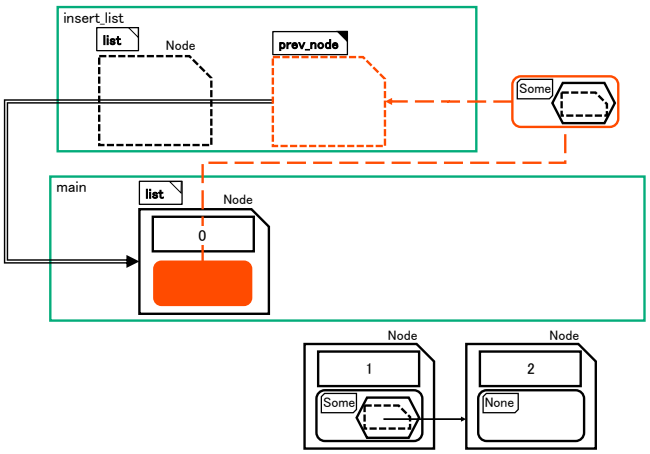

図 3　ソースコード 2 の 3 行目の所有状況

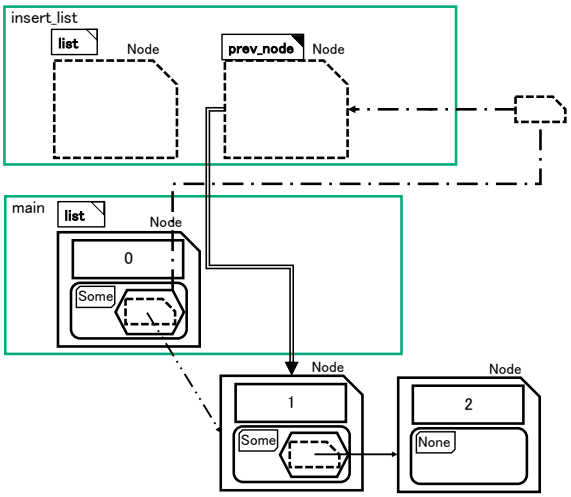

図 4　ソースコード 2 の 6 行目の所有状況

が発生している状況を表現している．型不一致はムーブの矢印で移動されるオブジェクトとムーブ先のオブジェクトの形が一致しないことから，枠線を赤色で記述してエラーを表現する．制約条件に違反するムーブは，ムーブの矢印を赤色で記述し，ムーブしようとするオブジェクトの存在した場所に赤色で塗りつぶした同じ形の要素を記述することで，エラーを表現している．

4.3　メソッド候補提示グラフ

4.3.1　変更候補の抽出

問題 (1) 解決のために，メソッド候補提示グラフのエッジとして学習者に呼び出して追加する列挙型のメソッドを提示する．メソッド候補提示グラフの対象とする代入文における理解支援に必要なメソッドの抽出方法を提案する．C と同様の記述のオブジェクト取り出しを行う代入文からコンパイルエラーを解消するには，as_mut メソッド等で参照を取った後に，unwrap メソッド等でオブジェクトを取り出す必要がある．メソッド候補提示グラフで必要となるメソッドは，標準ライブラリで定義されたジェネリック列挙型 Enum<T>に実装されたメソッドのうち，列挙型そのものの型を変換する変換系メソッドと，列挙型の中身のオブジェクトを取り出し，その型で使用可能にする取り出し系メソッドである．変換系メソッドと取り出し系メソッドの抽出方法として，メソッドの入出力の型を基準とする方法を提案する．抽出するメソッドの要件は次である．

- 変換系メソッドの呼出しを追加する必要がある場面で，エラーとなる別の変換系メソッド呼出しの追加
- 変換系メソッドの呼出しを追加する必要がある場面で，取り出し系メソッド呼出しの追加

1 つ目の要件は，エラーとなるメソッドの提示によって，可視化図と合わせてコンパイルエラーになる状況を理解させることを目的とする．2 つ目の要件は，学習者が列挙型からの取り出し処理が必要なことは理解し，取り出し系メソッドのみを適用する状況を想定したものである．ソースコード 2 の 3 行目の代入文を対象として，Option 型のメソッドを抽出した結果を表 1 と表 2 に示す．

表 1　抽出した変換系メソッド

メソッド名	引数の型 (self)	返り値の型
as_ref	&Option<T>	Option<&T>
as_mut	&mut Option<T>	Option<&mut T>
as_deref_mut	Option<T>	Option<&mut <T as DerefMut>::Target>

表 2　抽出した取り出し系メソッド

メソッド名	引数の型 (self)	返り値の型
expect	Option<T>	T
unwrap	Option<T>	T

4.3.2　メソッド候補提示グラフの例

4.2 節で定義した要素を用いた可視化図と 4.3.1 節で抽出したメソッド群を用いてメソッド変更候補提示グラフを作成する．ソースコード 2 の 8 行目の C と同様の記述とした代入文を対象とした変更項候補提示グラフを図 5 に示す．

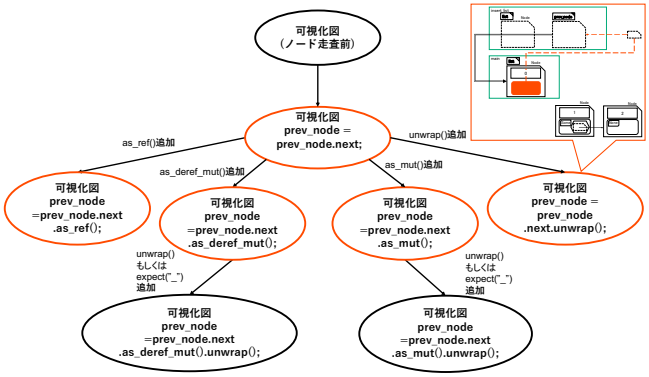

図 5　ソースコード 2 の代入文を対象としたメソッド候補提示グラフ

メソッド候補提示グラフは学習者が対象の代入文で行き詰まりが発生したと感じた際に，学習者に提示される．枠

線が赤色のノードはコンパイルエラーとなる状況の可視化図であり，枠線が黒色のノードはコンパイル可能な正しい状況の可視化図である．学習者は対象の代入文実行前の可視化図と，現状の C と同様の記述とした代入文の可視化図を確認し，ムーブ禁止の制約条件に違反する所有状況を確認する．エッジのメソッドを確認し，現状の代入文に必要な変換系メソッド，取り出し系メソッドを確認する．正しくコンパイルできる記述となるメソッドと，エラーとなるメソッドを呼出して追加した場合の所有状況を，各メソッドを追加した代入文のノードから可視化図を確認する．正しくコンパイルできる記述の代入文の可視化図と，コンパイルエラーとなる代入文の可視化図を見比べることで，所有権のルールに基づく制約条件によってエラーが発生することを確認することができる．メソッド変更候補提示グラフは学習者が対象の代入文で行き詰まりが発生したと感じた際に，学習者に提示される．

5 考察

5.1 可視化図の評価

本研究で提案した可視化図の要素について，理解支援として所有権ルールに基づく制約条件が可視化可能か，図 3 と図 4 の可視化例を用いて考察する．可視化図で表現するものとして以下がある．

- 型の不一致
- 列挙型のオブジェクト取り出しによるムーブ
- 参照，借用されているオブジェクトのムーブ禁止

型の不一致は，代入する際に代入元と代入先の要素の形で表現する．図 3 の例ではムーブされる要素がムーブ先の要素と形が異なっていることから型不一致を確認できる．また，要素の枠線を赤色とすることでエラーを表現している．

列挙型のオブジェクト取り出しによるムーブは，図 3 の例ではムーブされる要素がどこからどこに移動するかを矢印でムーブされる要素を囲むことで表現している．図 4 の例ではムーブが発生しない代入をムーブとは異なる矢印で表現することで，ムーブの発生する場合と発生しない場合を区別して表現可能である．

参照や借用されているオブジェクトのムーブ禁止について，オブジェクトが参照や借用されていることは変数名右上の塗りつぶしで表現する．ムーブされる要素が参照や借用されているオブジェクトの場合，ムーブ不可であることを元の位置を要素の形で赤色に塗りつぶすことで，ムーブ不可の要素をムーブしていることを表現している．以上によって参照，借用されていること，参照，借用に伴うムーブ不可の制約条件が表現可能である．

5.2 既存技術との比較

本研究で提案する可視化と既存技術の比較として RustViz と，オブジェクト間の関係の可視化を目的とすることから UML のオブジェクト図の仕様に沿った可視化図との比較を考察する．RustViz はムーブによる所有者の変更は表現可能だが，具体的にオブジェクトやその型の表現できず，オブジェクト同士の関係や型の明示による列挙型からの取り出しは表現が難しく，コンパイルエラーとなるプログラムには対応していない．オブジェクト図の仕様に沿った可視化図はどの変数がどのオブジェクトを所有しているかを表現可能で，型情報を文字で記述できる．しかし，ムーブや代入といった動的な表現が難しい．

5.3 適用範囲

適用範囲の考察として，Option 型と同じく標準ライブラリに定義されたジェネリック列挙型 Result からのオブジェクト取り出しを行う代入文へ提案手法の適用を試みた．対象の代入文は Result 型から動的文字列オブジェクトを取り出すものとした．この代入文ではオブジェクト取り出しに伴うムーブと借用されたオブジェクトのムーブ不可の制約条件が衝突が発生しエラーとなる．エラーの解消には参照を取った後に取り出しを行う．提案方法の適用により Result 型の変換系メソッドと取り出し系メソッドを抽出した結果，4.3.1 節で述べた抽出するメソッドの要件を満たすメソッドを抽出できた．可視化図については，動的文字列型を表現する要素を新たに定義することで作成可能である．したがって，対象の代入文のメソッド補提示グラフが作成可能であり，Result 型からオブジェクト取り出しに提案方法は適用可能である．

5.4 妥当性への脅威

妥当性への脅威として，提案方法の対象として想定する学習者の用意が難しく，提案方法のユーザビリティ評価が実施できていない点がある．代替方法として，オンラインコミュニティにて質問されている Rust に関する問題の中から，本研究で対象とする列挙型からのオブジェクト取り出しの問題と同様の事例を探し，提案方法の適用と，適用による問題の解決が可能か評価する方法が考えられる．

6 おわりに

本研究では，C 既習者が Rust を学ぶ際の行き詰まりの 1 つとして，列挙型からの値取り出しに伴うムーブと制約条件の衝突が理解しづらいことを問題とした．解決に向けた支援方法として，列挙型のメソッド呼出しの提示と各メソッド適用後の所有状況の可視化図を組み合わせたメソッド候補提示グラフを提案した．今後の課題はユーザビリティの評価と，他の事例への適用実験，メソッド抽出や可視化の自動化が挙げられる．

参考文献

[1] M. Almeida et al. : RustViz: Interactively Visualizing Ownership and Borrowing, 2022 IEEE Symposium on VL/HCC, 2022, pp. 1-10

[2] M. Coblenz et al. : Garbage collection makes rust easier to use: a randomized controlled trial of the bronze garbage collector ICSE ’22, pp. 1021-1032.