OCaml 初学者の学習傾向分析

理学専攻 情報科学コース 1940643 北川舞 (指導教員：浅井 健一)

1 はじめに

プログラミング初学者がどのようなプロセスを経てプログラムを書くのかということは、あまり理解されていない。そこで、2015 年から 2020 年における OCaml 初学者のプログラミング学習データを解析し、初学者のプログラミング行動について分析を行なった。

コーディングの進捗具合、コンパイル結果ごとのエラー状態やエラー原因を探ることで、初学者がどのように学習を進めるのか、また、どのようなプログラミング概念が初学者を悩ませているのか明らかにする。また、ビジュアルプログラミングエディタ OCaml Blockly[3] が初学者に与えた影響についても、データに基づき考察する。ここで得られた知見は、今後のプログラミング教育と教育支援環境の作成において、改善すべき問題点や姿勢を考える際に有用であると考える。

1.1 分析対象

分析には、お茶の水女子大学で毎年 15 週かけて行われる初歩的な関数型プログラミングの授業から、2015 年から 2020 年にかけて得られたデータを用いた (2020 年度のみ 14 回)。履修者は情報科学を専攻する学部 2 年生毎年約 40 名であり、この授業で初めて関数型プログラミング言語 OCaml を学習する。

テキストベースの OCaml プログラミング実行データ全 149831 件 (2015 年-2019 年[1])、OCaml Blockly を使用したプログラミング実行データ全 155824 件 (2020 年) が取得され、分析対象となった。

2 学生のプログラミング行動

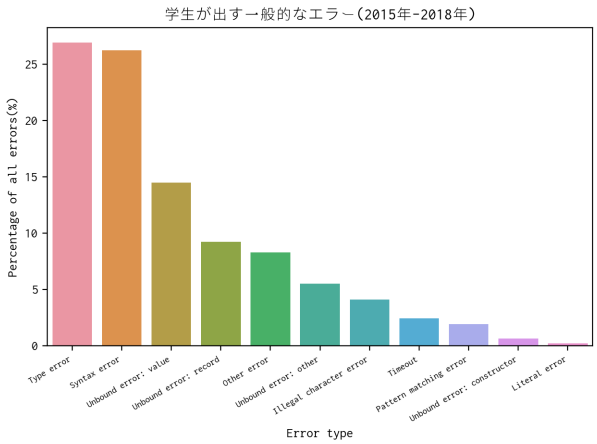

図 1: 学生が出す一般的なエラー

2015 年から 2018 年に実行されたプログラム 124441 件を OCaml コンパイラの出力するエラーメッセージに基づき、OCaml でプログラムを学ぶ際に最も頻繁に遭遇するエラーを調べた。様々なエラーのうち、最も一般的な 4 つのエラーがプログラミング中に学生が発生させたエラーの 76.9%を占めていて、その内訳は Type error(26.9%)、Syntax error(26.2%)、Unbound error: value(14.5%)、Unbound error: record(9.3%) となった (図 1)。

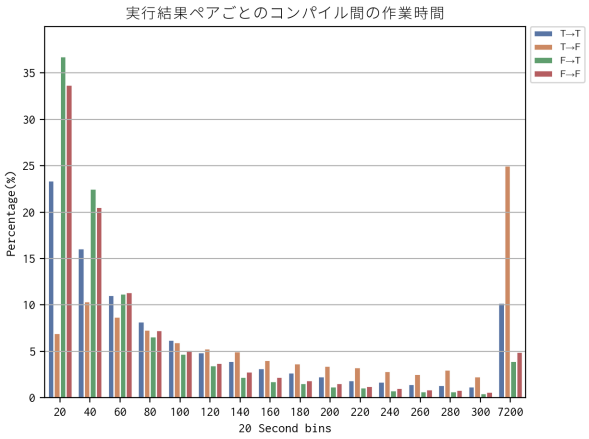

図 2: プログラムペアごとのコンパイル間の作業時間

再コンパイルするまでの作業時間について、構文的なエラーを含むプログラム (F) と構文的なエラーを含まないプログラム (T) のペアごとに割合を算出した (図 2)。直前のプログラムが F である時、学生は短時間で再びコンパイルする傾向にあり、68.1%の割合で 1 分以内に再コンパイルする。また、直前のプログラムが T であるとき、それに続いて行うプログラム変更作業には時間を費やす傾向があった。

これらの観察から、学生がエラーメッセージを読みプログラムを編集し再びコンパイルするという一連の作業に対し、あまり時間を費やさないことがわかる。また、エラーを修正する作業に比べて、構文的に正しいプログラムから新たなプログラムを書き足す作業の方により時間をかける傾向にある。この結果は Java による先行研究 [2] の結果と類似しており、言語によらず初学者行動の一傾向を表していると考えられる。

3 難しいプログラミング行動

初学者がプログラミングに苦戦する箇所を理解するためには、主なプログラミング概念の難しさと、その難しさが授業課題に取り組んでいく中でどのように変化するかを理解する必要がある。そこで、週ごとに発生するエラーと修正に 5 分以上の時間を要するエラーについて調査した。

3.1 起こりやすいエラーと難しいエラー

Syntax error、Unbound error、Type error は全ての週で一定割合発生していた。Syntax error と Unbound error は、発生件数割合に対して修正に時間を要するエラーの割合は小さく、起こりやすさのわりに解消に時間を要していなかった。一方、Type error は、発生件数割合のわりに修正に時間を要するエラーが多い様

[1]2019 年度は一部の学生が OCaml Blockly を使用してプログラムを作成した。しかし、実行環境の問題でテキストベースのデータ取得形式であることには変わりないため、こちらに含めている。

子が伺えた。また、修正に時間がかかったエラーのみで割合をみると、大半の週で Type error が修正に時間のかかるエラーとして最も割合を占めていた。

発生させてしまうものの解消しやすいエラーがあるのに対し、初学者にとって Type error は修正するのが難しい傾向にあるエラーであると考えられる。

3.2 Type error の発生原因

Type error の修正が難しい原因について目視で確認した。調査の結果、課題の内容によらず、2 つの修正に手間取る主な原因が確認できた。

1 つ目はデザインレシピ [1] の考え方から生まれるエラーで、学生が始めに作成するテストケースとその後作成する関数本体の型の不一致によって発生する。例えば、2 つのベクトルの足し算を行う関数 add を作成したいとする。デザインレシピにしたがって学生はまず、関数があると仮定し以下のように関数の目的とテストケースを作成する。

```
(* 目的: 2次元ベクトルの足し算 *)
let add vec1 vec2 = ... (* 作成前 *)
(* テスト *)
let test1 = add (1, 2) (3, 4) = (4, 6)
let test2 = add (-1, 5) (7, 3) = (6, 8)
```

この作業により、学生は作成したい関数の目的と方向性を把握する。その後学生は関数本体の定義に入るが、例えば、match 文の構文を理解できていないと、次のような誤ったプログラムを作成することがままある。

```
(* 目的: 2次元ベクトルの足し算 *)
let add vec1 vec2 = match vec1 vec2 with
  (x1, y1), (x2, y2) -> (x1+x2, y1+y2)
(* テスト *)
let test1 = add (1, 2) (3, 4) = (4, 6)
let test2 = add (-1, 5) (7, 3) = (6, 8)
(* 発生するエラーメッセージ:
   Error: This expression has type 'a * 'b
     but an expression was expected of type '
     c -> (int * int) * (int * int) *)
```

この時、関数 add だけで見れば構文上のエラーはなく定義可能である。ただ本来、整数のペアを想定していた引数 vec1 が vec2 を引数とするような関数になってしまっている。しかし、テストケースを事前に書くことで、このプログラム全体を実行した時上記のようにエラーメッセージが出力される。結果として、学生は作成関数の目的を明確にし、意味的に異なる関数を作成する失敗をせずに済むが、今回の調査から、Type error の読み解き自体に苦労していることがわかった。

2 つ目は構文の各節 (例えば、if 文の then 節と else 節) での型の不一致が原因で、初学者がプログラムの構造把握に難しさを感じていることがわかった。

初学者にとって、プログラムの構造やデータ構造を把握すること、また、それぞれのデータ構造を表す構文を適切に書き表すことが難しいポイントであることがわかった。困難を減らすためには、これらの構造を初学者にわかりやすい形で可視化しプログラミングを補助することが有用であると考えられ、OCaml Blockly が有用である可能性がある。

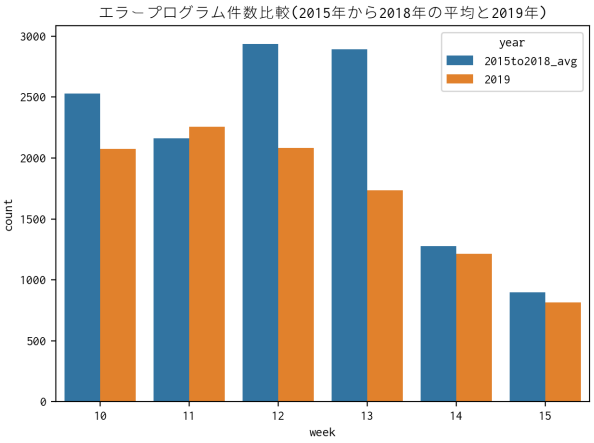

図 3: エラー発生件数 (2018 年までの平均と 2019 年)

4 OCaml Blockly の効果と影響

OCaml Blockly を使って課題関数を作成できた第 9 週までの課題について、課題解答時間中央値を 2018 年までのデータと 2020 年のデータで比較したところ、殆どの課題で 2020 年のデータが短い結果となった。

OCaml Blockly が使えなくなった第 10 週以降のエラー発生件数を、2015 年から 2018 年までの平均と 2019 年とで比較すると、第 11 週が若干増加しているものの各週同程度以下となっていた (図 3)。これは OCaml Blockly がその後のテキストプログラミングに悪影響を与えないという一検証になっていると考えられる。

5 まとめ

OCaml 初学者がどのようにプログラミング行動を進めるのか把握した。学生は長時間エラーを前に悩み続けることは少なく、比較的短時間で再コンパイルする様子が確認できた。Syntax error や Unbound error は、発生件数の割に修正に時間を要するエラーが少ない一方、長い時間をかけて修正を行うエラーは Type error であることが多く、データ構造を表す構文やプログラム全体の構造が把握できていないことが原因で発生するものが一定の割合を占めた。

2020 年度の OCaml Blockly を使用した学生のデータと、従来のテキストプログラミングで学習を行なった学生のデータを比較したところ、OCaml Blockly を使用した学生がある程度短時間で課題を完成させた様子が確認できた。また、2019 年度は一部の学生が初期に OCaml Blockly を使用したが、使用者のその後のテキストプログラミングデータからは、最初からテキストプログラミングで学習した学生に対して、プログラミング結果が悪化している様子は確認されなかった。

参考文献

[1] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs: An Introduction to Programming and Computing*. The MIT Press, 2018.

[2] Matthew C. Jadud. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*, Vol. 15, pp. 1–25, 2005.

[3] 松本晴香, 浅井健一. Blockly をベースにした OCaml ビジュアルプログラミングエディタ. 第 21 回プログラミングおよびプログラミング言語ワークショップ論文集, 2019.