モジュールをサポートするインクリメンタルステッパの設計と実装

理学専攻　情報科学コース　2140643　秋山 雛乃　(指導教員：浅井 健一)

1　はじめに

プログラマがプログラムを理解する方法として、プログラムを 1 ステップごとに書き下す方法がある。しかし、初学者にとって自分の手でプログラムの書き下しを行うことは困難である。そこで、プログラム実行の流れを書き下すステッパというツールが開発された。本研究では、OCaml を対象言語としたステッパであるインクリメンタルステッパ [2, 第 5 章] にモジュール宣言一般と参照の構文をサポートすることを目指す。インクリメンタルステッパとは、入力として型チェックの通ったプログラムを受け取り、プログラムの計算過程を 1 ステップごとに出力する OCaml ステッパ [1] をもとに開発されたステッパであり、使用者は普段通りプログラムを行い、エディタ上のボタンを操作することで、プログラムがステップ実行していく流れを確認することができる。モジュールと参照は実際にお茶の水女子大学の関数型言語の授業で扱われる構文であり、初学者用のステッパとして対応する必要があると考える。また、実装レベルで網羅的にモジュールをサポートすることは難しかったため、本研究では OCaml ステッパの定式化に取り組み、その定式化を元に実装を行った。

2　モジュール構文のサポート

モジュールをサポートするために問題となった点は、モジュールの内側の値や式をモジュールの外側で使用する際、モジュールの内側で宣言されているものなのか外側で宣言されているものなのか区別できなかった点である。この問題に対して本研究ではいくつ外側のモジュールで宣言されたかを表すレベルを OCaml の構文木に attribute として付与することで導入し、そのレベルを変化させることで解決した。また、レベルは多重モジュールに対応するため 0 以上の整数で導入した。

例えば、以下のプログラムを実行するとき、

```
type t = C
module X = struct
    let a = C
    type t = C
    let b = C
end
let b = X.b
let a = X.a
```

3 行目の `let a = C` のコンストラクタ `C` は、1 行目で宣言されているコンストラクタであるため、一つ外側で宣言されていることを表すレベル 1 が付与される。また、5 行目の `let b = C` のコンストラクタ `C` はモジュール `X` の中で宣言されているため、レベル 0 が付与される。このような情報が付与されることでモジュールの中身が代入される際区別がつくようにした。

そして、5 行目の実行が終わるとモジュール `X` の中身の実行が終わりとなるため、モジュール `X` の中身である 3 行目から 5 行目の情報を 7 行目以降のプログラムへ代入しにいく。7 行目の `X.b` には、モジュール `X` の中で宣言されたレベル 0 のコンストラクタ `C` がモジュール `X` の外側で使用されるため、`X.` が頭についた `X.C` が代入される。本研究では、このようにモジュールの中身をモジュールの外側へ代入することを「モジュールが持ち上がる」と定義し、その際、必要に応じてモジュール名をつけることを「レベルを持ち上げる」と定義し、lift 関数として定式化を行った。

次に 8 行目の `X.a` には、モジュール `X` の内側にある 3 行目のコンストラクタ `C` が代入される。この時、レベル 1 のコンストラクタ `C` はモジュール `X` の外側で宣言されているため、レベルが持ち上がっても `X.` が付与されない。これはレベル 1 からレベルが持ち上げられレベル 0 になり、宣言された場所と同じ階層に持ち上げられたため、`X.` は付けないという判断をすることで実装する。そのため、1 行目が実行されモジュール `X` の中へコンストラクタ `C` の情報を付与する際、一つ上のモジュールで宣言されていることを表すため、レベルを 1 加算する必要がある。この操作を「レベルを持ち下げる」と本研究では定義し、drop 関数として定式化を行った。また、モジュールの外側で宣言されているプログラムの情報をモジュールの内側へ代入することを「モジュールを持ち下げる」と本研究では定義する。

3　OCaml ステッパとインクリメンタルステッパ

OCaml ステッパの内部では、インタプリタにステップ出力機能を追加したものに対して、入力プログラムを渡し、全てのステップの文字列を生成する。そして、プログラムを全て実行するか、ステップ数の上限に達すると実行を終了し、ユーザには最初のステップを表示する。その後ユーザの操作に従って表示を行う。

OCaml ステッパはプログラムの実行が終わるまでユーザへの出力ができないため、長い OCaml プログラムを入力した際に実行時間が長くなることや、無限ループをするプログラムだった場合、（ステップ数を適当なところで打ち切るなどをしないと）ステッパを使用できない課題があった。無限ループに対処する際、同じ式が無限に繰り返される様子を観察できるなど、ステッパは無限ループの解消にも有用なツールであると考え、OCaml ステッパと表面上は同じ動作をするが、内部での処理方法や速度が大きく異なるインクリメンタルステッパが開発された。

インクリメンタルステッパは、インタプリタにステップ出力機能を追加したものに対して、入力プログラムを渡し、1 ステップの簡約を計算したら直ちにそのステップを出力する。出力後は続きを実行するのではなくプロセスを終了する。ユーザから次のステップを表示するなどの命令が指示され次第、前回の出力の一部を新しい入力として受け取り次の 1 ステップを実行する。

この時、インクリメンタルステッパが次のステッパに渡すプログラムは完全なプログラムである必要性と、コメントとして書かれているステップ数に関しても次のステッ

プに引き継ぐ必要性に注意する。インクリメンタルステッパは 1 ステップごとに結果を出力するとプロセスを終了する。そして次のステップを実行する際、その前のステップで出力されたプログラムを OCaml コンパイラで構文解析して読み込む。そのため、ステッパによって出力される OCaml の式が OCaml コンパイラで構文解析可能である必要がある。また、ステップ数に関してはユーザに見せる際にはコメントで出力しているが、コメントは次のステップで読み込む時 OCaml コンパイラには無視されてしまう。コメントを構文解析するために新たにプログラムを書くことを避けるため、インクリメンタルステッパでは上手にステップ数と次のステッパの入力となるプログラムを OCaml の attribute という構文を用いてプログラムに付与している。

4 参照サポートの問題点

インクリメンタルステッパで以下のプログラムをステップ実行すると、

```
let a = ref 1
let b = !a
```

まず以下の 1 ステップを出力する。

```
(* Step 0 *) let a = ref 1
(* Step 1 *) #1: 1 let a = (#1)
```

この時、2 行目の式の前にある「`#1: 1`」はその時点のロケーション状況とそれに対応する値を表している。そして式の中で青くなっている「`#1`」はロケーション状況を表している。その後、ユーザから次のステップを表示する命令がきたとき、以下のプログラムがステッパの入力となる。

```
let a = #1
let b = !a
```

この時点で多くの問題が発生する。まず、このプログラムは `#` の部分でシンタックスエラーが起きてしまう。これは、3 節で説明した、次のステップの計算を行う時の入力プログラムが完全なプログラムになっていないことが原因である。これが、一つ目の課題である、ロケーション状況を入力プログラムとして OCaml で読み込めない点である。次に、たとえ `#1` という表記をプログラムとして読み込み可能であり、参照の意味を有したデータにできたとしても、`Step 2` から `Step 3` にかけて行われる参照の中身の取り出しの動作のとき、`#1` という参照の参照先の値が 1 であるという情報が引き継がれていないため、中身を取り出すことができない。これは、3 節で示した次のステップへのプログラムを attribute として残す際に参照と参照先の値の情報が attribute として付与されていないことが原因である。これが二つ目の課題である、インクリメンタルステッパにしたことで、全ての状態を出力しなければいけない点である。

5 任意の型を持ちうる式への変換

一つ目の課題であるロケーション状況を入力プログラムとして OCaml で読み込めない点に対して本研究では、ロケーション状況を

- OCaml で読み込み可能
- 任意の型を持ちうる
- 参照の情報を attribute として持つ

ようなデータとして表現する。参照に対応する値は整数や文字列、ペア、リストなど様々な型を持ち得る。そのため、ロケーション状況を別の式で表現する時、その式は任意の型を持ち得る式でなければならない。そこで本研究では、任意の型を持ち得る `assert false` に参照情報という特別な attribute がついた式を用いた。この時、ユーザには `assert false` ではなく、インクリメンタルではないステッパの出力同様の「`#`」と「参照の生成が行われた順番」の表記が見えるようにする処理を行なっている。このような実装を行うことで、ユーザへの表示は OCaml ステッパと変わらず、ロケーション状況をインクリメンタルに扱うことができるようになった。

6 参照情報の引き継ぎ

OCaml ステッパではグローバル変数として管理されていた参照情報を attribute としてプログラムに付与し、そのプログラムを入力として受け取った時に、参照情報を獲得し、新たな参照をグローバル変数に格納する実装を行った。このような実装を行うことで、OCaml コンパイラのパーザをそのまま利用しつつ、参照情報をインクリメンタルに扱えるようになった。

7 まとめと今後の課題

本研究では、インクリメンタルステッパにモジュール宣言一般と参照の構文をサポートするため、以下を行った。

- モジュール宣言の場所を表すレベルの導入
- レベルを含めた OCaml ステッパの定式化
- ロケーション状況を任意の型を持ちうる式に参照の情報を付与したものに変換
- 参照情報をプログラムに attribute として付与

現在、インクリメンタルステッパは本研究で実装したモジュールと参照の他に、`if` 文や、型定義、`match` 文、引数付きの例外など初学者が扱いそうな構文（配列についてはまだサポートできていないが、本研究の参照のサポートをもとに対応予定）についてサポートしており、お茶の水女子大学で行われている関数型言語の授業に活用されている。授業ではブロック環境 [3] と Emacs 環境の両方で実際に使われている。また、ステッパのデモページ（`http://pllab.is.ocha.ac.jp/~asai/Stepper/demo/`）も公開している。今後は、ソースコードの整理をした上で公開し、より広く使えるようにしていく予定である。

参考文献

[1] Tsukino Furukawa, Youyou Cong, and Kenichi Asai. Stepping OCaml. *In Proceedings Seventh International Workshop on Trends in Functional Programming in Education, Chalmers University, Gothenburg, Sweden, 14th June 2018,*, Vol. 295 of Electronic Proceedings in Theoretical Computer Science, p. 17–34, 2019.

[2] 古川つきの. 代数的効果を含むプログラムのステップ実行. Master's thesis, お茶の水女子大学大学院　博士前期過程　人間文化創成科学研究科, 2020.

[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.