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

理学専攻・情報科学コース 1740664 松本晴香

1 はじめに

関数型言語の初学者が関数型言語を学び始めたとき，数々の本質的でない問題に陥りやすい．変数に何らかの値を再代入しようとしてシンタックスエラーを起こしたり，Int 型と Float 型を持つ値を足し合わせようとして型エラーになってしまうことがある．

プログラミング初学者向けのツールの 1 つに，ビジュアルプログラミング言語がある [1, 2, 3]．本研究では，ブラウザにおけるビジュアルプログラミング環境構築のための JavaScript ライブラリ Blockly [4] をベースにして，OCaml ビジュアルプログラミング環境を実装した．本システムでは，各ブロックが持つ型を形や色で視覚的に表現する [5] [1]．また，以下の 2 点を最終的な目標とする．

1. コンパイルエラーが起きるプログラムの組み立てを制限したユーザインタフェースを実装する．
2. 関数型言語初学者向けの授業で使えるように，豊富な構文を揃え，「使いやすさ」「理解しやすさ」といった高いユーザビリティを実現する．

関数型言語の初学者が，本システムを使用することによって，以下の利点を得られることを期待している．

- OCaml のシンタックスや型付けに慣れる．
- 自身が組み立てようとするプログラムが不正なものだとに瞬時に気づけるため，テキストベースよりもストレスが少なく，OCaml プログラミングの本質を優先的に学習できる．

また，本システムによって OCaml の言語仕様に慣れ親しんだのち，シームレスにテキストベースによるプログラミングに繋がるように，本研究ではブロックと OCaml コードの相互変換の実装も行った．

2 主な機能

本研究で行った主な実装のうち，4 つの機能をそれぞれ説明する．

2.1 変数束縛

初学者が OCaml におけるスコープの概念を理解しやすいように，変数参照は必ず変数宣言から生成させる (図 1)．これによって未定義の変数を参照することが原理的に起きえなくなる．Unbound value エラーを防ぐために，ドラッグ中のブロック内の変数のスコープを管理する (図 2)．また，一度生成した変数参照はその参照先を配置された位置によって変えない (図 3)．これによりユーザの想定していなかったシャドウイングによってプログラムの意図が変わることを防ぐ．

例えば，図 2 でユーザが組み立てようとしているプログラムは，左から順に「`(let foo = ? in ?); foo`」，「`let foo = foo in ?`」，「`let foo = ? in foo`」であるが，初めの 2 つはプログラムとして正当でない．

[1] プリミティブ型ごとに別の形で表し，パラメタ付き型は再帰的に描画し，型変数は固有の色でハイライトする．

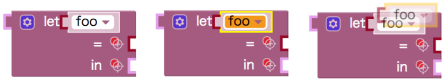

図 1. 変数ブロックを生成する流れ．変数宣言を囲うように描画されたブロック (左) にカーソルを合わせて (中) ドラッグする (右) と，変数ブロックの生成を行うことができる．

図 2. 変数ブロック「`foo`」を生成後にドラッグしたまま動かしたときのブロックの様子．正しく変数ブロックが束縛されない場合はブロックを不透明に表示する．

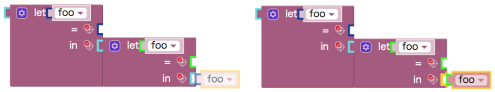

図 3. 外側の変数宣言部分から生成した変数ブロック (左) と，内側から生成した変数ブロック (右) を内側の in 以下に接続させようとしたときの比較．

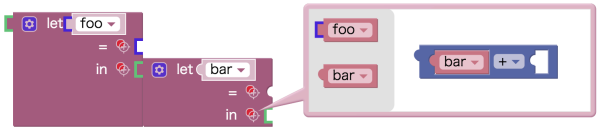

図 4. スコープお砂場の例．ライフルのスコープを模したアイコンをクリックするとポップアップが現れる．スコープから参照できる変数ブロックの一覧が見れ，ブロックの生成も行える．お砂場内で，変数ブロックをどんな型として扱うかによって，変数宣言のブロックの型も変化する．

よって，ドラッグ中のブロック全体を不透明にして，ブロックの移動や接続がプログラムとして不正であることを，ユーザに伝える．もしこの状態のまま，ユーザがその場にブロックを落とした場合は，ブロックが生成されたばかりならばブロックを削除し，そうでないならドラッグ開始前の位置，状況に戻らせる．

また，変数をホバーしたときに関連した全ての変数をハイライトさせることで，束縛関係を表現した．スコープに従った変数名の変更も行うことができる．

2.2 スコープお砂場

変数ブロックを含むブロックをスコープが有効な場所にしか置けないという制約を設けると，ブロックを自由な場所に置き，好きな順番でブロックを組み立てられるという，本来の Blockly にあった利点を損なってしまう．これを避けるため，本システムでは，スコープチェックのついた遊び場を設け，これを「スコープお砂場」と命名した．使用例を図 4 に示す．お砂場の所属先のスコープから参照することができない変数を含むブロックは，お砂場に移動できないようになっている．

2.3 let 多相を含んだ型システム

let 多相 [6] を含む型システムを実装した．ブロックを外した際は，型の単一化の解除を行う．

2.4 メッセージ出力

不正なプログラムを組み立てられないユーザインタフェースを設けると，なぜ不正なのかをまだ理解できない初学者が困惑しうるため，ドラッグ中のエラーを説明するツールチップの表示を行った (図 5).

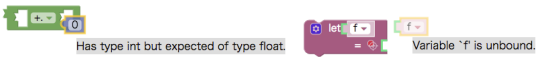

図 5. エラーツールチップの例.

3 スコープチェックの実装方法

ブロック上でのスコープチェックを実現するためには，変数の種類を「宣言される変数」と「宣言される変数を参照する変数」に分けると都合がよい．この 2 種類どちらかの変数をブロック内の変数出現部分に持たせ，スコープチェックを以下のように実装した．変数宣言を持つブロックに，スコープを作るコネクタ[2] を引数として，スコープ内で参照できる変数の一覧を返す関数を定義する．この関数を用いて変数環境を更新し，スコープ内のブロックに存在する全ての変数参照が変数環境にあるかを調べれば良い．ユーザがあるブロックを接続しようとした際は，接続を仮定したスコープチェックを行い，失敗したら接続を拒絶する．

4 ブロックと OCaml コードとの相互変換

本研究では，初学者がシームレスにテキストによるプログラムへ移行できるように，ブラウザ上でのブロックと OCaml コードの相互変換を実装した．

OCaml コードからブロックへの変換は，以下のように行っている．Blockly は元よりブロックと XML のエンコード，デコードをサポートしている．つまり，OCaml コードをブロック表現の XML に変換することができれば，ブロックを生成することができる．OCaml コンパイラのパッケージ compiler-libs を利用して OCaml コードから抽象構文木を得たのちは，木をトラバースして対応するブロックの XML を組み立てる．これらをビルドしたものを js_of_ocaml で JavaScript へと変換し，ブラウザ上で用いる．

5 現状と実使用に向けて

5.1 現状

現在本システムでサポートしている構文は，以下の通りである．使用例を図 6 に示す．

- `bool` 型，`int` 型，`float` 型，`string` 型.
- 2 つ組のみのタプル，リスト，ラムダ式，関数適用.
- 論理演算，数値演算，`fst` などのビルトイン関数.
- リストのパターンマッチを行う `match` 文.
- `in` を持つ `let`，`in` を持たない `let`，`let rec`.
- 1 引数のみのコンストラクタ定義.

本論文では，1 節で示した最終的な目標のサブセットとして，以下の目標を達成したと考えている．

1. シンタックスエラー，型エラー，Unbound value エラーが起きないことを保証する．
2. 元の Blockly の使いやすさを維持しつつ，OCaml に慣れるために必要な基礎的な構文を備える．

[2] ブロックとブロックを接続する切り欠きを表すオブジェクト.

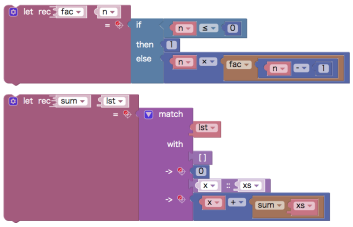

図 6. 本システムでブロックを組み立てた例．整数を受け取りその階乗を返す関数 fac (上) と，整数の入ったリストを受け取りその合計を返す関数 sum (下).

5.2 実使用に向けて

本論文では，最終的な目標の proof of concept となる目標を達成することができたと考えている．本節では，最終的な目標の達成のために必要な課題を整理する．

コンパイルエラーの起きない制約

標準の OCaml で起きうるコンパイルエラーは，本論文で対象としたエラーだけではない．OCaml におけるあらゆる種類のコンパイルエラーと一貫性を保つには，compiler-libs と js_of_ocaml を利用してブラウザ上で実際にコンパイルを行い，そのコンパイルエラーを流用することが確実である．この方法を実現するためには，組み立てた全ブロックとテキストによる OCaml コードの対応関係を的確に保たなければいけない．

ユーザビリティの点から

実使用に向けて，使い勝手の点から設計や実装が必要だと考える点は以下の 2 つである．

- OCaml の言語仕様に特化したブロックのデザイン.
- 型を視覚的に表示することによる限界.

まず始めに，ブロックのデザインを OCaml の言語仕様に特化したものに改造する必要がある．Blockly はどちらかというと命令型言語との方が相性が良く，関数型言語のような文より式が主な言語をそのまま載せると，元の Blockly にあった直感的なユーザ体験を損ないがちである．図 6 では，let ブロックの左側の余白部分が大きいために，代入や出力の関係が捉えにくい．

また，型が入れ子になって複雑になるほど，形や色による型の表現は理解しにくいものなってしまう．複雑な型は視覚的に表示することをやめて，ホバー時にツールチップの形で出力する，などといった策を取る必要がある．

6 まとめ

OCaml ビジュアルプログラミング環境を設計，実装し，実使用に向けての展望を述べた．今後は，ユーザ実験を行いフィードバックを得ながら，改良，拡張していきたい．

参考文献

[1] Viscuit. https://www.viscuit.com/.

[2] Scratch. https://scratch.mit.edu/.

[3] Snap! https://snap.berkeley.edu/.

[4] Blockly. https://developers.google.com/blockly/.

[5] Sorin Lerner, et al. Polymorphic Blocks: Formalism-Inspired UI for Structured Connectors. CHI '15.

[6] 五十嵐淳. プログラミング言語の基礎概念. サイエンス社, 2011.