ビジュアルブロックの自動生成を特徴とした ブロック言語処理系開発システムの提案

澤入圭佑[1,a)]　佐々木晃[2,b)]

概要：ブロックプログラミング言語は教育やロボット開発・アプリケーション開発など多分野で使用されている．しかし，開発言語やライブラリの理解など専門的な知識が必要であることや開発システムの操作が煩雑であるため開発に多大な時間がかかってしまう問題が存在する．本研究では，ブロックプログラミング言語処理系開発の支援の試みとして，①雛形ブロックから類似のブロックを作成できる支援機能②コード断片からブロックを提案する機能③ドキュメンテーションコメントからブロックを提案する機能を試作する．雛形ブロックから新しいブロックを作成できる機能を実装することで類似のブロックを作成する手間の削減や複雑なブロックの作成の簡易化を行い，既知の言語のコード断片からブロックを提案する機能を実装することでブログラミング言語の構文からブロックの候補を提案し作業の削減を図る．ドキュメンテーションコメントからブロックを提案する機能を実装することで一般的なブロックプログラミング言語の開発にかかる時間を削減する．

キーワード：ビジュアルプログラミング，ブロックプログラミング，言語処理系

1. はじめに

ブロックプログラミング言語とはプログラムをテキストで記述するのではなく，ブロック型の視覚的なオブジェクト（ブロック）を組み合わせることでプログラムが作成可能なプログラミング言語である．ブロックプログラミング言語はプログラム構造が理解しやすい点やエラーが起こりにくい点などの利点からプログラミング初心者にも扱いやすく，現在は主に教育やロボット開発・アプリケーション開発の場で利用されている．また今後，人工知能分野など他分野での開拓も期待される．しかし，ブロックプログラミング言語の開発には開発言語やライブラリの理解が必要である問題があり，また開発システムの操作が煩雑であることから多大な時間がかかってしまう問題が存在する．

ブロックプログラミング言語の開発における問題点として専門的な知識が必要である点，開発に多大な時間がかかってしまう点が挙げられる．ブロックプログラミング言語は GUI を含んだエディタとして実装される必要がある．これを実装するには言語を構成するブロックなどの GUI やブロックの言語処理系など，グラフィックス開発や言語処理系分野の知識が必要となってしまう．そのため Blockly[1] や Scratch Blocks[2] のようなブロックプログラミング言語開発用のライブラリを用いるのが一般的な開発方法となっている．しかしライブラリや開発言語の理解が必要となるため，開発の敷居は高

[1] 法政大学大学院情報科学研究科
[2] 法政大学情報科学部
[a)] 17t0014@cis.k.hosei.ac.jp
[b)] asasaki@hosei.ac.jp

くなっている．また Blockly ライブラリで実装されたブロックプログラミング言語開発用の DSL である Blockly Developer Tools[3] や JavaScript 上で Scratch[4][5] を再実装した Snap![6] では，システムで用意されたブロックを組み合わせることでより簡単にカスタムブロックの作成ができる．しかし言語を構成するブロックの数は大量であることが一般的であるとともに，これらのツールでは煩雑な操作性から多大な時間がかかってしまう問題がある．また DSL を用いてブロックを作成する際に，ブロック同士の単なる組み合わせのみでは表現できないブロックも存在しこれを作成するには直接プログラムを記述せざるを得ない．たとえば「if」ブロックを作成する際には，「else if」部や「else」部の有無や個数によって形を変えられる機能を利用者に提供する必要がある．このような複雑なブロックを作成するには，ライブラリに関する深い知識も必要になる．さらに，Blockly のようなツールを用いてブロックを作成する際には，ブロックを何らかのプログラミング言語に変換するための機構（変換器）も作成する必要がある．変換器はブロックが表現する言語構成を，変換先のプログラミング言語の構文の意味に沿う形に変換する必要がある．また開発者はブロックの実装を理解した上で変換器を作成する必要があり，前述した複雑なブロックのように構造が複雑なブロックの変換器の作成が困難であることが問題となっている．

本研究では，①雛形ブロックを用いて類似ブロックの作成を支援する手法②既知のプログラミング言語のコード断片からブロックを提案する手法③ソースコード内のドキュメンテーションコメントを基にブロックを提案する手法の 3 つの手法を提案する．また手法に基づいたシステムの実装を行い，本提案手法の効果性について考察を行う．

2. 既存のブロックプログラミング言語の開発環境

上記のブロック作成システムを紹介する．

2.1 Snap!

Snap!は Scratch から着想を得て，これを拡張したブロックプログラミング言語である．Scratch は子供向けに開発されたブロックプログラミング言語でアニメーションやゲームなどを簡単に作成することができる．Snap!は，Scratch には含まれない高度な言語機構を持ち，関数定義による新たなブロックの作成が行える．ブロックを生成する機能では最初から用意されているブロックを組み合わせることで新しいブロックを定義・作成することができる．例えば，四角形を描画するブロックを新しく作成したいときは，ブロックエディターで「100 歩進む」ブロック，「左に 90 度回す」ブロック，「4 回繰り返す」ブロックを組み合わせることで新しく「square」ブロックが作成され，システム内で使用が可能になる．複数の処理を 1 つのブロックにまとめることができるので，それ以降のプログラム量が大幅に少なくなる利点がある．また first class continuation を含め強力な言語機構をサポートするため，自由なブロックの構成が可能である．一方で，これらは Snap!内部で閉じている機能であるため，基本的には Snap!で提供されるインタプリタの上で動作させる必要があることや，ブロックを何らかのプログラムに変換するビジュアルプログラミング言語を開発することができないという制約がある．

2.2 Blockly と Blockly Developer Tools

Blockly はビジュアルプログラミング言語開発用のライブラリであり，ブロックの作成やプログラム生成を支援するライブラリである．Blockly Developer Tools は Blockly ライブラリを使用したブロックプログラミング言語作成支援のシステムであり，ブロックの形を定義するブロックや型や色を定義するブロックを組み合わせることにより，様々なブロックを作成できる．例えば「root」，「value_input」，「statement_input」，「field_static」の 4 つのブロックを組み合わせることで，条件を付与できかつ内部に構造を保持可能な「while」ブロックを作ることができる．

Blockly Developer Tools では変換器を作成する

ことが可能である．そのためこのシステムではさまざまな用途のブロックプログラミング言語を開発することができる利点がある．

しかし，前述したように変換器の作成の困難さや煩雑な操作性，プログラムを記述しなければ作成できない複雑な構造のブロックの存在などが問題として挙げられる．

3. 提案手法

3.1 概要

ブロックプログラミング言語開発の問題点をまとめると以下のようになる．

(1) ライブラリや開発言語の理解が不可欠であること
(2) 変換器の作成には専門的な知識が必要であり困難が伴うこと
(3) 開発システムでは複雑な構造のブロックの作成が困難であること
(4) 開発システムの操作が煩雑であることで開発に多大な時間がかかってしまうこと

そこで本研究ではこれらの問題点を解決するために 3 つの手法を提案する．

- 手法 1　雛形ブロックを用いた支援
  ブロックの雛形を用意し，その雛形ブロックに直接変更を加えることで新規のブロックを作成する．
- 手法 2　コード断片からブロックを提案
  既知のプログラミング言語の構文を入力することでそれに対応したブロックを提案する．
- 手法 3　ドキュメンテーションコメントからブロックを提案
  変換先言語で記述されたソースコードに含まれるドキュメンテーションコメントに基づきブロックを提案する．

3.2 雛形ブロックを用いた支援

雛形となるブロックを形や変換器の内容別に複数用意しておき，それを編集することで新しいブロックを作成できる．雛形ブロックは形や変換器の内容を基準に選定する．編集を行えるのはブロックの見出しに相当する部分のみであり，Input や Field の追加・削除は行えない．また見出し以外の部分を編集した場合その処理は完成ブロックに反映されない．

例えば，変数定義ブロックから定数定義ブロックを作成することを考える．変数定義ブロックは雛形ブロックとして用意されているものとする（図 1 左上左側）．このブロックからはその右側のブロックが完成する．この雛形ブロックの左側の入力フィールドの「var」を「const」に変更する．すると変更が反映され新しく定数定義ブロックが作成できる．雛形ブロックを編集したことでブロック定義の対応した部分が自動で変更される．変換器も対応した部分を手動で変更することで新しいブロックが完成する．またこの例においてブロックの見出しとなる部分は「var」「const」の部分であり，「id」と入力された Field は見出しではない．したがって，もし「id」を他の値に変更したとしても完成ブロックには影響しない．

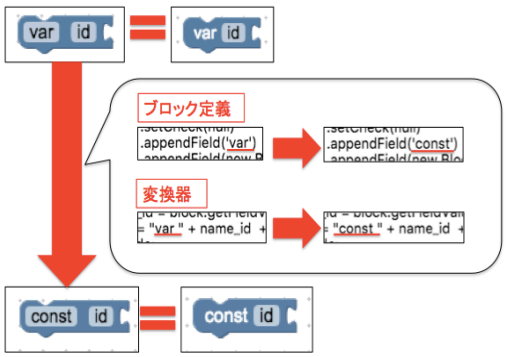

図 1　雛形ブロックの処理の流れ

3.3 コード断片からブロックを提案

既知の言語の構文を入力することでブロック定義と変換器を自動生成し，ブロック作成支援を行う． 構文からブロックが提案されることによって変換器の作成を省略でき，ブロックの実装を意識する必要がなくなるためブロックの作成が簡易化される．

本提案手法ではコード断片とブロックの間に中間表現を挟んでいる．中間表現は入力されたコード断片の抽象構文木を基に作成している．この中間表現には抽象構文木とほぼ同等の内容が含まれておりその情報からブロック定義と変換器を自動生成することでブロックを提案している．抽象構文木には不要な情報を省いた構文の情報が含まれている．

例えば「while 文」を入力した場合「 ( )」内に入力される「test」と「{ }」内に入力される「body」のような抽象的な情報のみが木として表現される．この「test」や「body」などについてそれぞれブロック定義と変換器を作成し，それらを組み合わせることでブロックの自動生成を実現している．(図 2)

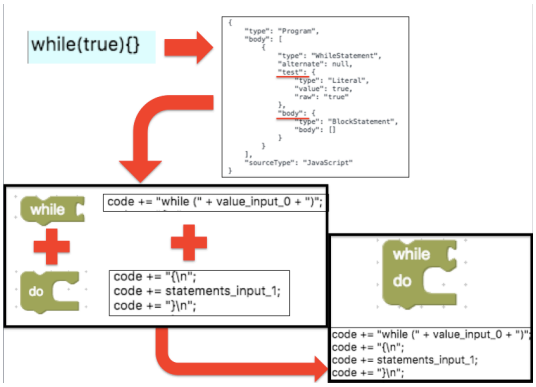

図 2　コード断片からブロックへの変換

3.4　ドキュメンテーションコメントからブロックを提案

ソースコード内のドキュメンテーションコメントに基づいて，それに対応したブロックを提案する．ブロックプログラミング言語には既知の言語のライブラリやモジュールが DSL として実装されたものが多い．一般的にライブラリ関数は大量であるため，これを実装しようとすると多大な時間がかかってしまう．ドキュメンテーションコメントからブロックを自動生成し提案することで，大量の手間を省くことができる．またドキュメンテーションコメントに含まれる関数の説明などもブロックの情報として組み込むことができるため，より効率的にブロックを作成できる．

本提案手法ではドックレットを自作することでドキュメンテーションコメントの解析を行う．ドックレットとは JavaDoc[7] のようなドキュメントジェネレータの出形式を決定するものである．JavaDoc はデフォルトでは HTML 形式で API ドキュメントを出力する標準ドックレットを採用している．本手法ではコメント解析結果を JSON 形式で出力しブロックに変換する．

4.　システムの実装

本システムは Blockly ライブラリを使用してシステムを構築する．使用言語は JavaScript とする．

4.1　システム構成

本システムは，BlockPlant と BlockConverter の 2 つの編集ページから構成される．BlockPlant では Blockly Developer Tools の既存機能と雛形ブロックの機能を利用することができる．BlockConverter ではコード断片からブロックを作成する機能を利用することができる．BlockPlant と BlockConverter の切り替えはそれぞれのページの右上部のボタンで行う．またそれぞれのページのプログラム部にはプログラムの保存を行えるボタンが備わっており，既存システムで手間となっていた保存の処理を簡略化している．

4.2　BlockPlant

ワークスペース部（図 3①），プレビュー部（図 3②），プログラム部（図 3③）の 3 部で構成される．ワークスペース部は雛形ブロックなどを格納しておく「ツールボックス」とブロックを配置する「ワークスペース」で構成されている．プレビュー部はブロックの完成形を表示する「ブロックプレビュー」とブロックから変換されたコード断片を表示する「プログラムプレビュー」で構成されている．プログラム部は上段に位置するブロックの形や色などの情報が記述されたプログラムを表示する「ブロック定義部」と下段に位置するブロックをコード断片に変換する際の処理を表示する「変

換器部」で構成されている．

雛形ブロックはツールボックスに格納されており，ワークスペースに移動することで編集が可能になる．デフォルトで配置されているのは「factory_base」ブロックでこれは Blockly Developer Tools の機能を用いてブロックを作成するときの基礎となるブロックである．このブロックが配置されているときは雛形ブロックの機能を利用することができないためワークスペース上から削除する必要がある．また雛形ブロックが複数個配置されている場合には最初に置かれたブロックのみが新しいブロックとして変換される．

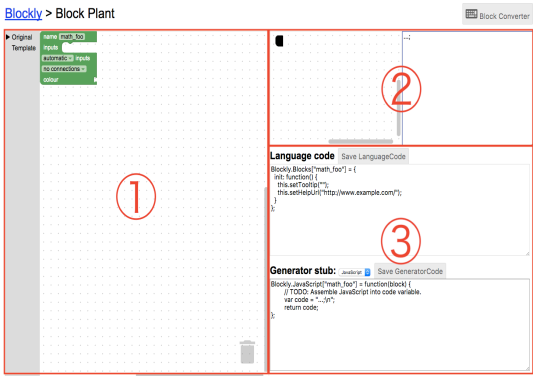

図 3　BlockPlant 全体構成

4.3　BlockConverter

コード入力部（図 4①），ブロックプレビュー（図 4②），プログラム部（図 4③）の 3 部で構成される．コード入力部には既知の言語のコード断片を入力することができる．ブロックプレビューにはブロックの完成形が表示される．プログラム部は BlockPlant のプログラム部と同じ役割を果たす．

コード断片からブロックへの変換はショートカットキーを用いて実行する．またページ中央上部に配置されたドロップボックスから言語を選択することで入力するコード断片の文法を決めることができる．現在は JavaScript と Python のみ対応している．さらに入力された構文が文を複数保持していた場合にはそれらすべてがブロック化される．さらに，ブロックの見た目は柔軟に変更可能である．コード入力部とブロックプレビュー部の中間にある「tree ボタン」を押下することでコード断片と対応する抽象構文木のグラフィックス表現を表示できる．ここからノードを選択することで input 部や field 部の有無を選択することができる．

本システムでは構文エラーが発生するような入力 （コード断片）の場合には抽象構文木への変換が行われず，ブロックは作成されない．構文エラーが発生した場合にはコード入力部の背景が赤くなり，エラーがない場合には背景が青く表示される．

また作成したブロックの動作確認を行うこともできる．指定されたショートカットキーを押すことで確認モードに突入し，ブロックが変換器に即した処理を適切に行うかを確認できる．確認モード中にブロックプレビューでブロックを操作するとコード入力部に処理結果が出力される．

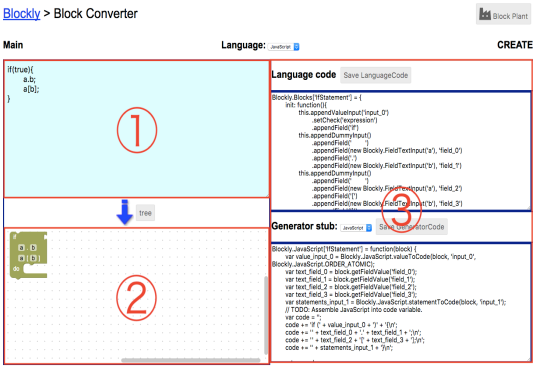

図 4　BlockConverter 全体構成

5.　考察

既存システムと提案システムで 2 つのブロックプログラミング言語を例題として本システムによって実装を行い，ブロック作成に必要な工程数について比較評価を行なった．1 つ目の例題は JavaScript の構文をブロックプログラミング言語化したものでブロック合計数は 59 個である．そのうち雛形の手法のみで作成できたのは 8 個，コード断片の手法のみで作成できたのは 27 個，2 つの手法を組み合わせて作成できたのは 19 個，作成できなかったのは 5 個となった．2 つ目の例題は Scratch のブ

ロックの中でタートル命令を表現したブロックのみを対象とし，ブロック合計数は 22 個である．そのうち雛形の手法のみで作成できたのは 0 個，コード断片の手法のみで作成できたのは 18 個，2 つの手法を組み合わせて作成できたのは 3 個，作成できなかったのは 1 個となった．実験では基本的な形のブロックや関数呼び出しのような単位の小さなブロックであれば提案システムのほうが短時間で作成でき，手法を組み合わせることでより簡単にブロックを作成できることも発見できた．だが問題点も見つかった．雛形ブロックを用いる手法では複雑な構造のブロックを雛形として用意し支援を試みている．しかし雛形ブロックとは異なる構造を持ったブロックは結局作成が困難になってしまった．コード断片からブロックを提案する手法では，現実装では JavaScript の構文解析器を用いているためそれ以外の言語の構文や未知の入力に対して，ブロックを提案できない．これらの問題点は今後の課題として解決していく．

6. おわりに

本稿ではブロックプログラミング言語処理系開発の支援のための手法を提案しその実装を行った．ブロックプログラミング言語開発の問題点として専門知識が必要なことと時間がかかりすぎることが挙げられる．本提案では雛形ブロックを用いる手法，コード断片からブロックを提案する手法を示し，専門知識がなくても言語の開発が行える環境を提供する．ドキュメンテーションコメントからブロックを提案する手法では，効率よくブロックを自動生成することで開発にかかる時間を削減できると考えている．

今後は雛形ブロックを用いる手法で発見された問題点の改善，デザイン面をもっと柔軟に変更できるような機能の考察を行う．雛形ブロックを用いる手法の問題点とは雛形ブロックとは異なる構造を持ったブロックは結局作成が困難となってしまうことだ．これは雛形として用意するブロックをさらに精査することで様々なパターンのブロックに対応できるようになると考えられる．またブロックのデザイン面については，Snap!のように複数のブロックを 1 つにまとめられるマクロ化のような機能など，様々な形のブロックが簡単に作成できるような仕組みを考えていく必要があると感じている．

参考文献

[1] Google,Blockly. https://developers.google.com/blockly/ (参照 2017-12-04).

[2] MIT,Scratch-Blocks. https://github.com/LLK/scratch-blocks (参照 2017-12-04).

[3] Google,Blockly Developer Tools. https://blockly-demo.appspot.com/static/demos/blockfactory/index.html (参照 2017-12-04)

[4] MIT,Scratch. https://scratch.mit.edu (参照 2017-12-04)

[5] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages.

[6] Snap!. http://snap.berkeley.edu (参照 2017-12-04)

[7] Oracle, JavaDoc. https://docs.oracle.com/javase/jp/8/docs/technotes/tools/windows/javadoc.html (参照 2017-12-04)