

Blockly を用いた多言語対応の プログラミング学習支援環境の開発

香川大学工学部電子・情報工学科	
卒 業 論 文	
卒 業 年 度	平成 29 年度 (2017 年度)
指 導 教 員	香 川 考 司

香川大学工学部 電子・情報工学科

佐野 裕也

平成 29 年 2 月 15 日

Development of programming learning support environment for multilingual support using Blockly

Abstract When a programming beginner learns a new programming language, he must learn the fundamental concept of programming and the grammar of language at the same time. This is a heavy burden for the learner. To solve this, Ozaki's research adapted Blockly, a Web-based graphical programming editor, to the C language. In addition, Yamagata's research implemented a function that presents exercises to Blockly. In this research, Blockly is compatible with multiple languages on the premise that the operation of the system is easy, increasing the dynamic deformation function of blocks other than the Mutator function, so that beginners of programming can expand the range of learning of program beginners. As a result of evaluating the students in the laboratory, it was possible to broaden the range of learning of programming beginners by multilingualizing the system, but it was pointed out that the operation is complicated in some systems. In the future, we aim to improve the system based on the feedback obtained in the laboratory, and a system that can confirm the effect in university lessons.

あらまし プログラミング初心者が新たなプログラミング言語を学習するとき、プログラミングの基礎概念と言語の文法を同時に学習しなければならない。これは、学習者にとって大きな負担である。これを解決するために尾崎の研究では、Web ベースグラフィカルプログラミングエディタである Blockly を C 言語に対応させた。また山形の研究では、Blockly に練習問題を提示する機能を実装した。本研究では、システムの操作が容易であることを前提に Blockly を多言語対応し、Mutator 機能以外のブロックの動的変形機能を増やして、プログラミング初心者の学習の幅を広げられるようにした。研究室の学生を対象に評価をした結果、システムの多言語化を行うことでプログラミング初心者の学習の幅を広げることができたが、一部のシステムにおいて操作が複雑であることが指摘された。今後は、研究室で得られたフィードバックをもとにシステムを改善し、大学の授業で効果を確認できるようなシステムを目指す。

キーワード プログラミング学習, C, JavaScript, Haskell, Flex, Web ベース

目次

1	はじめに	1
1.1	はじめに	1
1.2	Web ベースの利点	1
1.3	先行研究	2
1.3.1	尾崎の研究	2
1.3.2	山形の研究	2
1.4	システムに求められる要件	2
2	Blockly	5
2.1	全体の主なファイル構成	5
2.2	システム構成	6
2.2.1	ワークスペース部	6
2.2.2	ブロックメニュー部	7
2.2.3	ソースコード部	8
2.2.4	XML コード部	9
2.3	オプション機能	9
2.4	Mutator 機能	10
2.5	ブロックの定義	10
3	実装	14
3.1	多言語化	14
3.1.1	対応する言語の種類	14
3.1.2	システムの WEB ページ	15
3.2	動的変形機能	15
3.3	新たに作成したブロック	16
3.3.1	Blockly for C	16
3.3.2	Blockly for Flex	17
3.3.3	Blockly for Haskell	19
3.4	サンプルボタン機能	22
3.4.1	問題点	23
3.4.2	改善点	23
3.5	新たに実装したオプション機能	24

4	評価	25
4.1	項目	25
4.2	結果	25
4.2.1	Blocly for C を利用しての意見	26
4.2.2	Blocly for Haskell を利用しての意見	27
5	おわりに	28
5.1	まとめ	28
5.2	今後の課題	28
	謝辞	31
	参考文献	32
	付録 A プログラムの全ソース	33
A.1	ファイル名	33

第 1 章

はじめに

1.1 はじめに

プログラミング学習者は、以下の 2 つを同時に学ばなければならない。

- プログラミングの基礎概念
- 各プログラミング言語の文法

これらを学ぶことは、学習者にとって大きな負担である。この負担を軽減するために、文法を意識せずにプログラミングができる学習環境が必要である。これを解決するために、Web ベースのビジュアルプログラミング言語である Blockly[1] を用いる。

Blockly とは、Google で開発されているグラフィカルなプログラミングエディタである。図 1 のようなブロックを繋ぎ合わせることでプログラミングを行う。このため構文エラーに悩まされず、直感的にプログラミングをすることができる。JavaScript で記述されており、ドキュメントも豊富に用意されているためカスタマイズが容易である。また、Blockly は Web ベースのアプリケーションであるため、導入の作業が不要である。さらに、Blockly で作成したプログラムは、JavaScript、Dart、Python、Lua、PHP の 5 種類のコードに変換して出力することができる。

本研究では Blockly の多言語化を目標とする。多言語化によって、より Blockly で学習支援できる範囲が広がり、学習者が自分に適した言語を選ぶことができる。しかし、必要となるブロックの種類や形状は増加するので、ブロックの形を動的に変形させることも考えなければならない。

1.2 Web ベースの利点

本研究で開発したシステムは、Web ベースシステムである。Web ページすることにより、学習者に次のような利点があると考えられる。

- システムのインストールが不要
- Web ブラウザがあれば実現可能
- 常に最新版のシステムが利用可能

また、教員側にも次のような利点がある。

- システムのカスタマイズが容易
- システムの不具合をすぐに修正可能
- 導入に関する指導が不要

1.3 先行研究

1.3.1 尾崎の研究

尾崎の研究 [2] は、Blockly を C 言語、Flex 言語に対応させたもので、システムの対象者がプログラミング入門者である。このシステムによって文法を意識せずにプログラミングを学ぶことができる。しかし、大学の講義で学習する言語に対応しきれていない。システム自体が便利であっても、大学の講義で学習する言語に対応していないと、大学生はこのシステムを利用することができない。また、ブロックの形を動的に変形することができないため、柔軟性のあるプログラミング言語をブロックの形状によって制約されてしまうことになる。尾崎の研究で実装したシステムのイメージを図 1.1 に示す。

1.3.2 山形の研究

山形の研究 [3] は、尾崎の研究で Blockly を C 言語に対応させたものに、問題提示機能の実装を行ったものである。このシステムは、ブロックが構文と一対一で対応しており、Blockly を使った予習の段階から演習の段階まで利用できるようになっている。また、練習問題を選択問題や穴埋め問題だけでなく他の種類の問題も行うことができる。山形の研究で実装したシステムのイメージを図 1.2 に示す。

1.4 システムに求められる要件

先行研究の問題点より、本システムに求められる要件は以下の 3 つである。

- システムの操作が容易

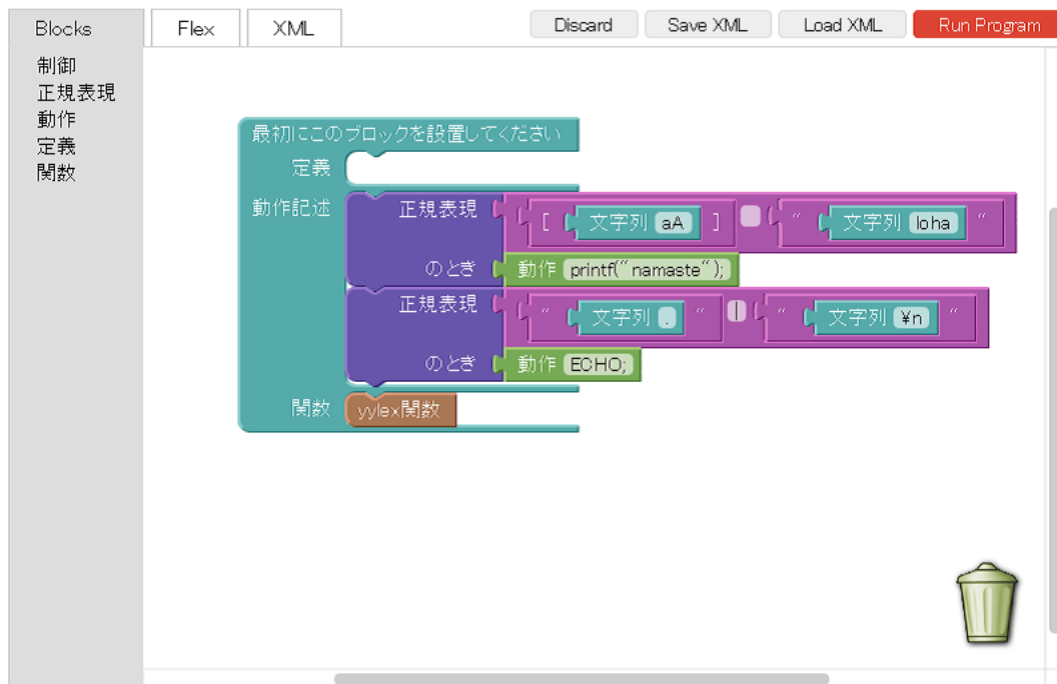


図 1.1: 尾崎の研究で実装したシステム

- 大学の講義で学習する言語に対応させるための多言語化に対応
- プログラミング言語の柔軟性の保持

大学の講義で学習するプログラミング言語には、C、Java、JavaScript、PHP、Haskell、Flex などがあるが、いずれの言語もそれぞれの特性ともっており、その特性を踏まえて何も無い状態からソースコードを記述するまで理解するにはかなりの時間と負担がかかる。前の項でも述べた通り、Blockly はプログラミング入門者に適したシステムなのだが、大学の講義で学習するプログラミング言語に一部対応していない。

これらの問題点を踏まえて、システムの多言語化が必要である。例えば、関数型言語の Haskell はパターンマッチングを行うが、既存の Blockly for JavaScript にある関数定義ブロックではこのパターンマッチングに対応しておらず、新たに定義する必要がある。このように、プログラミング言語の文法の違いで新たにブロックを定義する必要があるので、システムの多言語を行うことで必要となるブロックの種類が増加する。また、柔軟性のあるプログラミング言語をブロックの形状によって制約されてしまう。これらに対応するために、ブロックの動的変形機能を拡張する。その際に、プログラミング学習者の負担を軽減させることが目的なので、システムの操作は容易である必要がある。

本論文では、第 2 章で Blockly について、第 3 章で実装したシステム、第 4 章でシステムの評価、第 5 章でまとめ、今後の課題について述べる。

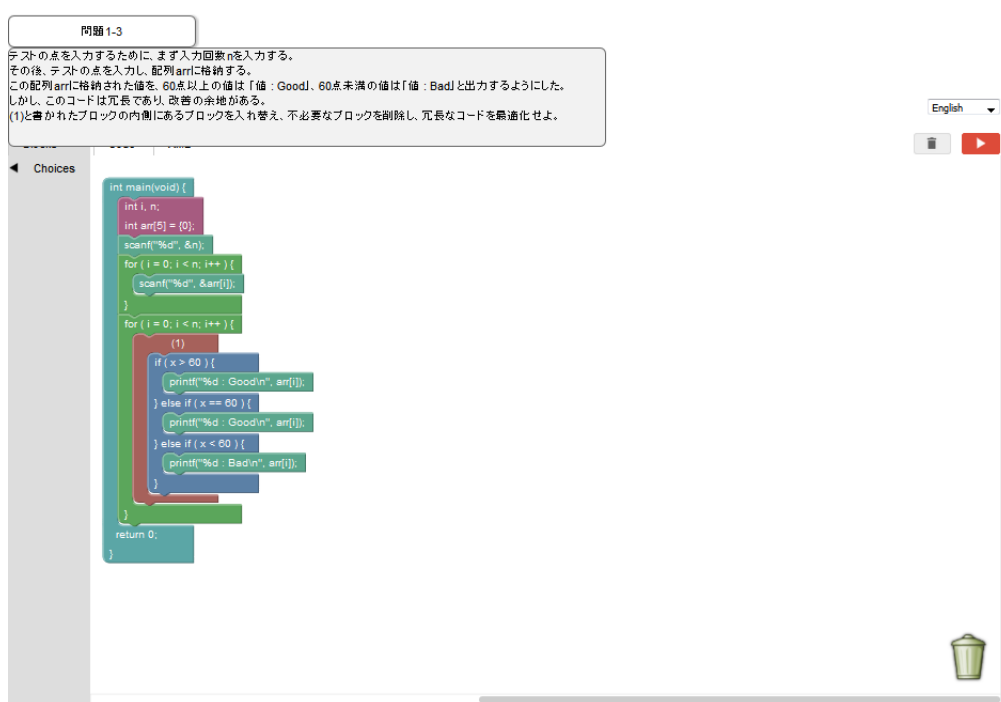


図 1.2: 山形の研究で実装したシステム

第 2 章

Blockly

2.1 全体の主なファイル構成

Blockly は、以下のファイルで構成されている。システムに関するファイルのソースコードは JavaScript 言語で記述されており、WEB システムの構成は html 言語で記述されている。

- core フォルダ

システム自体の定義を行うプログラムで構成されている。また、blockly フォルダには、Blockly の本体である blockly_compressed.js ファイルが存在する。

- demos フォルダ

各システムの WEB ページの構成を表すファイルが存在する。そのほとんどが html で記述されている。

- blocks フォルダ

このフォルダの直下にブロックの種類ごとのファイルがあり、ブロックの形状を定義している。これらのプログラムはビルドされ、Library フォルダの blocks_compressed.js ファイルに圧縮されている。そのため、これらのファイルを書き換えてもシステムが変更されることはない。

- generators フォルダ

このフォルダの直下に各プログラミング言語のフォルダがある。それらのフォルダの中にブロックの種類ごとのファイルがあり、各ブロックを配置したときに出力されるソースコードの内容が書かれている。

- i18n フォルダ

拡張子が .soy のテンプレートがあり、コンパイルすることで JavaScript ファイルを生成する。

- scripts フォルダ

拡張子が、.sh のシェルスクリプトがあり、全体のファイル操作、プログラムの実行内容が記述されている。

2.2 システム構成

Blockly は、ワークスペース部、ブロックメニュー部、ソースコード部、XML コード部の 4 つのコンポーネントによって構成される。図 2.1 に Blockly のイメージを示す。デフォルトでは、ワークスペース部が表示されており、ソースコードタブを押すとソースコード部のコンポーネントに切り替わり、XML コードタブを押すと XML コード部のコンポーネントに切り替わる。ブロックメニュー部は常に左部に表示されている。ここでは、それぞれのコンポーネントについて説明する。

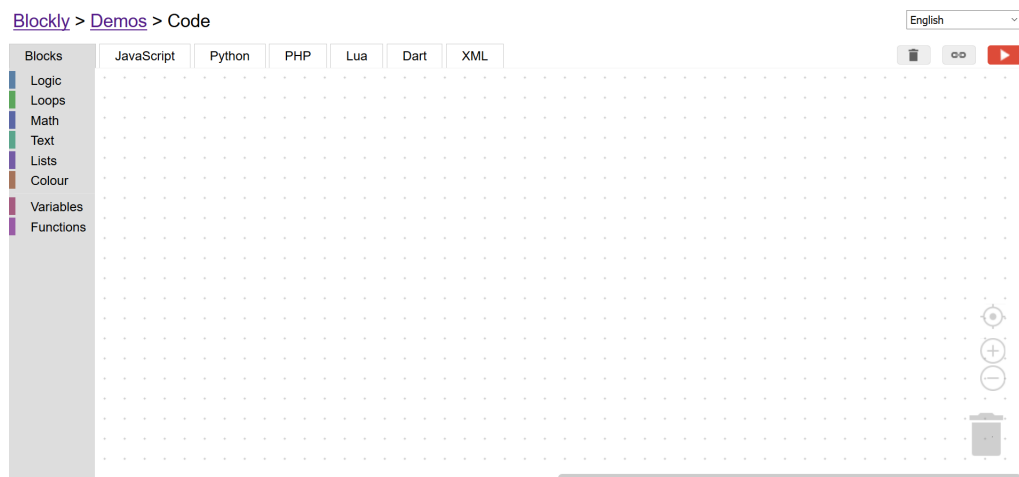


図 2.1: Blockly

2.2.1 ワークスペース部

ワークスペース部は、ブロックを使ってプログラミングを行うスペースである。図 2.2 にワークスペース部のイメージを示す。ユーザは、ブロック部に存在するブロックをワークスペース部にドラッグ&ドロップで自由につなぎ合わせることでプログラミングを行う。ワークスペース部の右下にはゴミ箱アイコンが存在する。ワークスペース部に設置したブロックをこのアイコンに

ドラッグすると、ブロックを削除することができる。その上に拡大縮小アイコン、さらにその上に中央に寄せるアイコンが表示されている。

また、ワークスペース部に設置したブロックを右クリックすることで、以下のような動作が行える。

- ブロックの複製
- コメントの追加
- ブロックの接続表現の切り替え
- ブロックの表現の簡略化
- ブロックの透明化
- ブロックの削除
- ブロックの動作を説明する Web ページを表示



図 2.2: ワークスペース部

2.2.2 ブロックメニュー部

ブロックメニュー部は、定義されたブロックが存在するスペースである。図 2.3 にブロックメニュー部のイメージを示す。ブロックは、論理、数、リストなどのカテゴリーにそれぞれ格納され、カテゴリーをクリックすると、そのカテゴリーに格納されたブロックが表示される。図 2.4 にカテゴリーをクリックしたときのイメージを示す。ユーザは、このブロックメニューから新たに必要なブロックを選択して使用する。



図 2.3: ブロックメニュー部

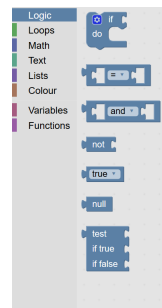


図 2.4: カテゴリーをクリックしたとき

2.2.3 ソースコード部

ソースコード部は、作成したプログラムのソースコードを表示するスペースである。図 2.5 にソースコード部のイメージを示す。タブ名は各プログラミング言語の名称になる。ワークスペース部でブロックを組み合わせて作成したプログラムが、リアルタイムにソースコードに変換され、このソースコード部でいつでも確認することができる。この機能によって、Blockly でプログラミングを学んだ学習者がソースコードを記述できるように支援する働きを持つ。システムの拡張の際に新たなブロックを実装した場合は、新たなブロックの定義と共に新たにソースコードも定義しなければならない。

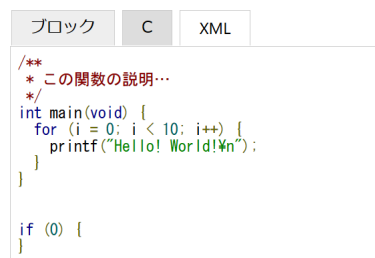


図 2.5: ソースコード部

2.2.4 XML コード部

XML コード部は、作成したプログラムの XML コードを表示するスペースである。図 2.6 に XML コード部のイメージを示す。ワークスペースで組み合わせたブロックの構造が XML 形式で出力され、その出力された XML コードをセーブ、ロードすることができる。この機能によって、組み合わせたブロックを保存したり、他の学習者や教員が組み合わせたブロックを自分のワークスペース上に再現することができる。以下は、Blockly for C におけるもしも実行ブロックの XML コードである。block という要素タグに、type、id、x、y の 4 つの属性がある。type の属性値には、ブロックの名前が入り、id の属性値は 20 文字のランダムな文字列が入り、id は重複しないようになっている。x、y の属性値はワークスペース上の座標が入る。

```
<block type="controls_if" id="R1lK7egbl/90CZu2p_wZ" x="-2" y="27"></block>
```

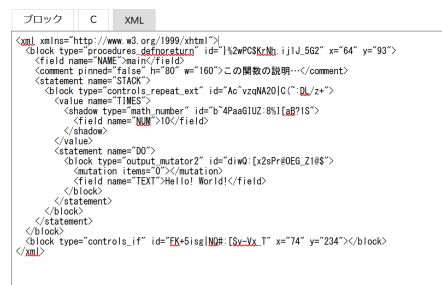


図 2.6: XML コード部

2.3 オプション機能

Blockly には、オプション機能がある。以下の図 2.7 にオプション機能のイメージを示す。これは、システム WEB ページの右下にあるものである。図には 4 つのアイコンがあり、上から、中央に寄せる、拡大、縮小、削除である。

- 中央に寄せるアイコン

このアイコンは、ワークスペース部で組み立てたプログラムを中央に寄せることができる機能である。

- 拡大・縮小アイコン

拡大・縮小アイコンは、組み立てたプログラム全体を拡大・縮小することができる機能である。大規模なプログラミングを組み立てるときにはなくてはならない機能である。

- 削除アイコン

削除アイコンは、ワークスペース部にあるブロックを削除できる機能である。操作方法として、削除したいブロックをドラッグして削除アイコン上でドロップすることで削除できる。また、同様の機能として、削除したいブロック上で右クリックした際のコンテキストメニューにも削除がある。



図 2.7: オプション機能

2.4 Mutator 機能

Blockly の機能に Mutator がある。ここでは、その Mutator という機能について説明する。

Mutator は、ブロックの動的変形を行う機能である。この機能は、Blockly が用意している機能の中で唯一の動的変形である。図 2.8 に Mutator のイメージを示す。Mutator の機能が使えるブロックには、左上に歯車のマークがある、その歯車のマークを押すと、その近くに吹き出しの形をした小窓が現れる。小窓の左半分はブロックメニュー部、右半分はワークスペース部となっている。小窓の中のコンポーネントは、このブロック限定のものである。小窓のブロックメニュー部から拡張したいブロックを取り出し、小窓のワークスペース部の既存のブロックに取り付ける。すると、吹き出し元のブロックが変形される。図 2.9 が、そのときのイメージである。Mutator 機能によって、ブロックの形をカスタマイズでき、ブロックメニュー部で用意されるブロックの種類を大幅に減らすことができる。実装で動的変形機能の拡張を行うときは、この機能を参考にして新たな動的変形機能を実装する。

2.5 ブロックの定義

ここで、ブロックの定義方法について述べる。まず初めに、一般的なブロック定義のソースコードを以下に示す。ブロックの形状を定義したいときは、init 関数のなかに関数を記述していく。



図 2.8: 動的変形前の Mutator ブロック

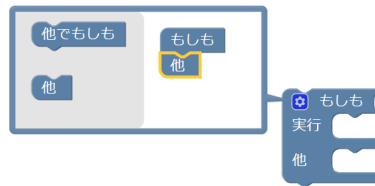


図 2.9: 動的変形後の Mutator ブロック

```
Blockly.Blocks['sample_name'] = {
  init: function() {
    this.setColour(100);
    this.appendDummyInput()
      .appendField("サンプル");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setOutput(true, 'String');
    this.appendValueInput('A');
    this.setInputsInline(true);
    this.appendStatementInput('D00');
    this.setTooltip("このブロックは、サンプルです。");
  }
};
```

Blockly には新たなブロックを定義するためにいくつかの関数が用意されている。以下にそれぞれの関数の機能と使用方法を示す。

- appendField

appendField は、ブロック上に文字列や入力フォーム、ドロップダウンメニューを定義する関数である。この関数の前に appendDummyInput 関数を呼び出す必要がある。以下は、appendField の使用例である。

```
this.appendDummyInput()  
    .appendField("サンプル");
```

- setColor

setColor は、ブロックの色を定義する関数である。0 から 360 までの数値を指定することで、ブロックの色を定義することができる。以下は、setColor の使用例である。

```
this.setColour(100);
```

- setPreviousStatement

setPreviousStatement は、ブロック上接続部の有無を定義する関数である。第 1 引数が true なら接続部が有り、false なら無しである。以下は、setPreviousStatement の使用例である。

```
this.setPreviousStatement(true);
```

- setNextStatement

setNextStatement は、ブロック下接続部の有無を定義する関数である。この関数も同様に、第 1 引数が true なら接続部が有り、false なら無しである。以下は、setNextStatement の使用例である。

```
this.setNextStatement(true);
```

- setOutput

setOutput は、ブロック左接続部の有無を定義する関数である。第 1 引数が true なら接続部が有り、false なら無しである。第 2 引数は、接続先のブロック型を指定する。接続先のブロックの型が指定したものと同じならブロックを接続することができる。型の指定がない場合は、第 2 引数ごと無くすか、null を指定する。以下は、setOutput の使用例である。

```
this.setOutput(true, 'String');
```

- appendValueInput

appendValueInput は、ブロック右接続部の有無を定義する関数である。この接続部に関する関数は、1 つのブロックに対し複数指定できる。第 1 引数には、接続部の名称を指定できる。接続部の名称を指定することで、ブロックの他の接続部と区別させるためである。一般には、変数ブロックや数ブロックを接続する。setChecked 関数をそのあとにつけることで接続先のブロックの型を指定できる。以下は、appendValueInput の使用例である。

```
this.appendValueInput('A')  
    .setChecked('int');
```


- `setInputsInline`

`setInputsInline` は、`appendValueInput` 関数で定義した右接続部をブロックの内側 (以後ソケットと呼ぶ) にするかを定義する関数である。第 1 引数が `true` ならソケットとして定義し、`false` なら右接続部そのままである。以下は、`setInputsInline` の使用例である。

```
this.setInputsInline(true);
```

- `appendStatementInput`

`appendStatementInput` は、ブロックの内側で上接続部を定義したブロックと接続できるように定義する関数である。第 1 引数には、接続部の名称を指定できる。接続部の名称を指定することで、ブロックの他の接続部と区別させるためである。一般には、命令ブロックと接続する。以下は、`appendStatementInput` の使用例である。

```
this.appendStatementInput('DO');
```

- `setTooltip`

`setTooltip` は、ブロック上にカーソルを合わせたときに表示される内容を定義する関数である。第 1 引数は、表示する内容を文字列で定義する。以下は、`setTooltip` の使用例である。

```
this.setTooltip("このブロックは、サンプルです。");
```

第 3 章

実装

本研究では、Blockly で多言語化を行うために、以下のような実装を行った。

3.1 多言語化

3.1.1 対応する言語の種類

まず、Blockly の多言語化を行うために、対応するプログラミング言語の種類を決めなければならない。大学の講義で学ぶプログラミング言語から考えて本システムでは、4 種類のプログラミング言語に対応できるようにした。それが以下の 4 種類のシステムである。

- Blockly for C

先行研究で尾崎と山形が開発したシステムを引き継いで実装を行った。

- Blockly for JavaScript

Blockly の既存のシステムに用意された言語である。

- Blockly for Haskell

Haskell[4] 言語は、本研究であらたに導入される言語である。Blockly の既存に対応している言語と先行研究で実装されている言語はいずれも命令型プログラミング言語であるが、Haskell は宣言型プログラミング言語である。Blockly に宣言型プログラミング言語を導入してはどうかということで、Blockly にあらたに導入される言語として決定した。

- Blockly for Flex

先行研究で尾崎が開発したシステムを引き継いで実装を行った。特に、正規表現ブロックの一部を改良した。

3.1.2 システムの WEB ページ

Blockly の既存のシステムでは、1 つの WEB ページで Blockly の対応する言語をサポートしている。1 つの WEB ページに対応言語名が書かれた複数のソースコードタブが存在し、ワークスペースでブロックを組み立てた後に、ソースコードを表示したい言語のタブを押してソースコードを確認するようになっている。しかし、どのソースコードタブを押してもブロックカテゴリーはすべて同じなので、Blockly に対応するすべての言語に共通できるブロックしか作成することができない。

本システムでは、多言語化を行うために 1 種類の言語に、1 つの WEB ページを用意した。また、筆者が開発した WEB ページにスムーズにアクセスできるように、それぞれの言語のシステムのリンク先を貼り付けたデモページの索引ページも作成した。索引ページのイメージが図 3.1 である。

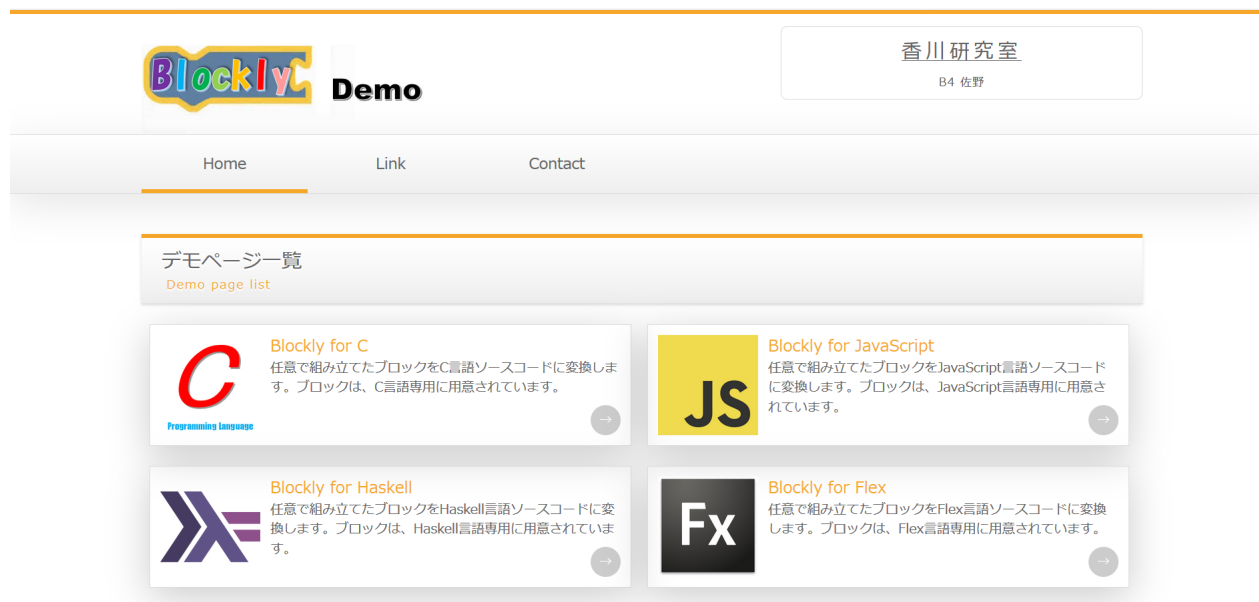


図 3.1: 索引ページ

3.2 動的変形機能

Mutator は、Blockly の既存に用意されている機能だが、本システムで新たに実装した動的変形機能を具体的なブロックを例に挙げて説明する。

本システムで新たに開発した動的変形機能は、printf ブロックに実装されている。そのブロックのイメージは、図 3.2 である。このブロックは、Blockly for C で用意されている。Mutator 機

能では、ブロックの左上に歯車マークがあり動的変形機能が実装されていることが分かるが、このブロックは歯車マークの代わりになるものがないので、動的変形機能が実装されていることが分からない。しかし、このブロックの入力フォームに図 3.3 のような文字列を入力すると、動的変形機能が実装されていることが分かる。これは、入力フォームで”%”の数を検出して、その数だけソケットの数を動的変形機能で増やしている。ソケットとは、変数ブロックや数ブロックを挿入できる穴である。検出のタイミングは、入力フォームの中身に変化があるごとに行われる。この機能は、Mutator 機能と違っていちいち歯車マークをクリックして別途でブロックを組み立てる必要もないので、動的変形の手間を省きかつ操作もシンプルになっているので、プログラミング学習者の負担を軽減させることができている。



図 3.2: 動的変形前の printf ブロック

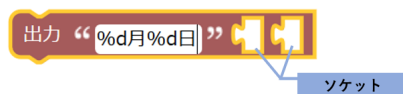


図 3.3: 動的変形後の printf ブロック

3.3 新たに作成したブロック

ここでは、本システムで新たに実装したブロックについて分類ごとに分けて説明する。

3.3.1 Blockly for C

本小節では、Blockly for C で新たに実装したブロックについて説明する。

- 入力・出力ブロック

プログラムにおける入力、出力の処理を表すブロックである。ブロックカテゴリーに格納されている状態では入力フォームのみだが、動的変形機能によって入力フォームに入力され

た”%”の数を自動検出してソケットを増やしていく。以下の図 3.4 に、入力・出力ブロックを示す。

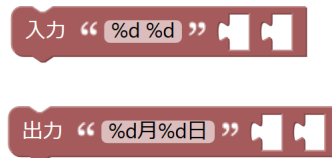


図 3.4: 入力・出力ブロック

- ヘッダブロック群

C 言語のヘッダファイルを表すブロック群である。Blockly for C のブロックカテゴリーのヘッダに格納されている。以下の図 3.5 に、ヘッダブロック群を示す。



図 3.5: ヘッダブロック群

3.3.2 Blockly for Flex

本小節では、Blockly for Flex で表現できる文法を先に説明し、その後に新たに実装したブロックについて説明する。

- Blockly で表現できる正規表現の範囲

- 文字列を表す。以下の表記が記述例である。abc+という 4 文字を表している。

```
" abc+ "
```

- オペレータを文字化する。以下の表記が記述例である。* というオペレータ 1 文字を文字化してる。



*

- 文字の集合を表現する。以下の表記が記述例である。a または b または c の 1 文字を表している。




[abc]

- 文字集合の範囲を指定する。以下の表記が記述例である。0 から 9 までの 1 文字を表している。



[0-9]

- 補集合を指定する。以下の表記が記述例である。数字位階の 1 文字を表している。



[^0-9]

- NL を除く任意の文字を表現する。以下の表記が記述例である。



.

- 二者択一を表現する。以下の表記が記述例である。a または b の 1 文字を表している。



a|b

- 選択の任意を表現する。以下の表記が記述例である。ab または a を表している。



ab?

- 0 個以上の繰り返しを表現する。以下の表記が記述例である。a の 0 個以上の繰り返しを表している。



a*

- 1 個以上の繰り返しを表現する。以下の表記が記述例である。a の 1 個以上の繰り返しを表している。



a+

● 正規表現のブロック群

Blockly for Flex では、先行研究で尾崎が実装したブロックを参考にしながら正規表現のブロックを一新した。ソケットを使うことで、複数の文字範囲を正規表現で表せるようになっているが、2 個が限界である。動的変形機能を使って 3 個以上の文字範囲が実装できるように開発中である。ブロックの表現は、実際の Flex の文法の記述形式としている。これでユーザは、構文エラーに悩まされることなくプログラミングを行うことができる。以下の図 3.6 に、正規表現のブロック群を示す。

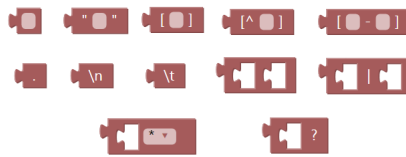


図 3.6: 正規表現ブロック群

3.3.3 Blockly for Haskell

本小節では、Blockly for Haskell で新たに実装したブロックについて説明する。Haskell では、関数におけるパターンマッチングによる場合分けで処理を定義することができ、引数がリストの場合にも対応した。また、内包表記を扱ったリストもブロックで表現可能である。

- もしも実行ブロック

もしも実行ブロックは、他の言語のシステムでも既存に用意されてるブロックだが、Blockly for Haskell では、Haskell の文法の性質上、違うので新たに用意した。また、“if then else” 文の “then” と “else” のあとには、式がくるので、ブロックのつなぎ目の形を変形させた。以下の図 3.7 に、Blockly for Haskell で新に実装したもしも実行ブロックを示す。



図 3.7: もしも実行ブロック

- 制御ブロック群

Blockly for Haskell でブロックカテゴリーの制御で新たにブロックを実装した。main ブロック、do ブロック、let ブロックである。以下の図 3.8 に、Blockly for Haskell の実装した制御ブロック群を示す。

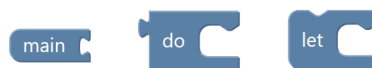


図 3.8: Haskell の制御ブロック群

- リスト内包表記ブロック

Blockly for Haskell のリストカテゴリで新たに内包表記ブロックを実装した。以下の図 3.9 に、内包表記ブロックを示す。図の左が、リスト内包表記ブロックに何も接続していない初期状態で、右がリスト内包表記ブロックの接続例である。リスト内包表記のソースコードは、リストの中に 1 つの式と複数の限定式で記述される。リスト内包表記ブロックでは、内包表記と書かれた右のソケットに式を挿入式し、限定式と書かれたところの右に複数の限定式を表すブロックを挿入する仕様となっている。

[式 | 限定式, ... , 限定式]

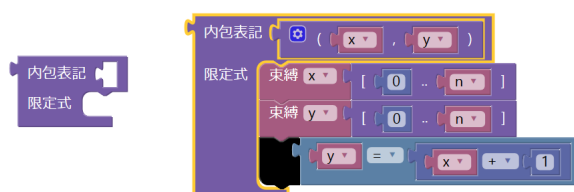


図 3.9: リスト内包表記ブロック

- 関数定義ブロック

Blockly for Haskell で新たに関数定義ブロックを実装した。以下の図 3.10 に、関数定義ブロックの何も接続していない初期状態と関数定義ブロックの小窓の中身を示す。関数定義ブロックでは、Mutator 機能でブロックを動的に変形させることができる。この Mutator 機能で関数の仮引数を自由に設定することができ、小窓のなかの型ブロックを引数コンテナに組み立てることで仮引数を調整できる。また、引数コンテナに接続された一番下の型ブロックは、関数定義ブロックの返却値となり、関数定義ブロックを動的変形に変形しない仕様となっている。

1 つの関数定義ブロックで 1 つのパターンマッチングができるようになっており、仮引数の型が書かれている接続部に仮引数のパターンの変数ブロックやリストブロックを接続する。それらの下の式と書かれた接続部にパターンに対応した式ブロックを接続する。

- 関数呼び出しブロック

関数呼び出しブロックは、関数ブロックを右クリックしてコンテキストメニューで取り出すことができる。このブロックのソケットは、実引数を表し変数ブロックや数ブロックを挿入する。関数呼び出しブロックは、ブロックカテゴリ部にはない唯一のブロックである。関



図 3.10: 関数定義ブロックの初期状態と小窓の中身

数定義ブロックの仮引数の数と関数呼び出しブロックの実引数のソケットの数は動的に対応できるようになっている。以下の図 3.11 に関数呼び出しブロックを示す。



図 3.11: 関数呼び出しブロック

- 関数定義ブロックと関数呼び出しブロックの接続例

以下の図 3.12 に、関数定義ブロックと関数呼び出しブロックの接続例を示す。図の関数定義ブロックでは、小窓のなかの引数コンテナに型ブロックが 3 個接続されている。そして、本体の関数定義ブロックには、仮引数の接続部が 2 か所ある。これは、小窓のなかの引数コンテナに接続されている一番下の型ブロックが、関数の返却値を示しているためである。

今回の例のパターンマッチングでは、第 1 仮引数は任意の変数で、第 2 仮引数は任意のリストである。その際のリストの先頭要素を x 、それ以降のリストを xs を表す。式と書かれた接続部では、もしも実行ブロックを接続し、もしも任意の変数 n とリストの先頭要素 x が等しければリスト xs を返し、それ以外なら先頭要素が x でそれ以降の要素は関数 `deleteOne` を呼び出したリストを返す。リストの中で呼び出した関数 `deleteOne` の第 1 実引数は n 、第 2 実引数は xs である。

このブロック状態でのソースコードは以下のようになっている。

```
deleteOne n (x:xs) = if n == x then xs else (x:deleteOne n xs)
```

- リストのブロック群

Blockly for Haskell のリストカテゴリで、内包表記ブロック以外にも新たに 3 つのリストブロックを実装した。以下の図 3.13 に、新たに実装したリストのブロック群のイメージを

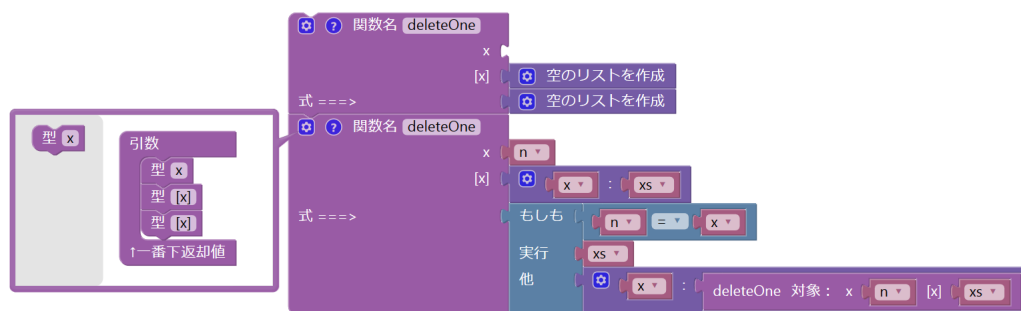


図 3.12: 関数ブロックと関数呼び出しブロックの接続例

示す。図中の上のブロックは、リストを表すブロックである。Mutator 機能で、小窓の中の項目ブロックを結合コンテナに接続することでリストを表すブロックのソケットを増やすことができる。図中の真ん中のブロックは、リストの中の組を表す。こちらも、上のブロックと同じ要領でソケットを増やすことができる。図中の下のブロックは、リストの値の範囲を表す。いずれのブロックもソケットには、数ブロックや変数ブロックが入る。

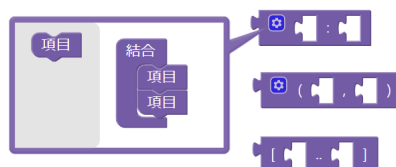


図 3.13: Haskell のリストブロック群

3.4 サンプルボタン機能

Blockly を初めて使う人にとって、ワークスペース部が何も無い状態でブロックを一から組み立てて、実行できるプログラムを完成させることは難しい。そこで、このシステムの初心者が利用しやすくするためにサンプルボタンを実装した。図 3.14 は、サンプルボタンを押したときのワークスペース部のイメージである。WEB ページの上部にサンプルボタンを複数個用意した。このサンプルボタンを押すことによって、システム開発者側が用意したブロックを一瞬でワークスペース

ス部に表示される。システム初心者は、その完成されたブロックのプログラムをアレンジしながらシステムを理解することができる。

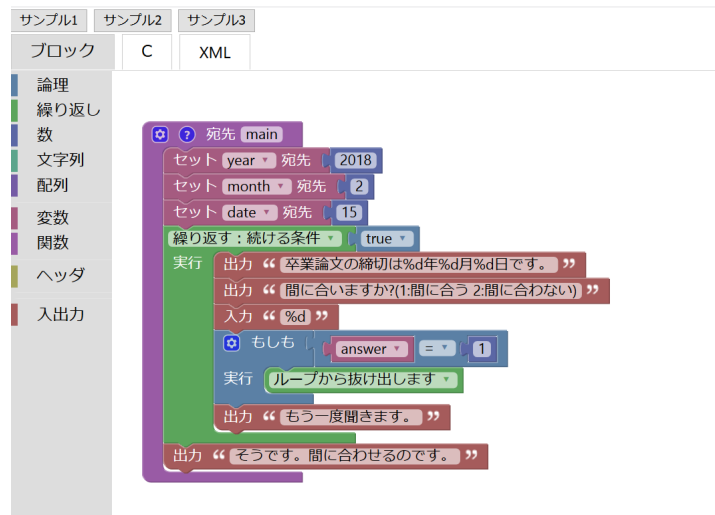


図 3.14: サンプルボタンを押したときのワークスペース部

3.4.1 問題点

サンプルボタンを実装したときに問題が発生した。筆者が実装した動的変形機能付きの入出力ブロックがサンプルボタンを押したときに初期状態のままで出力される。つまり、ソケットがない状態でブロックが出力される。入出力ブロックの動的変形機能は、入力フォームの%の数を検出してソケットの数を増やすのだが、その検出のタイミングが入力フォームに何か変化があるときである。サンプルボタンを押して、入出力ブロックを表示させると入力フォームの変化するタイミングがないため、ソケットも初期状態のままである。

3.4.2 改善点

サンプルボタンの問題点を改善するために、JavaScript の LocalStorage[4] というドキュメントを利用した。LocalStorage とは、データをブラウザ側に保存する仕組みである。同様の有名なドキュメントに Cookie があるが、Cookie には保存の有効期限があるのに対し、LocalStorage には有効期限がない。また、Cookie よりもより多くのデータ量を保存できる。

この LocalStorage を以下のようなソースコードで、入出力ブロックの中の入力フォームの中身を記録することができる。関数 `localStorage.setItem` の第 1 引数に記録させたいブロックの id の属性値、第 2 引数に記録する入力フォームの中身であり、いずれも文字列である。この記録した

中身をサンプルボタンを押した際に、入出力ブロックの入力フォームに渡すことで動的変形を行うことができる。

```
localStorage.setItem(ブロックの id の属性値, 入力フォームの中身);
```

3.5 新たに実装したオプション機能

システム WEB ページ上では右上部に 5 つのボタンが表示されてある。左から、ホーム、ヘルプ、読み込み、保存、リセットである。以下の図 3.15 に新たに実装したオプション機能のイメージを示す。

- ホームボタン

ホームボタンは、筆者が開発したデモページのホームページへ移動できるようになっている。

- ヘルプボタン

ヘルプボタンは、このシステムを初めて使う人向けの機能である。ボタンをクリックすると、順追ってシステムの各機能を説明してくれるようになっている。

- 読み込みボタン

読み込みボタンは、過去に Blockly で保存したプログラムのデータを読み込み、ワークスペース部に出力できる機能である。前回は中断したプログラムの続きを行うことができる。

- 保存ボタン

保存ボタンは、ワークスペース部で組み立てたプログラムを保存することができる。保存形式は、XML である。プログラムを保存しておけば、次回 Blockly を立ち上げたときに、プログラムの続きから行うことができる。

- リセットボタン

リセットボタンは、ワークスペース部で組み立てたプログラムをリセットして何も無い状態から再開できるボタンである。このボタンをクリックする前に、組み立てたプログラムを一旦、保存しておくことを推奨する。



図 3.15: 新たに実装したオプション機能

第 4 章

評価

香川研究室の学部生 3 名と院生 2 名を対象に、本システムの評価を行った。本章では、システム評価について述べる。

4.1 項目

実際にシステムを使用してもらい、その後以下の評価項目に自由に回答する形式で行った。

- 操作方法は直感的に分かったか
- 使ったブロックとそのブロックの評価
- 欲しい機能やブロック
- 不具合報告（無ければ書かなくてよい）

4.2 結果

学部生 3 名には Blockly for C を、院生 2 名には Blockly for Haskell について評価してもらった。Blockly for JavaScript は、既存の Google が実装したシステムでも完成度が高く、本研究ではほとんど手を加えなかったため評価対象から外した。Blockly for Flex は、尾崎の先行研究から正規表現ブロックに動的変形機能を適用させることができなかったため、評価対象から外した。評価の結果、それぞれのシステムの評価項目で次のような意見が挙がった。

4.2.1 Blocly for C を利用しての意見

Blocly for C では、全体的にシステムに関して肯定的な意見が多くみられた。特に、動的変形機能を適用させた入出力ブロックはとても評判が良かった。しかし、その動的変形機能にも一部不具合や例外処理に適應できておらず、まだまだ改善の余地もあることが判明した。以下は、評価項目での具体的な意見である。

- 操作方法は直感的に分かったか
 - 選択中のブロックが枠線で強調されたため、何かしらの動作ができることが分かり、操作する際の足がかりとなり良かった
 - ブロックにマウスカーソルを合わせるとブロックの詳細説明が表示され、分かりやすかった
 - 関数の引数名に重複がある場合、知らせてくれるのが良いと思った
 - 取り外しのできないブロックの色が薄く表示されており、着脱可能かどうか分かりやすかった
 - 直感的にわかったが、配列などの項目の設定などは一回使わないとわかりづらい
- 使ったブロックとそのブロックの評価
 - 出力ブロック: %で出力変数を動的に変更できるのは良いと思った
 - 文字 '%' を出力する際には、C 言語で "%%" と入力する必要があるが、この場合でも出力変数用の % として認識されていたため、 '%' の出力ができないと思った
- 欲しい機能やブロック
 - 文字列や実数の変数ブロック
 - 引数名重複エラーなどのエラーマークは青ではなく、赤に近い色が良いと思う
 - ブラウザにある戻る進むのような機能は欲しいなと感じた
 - ビット演算のブロックはあっても良いかなと感じた
- 不具合報告（無ければ書かなくてよい）
 - 不具合ではないのですが、サンプル 1 の出力ブロックは % による変数の動的検出のないブロックなのですが、動的検出のあるブロックと判別がつかないため、ブロックの見た目で見断できるようにしたほうが良いと思う
 - サンプル 3 の "卒業論文の締切は %d 年 %d 月 %d 日です。" を出力するブロックの変数 (year, month, day) がセットされていない

4.2.2 Blocly for Haskell を利用しての意見

一方で Blocly for Haskell では、評価の際に操作はできたものの分からない点や改善案が多く見られた。また、不具合も多く他の言語のシステムでも共通するような不具合も本システムで発見された。Blocly for Haskell に関しては、まだまだ表現できるプログラムが限定的で、本システムで実装したブロックでも改善点が多いので、まだまだ未熟なシステムであることが痛感される結果となった。以下は、評価項目での具体的な意見である。

- 操作方法は直感的に分かったか
 - Blocly を触ったのは初めてでなかったのもあるが、普通に操作できたと思う
 - どういう風につなげて良いかとかどう関数を作ったらいいかとかがわからなかった
- 使ったブロックとそのブロックの評価
 - 関数のブロック作成が分かりづらく感じた。実際に関数を作成しようとする、どうしてもブロックの数が増えるのは少し面倒に感じた
- 欲しい機能やブロック
 - 関数のブロックの扱いが分かりづらかったので、その説明または使いやすいブロックになるといいと思った
 - let を do を使わずに関数で使いたい
 - Haskell の授業の序盤の補助としてシステムの使用を考えるなら、ある程度出す問題に沿ったまとまったブロックをあらかじめ作成するのがいいのではないかと思った
- 不具合報告（無ければ書かなくてよい）
 - 変数のところにある変更 a は変数 a ではないか？
 - 関数を呼び出す時に変数を n-1 みたいな感じでやったらかっこがついてなかった
 - 自分がやるとサンプルが出なかった。(win8.1 クローム最新)
 - 関数で式が 2 つになったり引数部が上にきたりするときがあり、式 2 つになったときも Haskell のコードをみたら 1 つだけだった
 - スクロールが必要なほどブロックの数が多いタブは一番下のブロックが半分ほど隠れてしまう（デフォルトから 2 段階小さくすれば表示可能になる）

第 5 章

おわりに

5.1 まとめ

大学の講義で学習する言語に対応させるために、Web ベースグラフィカルプログラミングエディタ Blockly の多言語対応を行った。その際に、ブロックの種類が多くなりすぎないように Mutator 以外の動的変形の機能の拡張も行った。この拡張によって、柔軟性のあるプログラミング言語をブロックの形状によって制限されないようになっている。また、拡張した機能もシンプルで、学習者の負担も増やさないようにしている。これらの特徴は、第 1 章で述べたシステムに求められる

- システムの操作が容易
- 大学の講義で学習する言語に対応させるための多言語化
- プログラミング言語の柔軟性の保持

という要件を目指したものである。しかし、実際に授業で使用してもらっていないので、今回の研究室で得られたフィードバックをもとにシステムを改良し、授業で効果を確認する必要がある。

5.2 今後の課題

大学の授業で効果を確認できるようなシステムを目指すために、以下に本システムの課題を述べる。

- Blockly の対応言語をもっと増やす

本システムでの Blockly の対応言語は、C、JavaScript、Haskell、Flex の 4 種類にとどまった。それぞれの対応言語は種類や用途がまったく違うプログラミング言語である。これらの

言語は、いずれも本大学の講義で学ぶ言語であるが、本システムの対応言語が大学の講義で学ぶ言語にすべて対応できているとはいえない。そこで、この目標に達するために次の対応言語として、Java、Ruby、html の実装が必要である。

- Blockly for C の配列に対応させる

本システムの Blockly for C には、配列に対応させることができなかった。Blockly for JavaScript では、既存のシステムでリストが対応しており、Blockly for Haskell のリストは本システムで初めて実装した。これらを参考にしながら、Blockly for C の配列に対応させる必要がある。

- Blockly for Haskell の汎用性を増やす

本システムの Blockly for Haskell は、大学の講義に対応したシステムにするために本学科での講義「プログラミング・パラダイム」の内容に沿って作成した。しかし、講義の内容すべてに対応させることはできず、本システムは限定的なものとなってしまった。そこで、Blockly for Haskell の汎用性を増やすために、代数的データ型、クラス宣言、インスタンス宣言にも対応させる必要がある。

- Blockly for Flex に本システムで開発した動的変形を適用させる

先行研究の Blockly for Flex では、範囲の任意の文字を表す正規表現ブロックが実装されていた。しかし、このブロックは単体での範囲しか表すことができず、ソケット付きのブロックと新たに正規表現ブロックを追加しても 2 つの範囲が限界である。そこで、3 つ以上の文字範囲を表すことができるように、本システムの Blockly for C で実装した入出力ブロックの動的変形機能をこの正規表現ブロックに適用させる必要がある。以下の図 5.1 に、動的変形機能を正規表現ブロックに適用したイメージを示す。

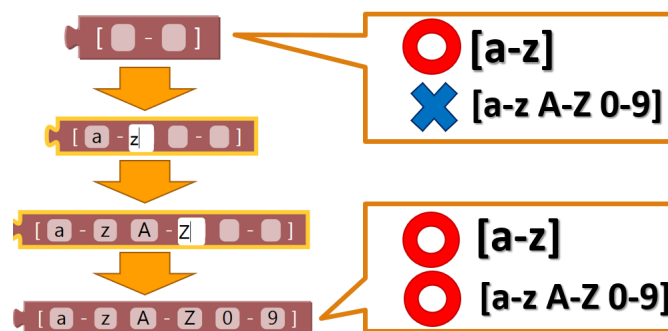


図 5.1: 動的変形機能を正規表現ブロックに適用したイメージ

- 実行タブの実装

本システムの評価の際に、「このプログラムをシステム上で実行するにはどうすればよいか」といった質問があった。本システムでは、プログラムを実行する機能は実装できていない。しかし、本研究室がオープンキャンパスの際に展示する Blockly を使ったお絵かきプログラミングには、プログラムを実行する機能が実装されている。本システムでも、実行できる機能を追加しなければならない。

- ブロック接続部の改善

本システムの Blockly for Flex で接続できないブロックがある。内包表記ブロックの限定式のところで、直接式ブロックを接続しようとしても、接続の形状的に接続することができない。そこで、急遽、接続できないときのために作られたブロックを用意した。以下の図 5.2 の黒色のブロックが急遽、用意したブロックである。このブロックは、プログラム上では何も意味を持たないものであり、このブロックがワークスペース部にある状態で、XML コードタブを押しワークスペース部に戻るとエラーでプログラムが出力できなくなる。この問題を解決するためにも、ブロックの接続部の改善を検討しなければならない。

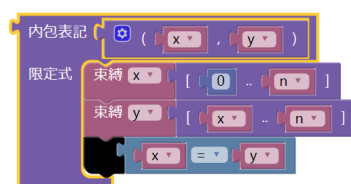


図 5.2: 接続できないときのために作られたブロックのプログラム例

謝辞

本研究においてご指導を承りました香川考司先生に心からの感謝の意を表します。

また、システムの評価に協力して頂いた、大橋健太さん、仁科陽彦さん、朝野有也さん、伊藤拓海さん、太田圭祐さんに深く感謝します。

参考文献

- [1] Google “Blockly,” <https://developers.google.com/blockly/>, 2018 年 2 月 14 日閲覧.
- [2] 尾崎陽一, “Web ベースグラフィカルプログラミングエディタを用いた円滑な移行が可能な C 言語学習支援環境の開発,” 香川大学大学院工学研究科 2014 年度修士論文, 2018 年 2 月 14 日閲覧.
- [3] 山形悠人 “WEB ベースグラフィカルプログラミングエディタを用いた C 言語学習支援への問題提示機能の実装,” 香川大学工学部電子・情報工学科 2016 年度卒業論文, 2018 年 2 月 14 日閲覧.
- [4] Shohei Tai “LocalStorage とは? その使い方を徹底解説! - イケてないコード,” <http://wp.tech-style.info/archives/742>, 2018 年 2 月 14 日閲覧.

付録 A

プログラムの全ソース

A.1 ファイル名

% ソースの実体