Vol.2023-CE-168 No.21
2023/2/12

ビジュアルプログラミングによるプログラミング試験について

田辺 良則[1,a)]　萩谷 昌己[2,b)]

概要：プログラミング能力を測る試験を実施するシステムについて報告する．受験者はブロック部品を組み合わせて，問題の解答となるプログラムを作成・提出し，システムは自動採点を行う．出題者はブロック部品を定義することが可能で，望む粒度の部品を受験者に使用させることができる．自動採点には 2 つの方式があり，あらかじめ用意された入力に対して正しい出力が生成されるかどうかを判定する従来の方式の他，著者らが提案している Presburger 算術に基づいたプログラミング言語への変換を通して，記号実行によって正しさを検証する方式が選択できる．後者では，原理的に，不正解の提出を誤って正解と判定することはおこらない．本発表では，自動採点方式の比較や，コードを書かせる方式との比較などの議論を行う．

キーワード：プログラミング，試験，自動採点，ビジュアルプログラミング，記号実行

On programming examination by visual programming

Yoshinori Tanabe[1,a)]　Masami Hagiya[2,b)]

Abstract: We report on a system that conducts examinations to measure programming ability. Examinees submit programs by combining block parts to answer questions, and the system automatically scores the programs. The question author defines the block parts to have an appropriate granularity for the examination. There are two automatic scoring methods: the traditional method of determining whether pre-prepared inputs produce the correct output and the authors' proposed method of symbolic execution through a programming language based on Presburger arithmetic. In the latter, in principle, incorrect submissions will not be misjudged as correct. We discuss the features of the two automatic scoring methods and compare the block parts method with the method of writing code.

Keywords: programming, examination, automatic scoring, visual programming symbolic execution

1. はじめに

近年の IT 産業の顕著な成長と，それに伴うソフトウェア技術者の相対的不足等を背景として，プログラミングを学習することへの関心が高まっている．初等教育におけるプログラミング教育の導入や，2022 年度からの学習指導要領でのプログラミング必修化により，この傾向はさらに強まっている．

このような背景から，プログラミング初級者に対して，その理解度を評価するための試験の重要度が増している．その際に，採点に必要なコストを考えると，自動採点を行えることが望ましい．大学入学共通テストのような大規模な試験では論を俟たず，授業における小テストであっても，採点に要する時間を減らし，他に労力を振り向けることができれば有用である．

自動採点を前提にした場合，出題形式としてまず考えられるのは選択肢による設問である．プログラミング問題における選択肢による設問形式の問題作成方法は自明ではな

[1] 鶴見大学
Tsurumi University
[2] 東京大学
University of Tokyo
[a)] tanabe-y@tsurumi-u.ac.jp
[b)] hagiya@g.ecc.u-tokyo.ac.jp

© 2023 Information Processing Society of Japan

いが，短冊式 [1] などの方式が提案されている．著者らも，プログラムの理解を問う選択肢による設問を組織的に生成する方法を提案している [2].

一方で，受験者のプログラミング能力を評価するのであるから，実際にプログラムを作成・提出させ，これを採点する試験の有用性は明らかである．本論文では，自動採点を含む，このタイプの試験の実施方式を提案し，試作したシステムを報告する．

自動採点の方式で広く用いられているのは，テスト用の入力を用意して，出力と比較するものである．本論文ではこれをテストスイート方式と称する．提案システムでも，テストスイート方式を利用することができる．しかし，この方式では，正しくないプログラムを誤って正解と判定してしまうことが避けられない．このため，提案システムでは，記号実行方式という，文献 [3] で提案した方式も使用できるようにしている．記号実行方式では，正しくないプログラムを正解と判定することはない．ただし，採点コストは高く，また，出題できる問題の範囲も相対的に狭い．

提案システムにおいては，受験者は，ビジュアルプログラミングを行う．すなわち，ブラウザ上で，グラフィカルなブロック部品を組み合わせる操作を行うことによって，指定された挙動を持つプログラムを組み上げるのである．著者らはすでに文献 [4] においてこの方式を提案していたが，受験者が答案を作成する際の手間が大きいという問題点があった．本提案では，出題者が，問題に応じた適切な粒度のブロック部品を作成する機能を提供して，この問題に対処している．他のいくつかの観点でも，ブロック部品方式には強みがあると考えられ，このことは本論中で議論する．

本論文の構成は次の通りである．第 2 節では，自動採点方式について述べる．第 3 節では，試験実施ステムについて，特にブロック部品に焦点を当てて説明する．第 4 節では，提案システムで採用しているブロック部品方式を，ソース記述方式と対比して論じる．第 5 節では，その他の設計上の論点について記述する．第 6 節は，まとめである．

2. 自動採点

提案するシステムでは，2 つの方式で自動採点を行う．

- テストスイート方式
- 記号実行方式

2.1 テストスイート方式

テストスイート方式は，従来から広く用いられている方法である．テスト用の多数の入力データと，それに対する正しい出力を用意する．受験者が提出したプログラムに対してそれらの入力データを与えて実行し，正しい出力と比較することによって，正解か否かを判断する．

この方式は汎用性があり，多くの問題に適用することができる．また，正しい提出プログラムを必ず「正解」と判断するという，基本的な要件を満足している．

一方で，正しくない提出プログラムを，正解と判断してしまうことがある．用意した入力データの範囲では正解と一致する出力を行うが，他の入力データで正しくない出力を行う (あるいはエラーを発生させる) プログラムに対して，これを不正解と判定することはできない．入力データを増やすことによって改善することは可能であるが，原理的な限界がある．どのようなプログラムに対しても，そのプログラムと 1 データの入力のみで異なる振舞をするプログラムを作成することが可能である．したがって，任意の誤ったプログラムを不正解と判定するためには，すべてのありうる入力を，この方式の入力データとして用意しておかなければならない．多くの問題において，これは現実的でない．

2.2 記号実行方式

記号実行 [5] は，プログラムの抽象実行方式であって，複数の入力に対する実行を, 実質的に一度に行うものである．

提案システムの自動採点における記号実行方式は，正しいプログラムをすべて正解と判定するだけでなく，原理的には，誤ったプログラムをすべて不正解と判定する．ただし，出題できる問題の範囲に制限があり，採点に要する時間も長い．

まず，プログラムのフローグラフを作成し，このすべての経路に関して，以下の処理を行う．

- 「経路条件」と呼ばれる条件式 P を保持する．経路条件の初期値は True である．
- 変数への入力を行うコードでは，具体的なデータは扱わない．その代わりに，変数の値として記号を割り当てる．以後，プログラムの変数の値は，これらの記号を用いた計算式となる．入力が満たすべき条件 φ があれば，経路条件を，$P \wedge \varphi$ に更新する．
- 代入文では，右辺の値を記号的に計算して，左辺の値として設定する．
- 条件分岐では，まず，経路条件 P を次のように更新する．分岐を判定する条件式の変数に，現在の変数の値である計算式を代入して得られる式を φ として，今検討している経路が条件分岐で Yes の枝である場合には，経路条件を $P \wedge \varphi$ で，No の枝である場合には，$P \wedge \neg\varphi$ で更新する．次に，更新された経路条件の充足可能性を，SMT ソルバを用いて判定する．充足不能である場合には，この経路の処理を終了する．
- 答を出力するコードでは，出力として指定された式の各変数に現在の値を代入して得られる計算式が，答の条件を満足する，という内容の条件式 φ を組み立て，条件式 $P \wedge \neg\varphi$ の充足可能性を判定する．充足可能であれば，提出された答案を不正解と判定する．

© 2023 Information Processing Society of Japan

いずれの経路の処理においても，不正解の判定がなされなかった場合には，提出された答案を正解と判定する．

記号実行方式を可能にするためには，SMT ソルバが，渡された条件式の充足可能性を判定できることが必要である．このために，第 2.4 で述べるプログラミング言語を設計した．この言語を対象とした問題であれば，記号実行方式による自動採点が可能である．

2.3 Presburger 算術

充足可能性判定ができる体系として，Presburger 算術 [6] がある．Presburger 算術とは，整数の集合に，順序と加法を入れた体系の理論 $\mathrm{Th}(\mathbb{Z}, 0, 1, =, +, -, <)$ のことである．この理論に，すべての正整数 c に対して，「c で割りきれる」ということを表す述語を付け加えると限量子除去が可能であり，このため，この理論は決定可能である，ということが知られている．

Presburger 論理式に対して，その充足可能性を判定するための理論的計算量はかなり大きい．論理式が一般の場合には，知られている上限は 2EXPSPACE 程度であって，これは，空間計算量が二重指数的であることを意味する．しかし，この計算量の上限は，変数の数を限ったり，限量子交代の回数に制限を付けたりすることによって減らすことができる．例えば，1 つの限量子ブロック中に現れることができる変数の数と不等号の数の上限を固定した場合には，Π_2 論理式に対する決定手続きは多項式時間であることが知られている [7].

理論的な検討が行われているだけでなく，Z3[8] など多くの SMT ソルバが，Presburger 算術の決定手続きを実装している．また，Princess[9] のように，Presburger 算術に特に焦点をあてて，充足可能性判定や interpolation などの機能を提供するソルバも存在する．

2.4 記号実行用言語

この節では，記号実行方式による自動採点を可能にするプログラミング言語を概観する．詳細は文献 [3] を参照されたい．

図 1 に，この言語の文法を示す．基本的には破壊的代入を持つ手続き型のプログラミング言語であるが，記号実行時に生成される条件式が Presburger 算術式になるように，さまざまな制約が加えられている．

- 基本データ型は整数型のみであり，実数型や文字列型を持たない．配列を持つが，配列長は定数でなければならない．
- 算術演算は，加法のみである．特に乗法を持たない．ただし，定数倍は加法の繰返しなので，表現可能である．また，整数による除算 (商と余りを求める) は，while 文を用いて表現可能である．
- if 文は，通常の記述ができる．

```
プログラム = { 宣言 } , { 文 }
宣言 = ( "var" | "input" )
     , 変数宣言 , { "," , 変数宣言 } , ";"
変数宣言 = 変数 | ( 配列 , "(", 定数 , ")" )
文 = ブロック | 入力文 | 代入文 | 条件文 | 繰返し文
   | break 文 | 条件繰返し文 | assert 文 | assume 文
ブロック = "{" , { 文 } , "}"
入力文 = "input" , ( 変数 | 配列 )
代入文 = ( 変数 | 配列要素 ) , ":=" , 表現 , ";"
条件文 = "if" , 条件式 , 文 , [ else , 文 ]
繰返し文 = "repeat" , 定数 , 文
break 文 = "break" , ";"
条件繰返し文 = "while" , 条件式 , 限定ブロック
限定ブロック = "{" , { 限定代入文 } , "}" , ";"
限定代入文 = ( 変数 | 配列要素 ) , "+=" , 定数 , ";"
assert 文 = "assert" , 拡張条件式
assume 文 = "assume" , 拡張条件式
条件式 = 原子条件式 | "not" , 条件式
       | 条件式 , "or" , 条件式
原子条件式 = 表現 , "<" , 表現
拡張条件式 = 条件式 | "forall" , 変数 , 条件式
表現 = 定数 | 変数 | 配列要素
     | "-" , 表現 | 表現 , "+" , 表現
配列要素 = 配列 , "[" , 表現 , "]"
定数 = ? 任意の整数 ?
変数 = ? 任意の識別子 ?
配列 = ? 任意の識別子 ?
```

図 1 記号実行用言語の文法

- 繰返し文は，定数回の繰返しのみが許される．ただし，繰返し部で break 文を使うことはできる．
- while 文は，条件部には制約はないが，ブロック部で使用できる文に制約があり，「変数に定数を加える」文のみを書くことができる．
- assume 文が用意されており，Presburger 算術式を指定できる．入力データに関する制約条件を表現するために用いることができる．
- assert 文が用意されており，Presburger 算術式を指定できる．出力データが満たすべき条件を表現するために用いることができる．

3. 試験実施システム

この節では，試作した試験実施システムについて述べる．

3.1 概要

試作システムは，ブラウザ上で動作するウェブアプリケーションである．受験者は利用者名とパスワードを使ってシステムにログインすると，問題リストを閲覧できるようになる．状況によっては，管理者の操作によってはじめて閲覧できるようにすることもできる．

各問題のページには，その問題において利用可能なブ

© 2023 Information Processing Society of Japan

```
in_out = Statement(
  %body = Statement(),
  %varname = Select("x", "y", "t"),
  expr = ["リスト xs にデータを入力",
          "%body",
          "変数 %varname の値を出力"],
  code = ["xs = list(map(int, input().split()))",
          "%body",
          "print(%varname)"],
  colour = "180"
)
```

図 2 ブロック部品の定義

図 3 作成されたブロック部品

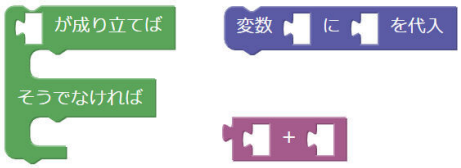

図 4 基本ブロック部品

ロック部品の一覧が掲示されている．受験者は「ワークスペース」にその部品を複写し，必要に応じて適切にリストから値を選択し，部品を組み合わせることによってプログラムを作成する．

受験者は，作成したプログラムに，その場で入力を与えて実行することができる．ブラウザ上で実行ボタンが押されると，システムは組み合わされたブロック部品から，Python 言語でソースコードを生成し，入力データとともにサーバに送付する．サーバ側では，Ubuntu + Python の docker インスタンスを一つ生成し，その中で，ソースコードに入力データを与えて実行する (第 5.2 節を参照)．得られた出力をブラウザに送り返すことで，ユーザに実行結果を表示する．

正しいプログラムが書けたと判断したら，プログラムを提出する．ブラウザ上で提出ボタンが押されると，システムは組み合わされたブロック部品から，ソースコードを生成し，サーバに送る．ソースコードは，自動採点方式に応じて，Python 言語または記号実行用言語のいずれかで生成される．

サーバでは，ブラウザから送られたソースコードを保存する．現在の実装では，採点は，独立したプログラムによって，バッチ的によって行われる．

3.2 Blockly

試作システムでは，ブロック部品を Blockly を用いて実現している．Blockly[10] は，Google が開発しているビジュアルプログラミング環境のためのライブラリである．ブロック部品を作成することができ，これらの部品から Python や JavaScript 等のコード片を生成することができる．本システム作成のため，記号実行用のソースコード片を生成するためのコードを追加した．

3.3 ブロック部品の定義

本システムでは，ブロック部品は，問題ごとに出題者が作成する．これを可能にするために，ブロック部品を定義するための簡易言語を設計した．出題者は，問題で必要となるブロック部品を，次節に述べる基本ブロック部品から選択するか，もしくは，この簡易言語を使って定義する．後者の例を，図 2 に示す．また，この定義から生成されたブロック部品の外観を図 3 に示す．

定義においては，キーワード expr を用いて，部品外観上に表示される文字列や「穴」などを指定する．キーワード code を用いて，この部品から生成されるコードを指定する．キーワード colour を用いて，部品の色の色相を指定する．

%で始まる識別子は，利用者が定義する名前である．この定義では，%body は Statement であると指定されているので，生成されるブロック部品では穴として表現され，そこには文を表す他のブロック部品を，受験者がはめ込むことができる．%varname は Select であると指定されているので，ブロック部品上ではドロップダウンリストとなり，選択肢は x，y，t となる．

定義が完了したら，用意されたスクリプトを実行することで，ウェブサーバに配置できるファイル群が生成される．これらをサーバ上の適切な位置に配置することによって，ブロック部品が問題ページに表示される．

3.4 基本ブロック部品

基本的なブロック部品については，事前に定義されており，出題者は，自分の定義ファイルに include をして用いることができる．

図 4 に，基本ブロック部品の例を 3 つ示す．

4. ブロック部品方式とソース記述方式

ソースコードを提出させる方式と比較して，ブロック部品を組み合わせてプログラムを作成させる方式には次の特長があると考えられる．

- 詳細文法知識を不要にできる
- 一部機能を隠せる
- 要求する能力を制限できる

© 2023 Information Processing Society of Japan

ある惑星では，次のような暦を使っている．

・1 年は 1 月，2 月，...，9 月の 9 箇月からなる．
・月は 1 日から始まる．
・月の日数は次の通り．
  ・2,4,6,8 月は 37 日．
  ・3,5,7,9 月は 38 日．
  ・1 月は，平年は 35 日で，閏年は 36 日．
・y 年が閏年であるかどうかは，次のように決定される．
  (a) y が 3 でも 5 でも割り切れれば平年．
  (b) y が (a) を満たさず，3 または 5 で割り切れれば閏年．
  (c) y が (b) も (c) も満たさなければ平年．

年と月と日を y と m と d に入力し，
y 年 1 月 1 日 から，y 年 m 月 d 日までの日数を
出力するプログラムを作成せよ．
1 月 1 日と m 月 d 日は含めて数える．

図 5 例題

```
y, m, d = map(int, input().split())
ans = 0
for i in range(1, m - 1 + 1):
    if i == 1:
        if y % 3 == 0 and y % 5 == 0:
            ans += 35
        else:
            if i % 3 == 0 or i % 5 == 0:
                ans += 36
            else:
                ans += 35
    else:
        if i % 2 == 0:
            ans += 37
        else:
            ans += 38
ans += d
print(ans)
```

図 6 Python によるプログラム

図 7 粗粒度ブロックによるプログラム

- 難易度調整ができる

各項目を以下で議論する．

4.1 詳細文法知識を不要にできる

著者らがブロック部品を組み合わせる方式を採用した直接の動機は，記号実行用言語の文法にある．この言語は，記号実行の課程で生成される条件式を，Presburger 算術の枠内におさめるために，文法的な制約が導入されている部分がある．典型的なのが，文法要素「条件繰返し文」である．

C 言語などの通常の言語では，while 文のブロック内には任意の文を記述することができる．しかし，本言語ではこのブロック内には限定代入文しか記述することができない．限定代入文とは，変数の値を定数だけ増加・減少させる文のことである．たとえば `v += 3;` は限定代入文である．一方，`v += u;` や `v = 3;` は限定代入文ではないので，while 文のブロック内に記述することはできない．詳細な定義は図 1 を参照されたい．

このような微妙な文法を，特に，本提案がその主たる対象としているプログラミング初心者に正確に理解させることは難しい．ブロック部品を組み合わせる方式では，while 文のブロックには，限定代入文以外の文は接続できないように制約を課すことが可能である．この機能によって，受験者はある文が while ブロック内に置けるかどうかを，実際にブロック部品を組み合わせてみることで判断できる．限定代入文以外では，組合せが拒否されるので，その位置には置けないということがわかる．

4.2 機能を隠す

現実に用いられるプログラミング言語には，さまざまな機能があり，ライブラリなどの形で実現されている．このことが，プログラミング試験の観点からは問題となることがある．たとえば，多くの言語は標準ライブラリとして整列機能を提供しているが，このために，整列アルゴリズムの理解を問うプログラミング試験問題を作成しにくくなっている．「sort 関数を用いないで解け」などと指示することは可能かもしれないが，自動採点は難しい．

ブロック部品方式では，受験者が利用できる機能を作問者が指定することができるので，この問題は発生しない．

© 2023 Information Processing Society of Japan

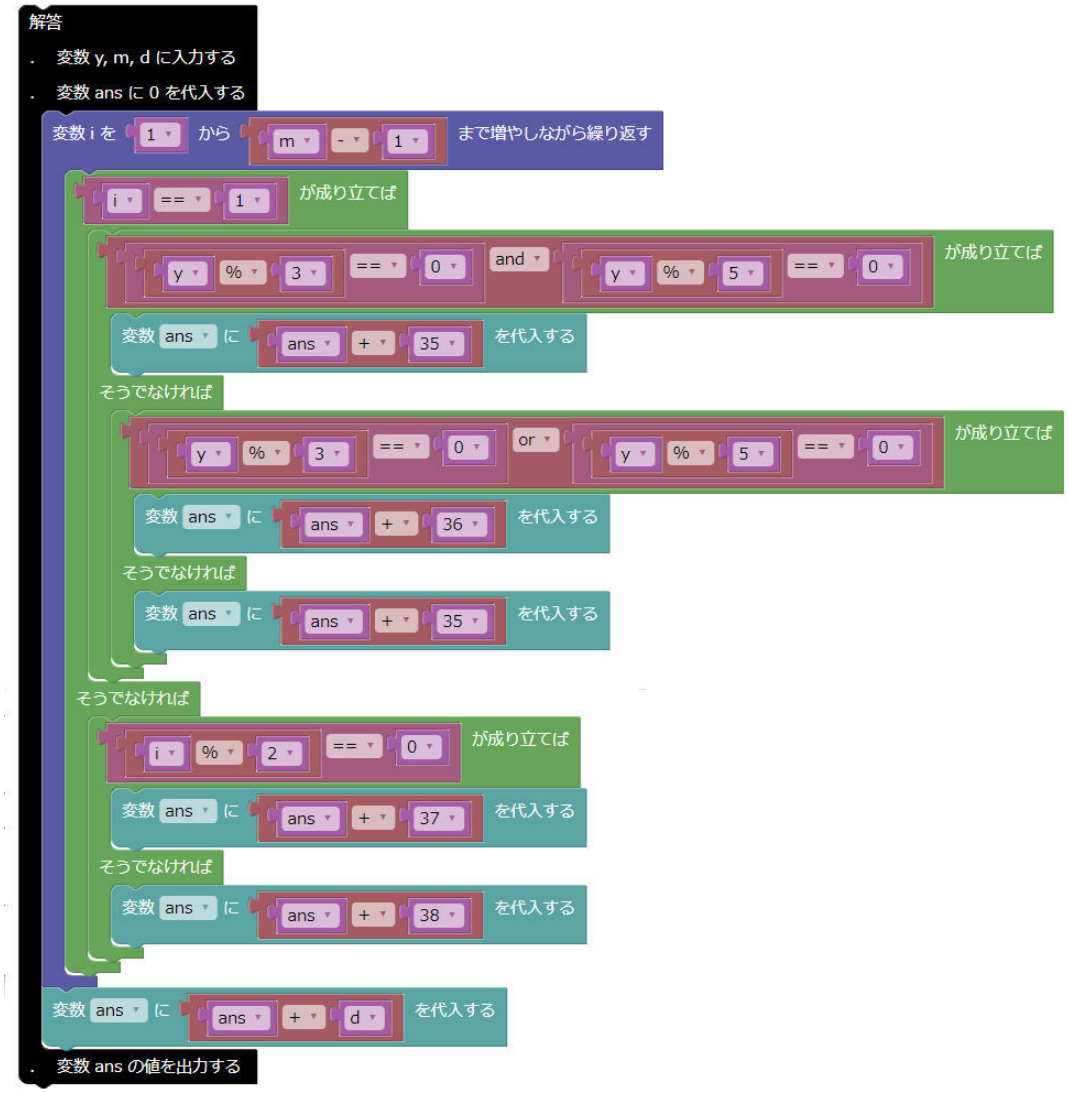

図 8 細粒度ブロックによるプログラム

4.3 要求能力の制限

前項目とは逆に，言語が単独では持っていない機能を受験者に提供することも可能である．

たとえば，Python 言語による試験において，「スペース区切りで 1 行で与えられた整数の並びをリストに読み込み，そのリストにしかじかの処理を行った結果を出力せよ」という問題が出されたとする．出題者の意図は，しかじかの処理を行う能力の判定にある．しかし，受験者が入力を処理するコードを書けないと，本来の問題の部分をプログラムする能力があっても，不正解となってしまう．

ブロック部品方式では，「データをリスト xs に読み込む」というラベルを持ち，`xs = list(map(int, input().split()))` というコードを生成するブロック部品を用意することで，問いたい知識・能力に焦点をあてた問題とすることができる．

4.4 ブロック粒度による難易度調整

同じ仕様のプログラムを作成させる問題であっても，受験者に使用させるブロック部品の「粒度」によって，問題の難易度や，受験者が解答するために要する時間を調整することができる．

図 7 および図 8 は，図 5 に示す同一の例題の，同じアルゴリズムによるプログラムである．図 7 は，「粗い粒度」のブロック部品群が与えられた場合の解答，図 8 は，「細かい粒度」のブロック部品群が与えられた場合の解答である．Python でプログラムを作成すれば，図 6 のようになるであろう．実際，図 6 は，図 7 のプログラムから，Python コードを生成したものである．

前者では，受験者の組合せ操作の負荷は少ない．たとえば，2 番目の if 文の条件部「`y % 3 == 0 and y % 5 == 0`」を組み立てるために必要な操作は，1 つのリストから項目を選ぶだけである．一方で，たとえば最初に考えた処理が，まず「`y % 3 == 0`」で場合分けして yes の場合さら

© 2023 Information Processing Society of Japan

に「`y % 5 == 0`」で場合を分ける，というものであったとき，そのような選択肢は無いので，処理を考え直す必要がある．逆に，独力では処理を思いつけなくても，この選択肢を見ることで，正解に到達してしまうことがあるかもしれない．

後者では，ブロック部品を多数組み合わせる操作が必要となる．上と同じ条件部を作るためには，多数のブロック（論理演算子 1，比較演算子 2，算術演算子 2, 変数 2, 数値 4）を組み合わせた上で，各リストから適切な値を選ばなければならない．ブロック操作にある程度習熟すればコピー・ペーストを駆使するなどして短時間で組み上げられるが，慣れないと時間がかかる．しかし，自分の考えた処理がそのまま書ける可能性は，前者より大きい．

5. 設計上の論点

この節では，試作システムの設計の際に考慮した点について述べる．

5.1 実行環境

一般に，プログラムを作成する際に，実行することなしに正しいプログラムを得ることは困難である．

試験という限定された状況においては，適切に難易度を調整することによって，実行することなしに正しいプログラムを得られることをもって受験者の能力・達成度を判定できるとの考え方もあろうし，実際の開発環境に合わせて，試験でも受験者に実行環境を与えるのが良いとの考え方もあろう．本試作システムでは，後者を採用した．

実行時にいわゆるプリントデバッグができるように，出題者は，変数の値を表示するブロック部品を用意しておくことも可能である．

実行時にエラーが発生した場合，発生位置（ブロック部品）を表示するように実装することも可能であろうが，本試作システムではまだ実現されていない．

5.2 セキュリティ

提出されたソースコードを実行するタイプの自動採点方式には，セキュリティ上の問題がある．受験者は任意のコードを書くことができるので，これを何の対策も無く実行するのは危険である．このため，提出コードの実行時に仮想マシンを用いたり，サンドボックスを用いるなどの工夫が行われてきた．要求するリソース量などを勘案して Docker が広く用いられている [11].

本システムの場合，受験者はブロック部品を組み合わせてプログラムを作成するので，ブロック部品が生成するコードが安全であれば，上記のような問題は発生しない．

しかし，本システムでは，出題者がブロック定義を行う．システム管理者以外の利用者が出題者になる場合には，危険なブロック部品が利用可能になる可能性がある．このことを考慮して，本試作システムでは，自動採点の際のコード実行は，Docker 内で行うこととした．

6. おわりに

本論文では，プログラミング試験の実施環境を提案した．特長としては，受験者がブロック部品を組み合わせてプログラムを作成すること，出題者がブロック部品の粒度を制御できること，自動採点の方式として，記号実行に基づく，誤判定の無いものが使える，ということがあげられる．

今後は，本システムを実際に試験に用いることで，ブロック部品方式の利害得失を，より具体的に評価していくことを予定している．

謝辞　本研究は JSPS 科研費 20K12106 の助成を受けた．

参考文献

[1] 久野靖. 情報入試研究会試作問題#001 問題解説. 情報入試フォーラム 2013 資料集, pp. 4–10. 情報入試研究会, 2013.

[2] Masami Hagiya, Kosuke Fukuda, Yoshinori Tanabe, and Toshinori Saito. Automatically generating programming questions corresponding to rubrics using assertions and invariants. In Arthur Tatnall and Nicholas Mavengere, editors, *Sustainable ICT, Education and Learning*, pp. 89–98, Cham, 2019. Springer International Publishing.

[3] 田辺良則, 萩谷昌己. 試験問題の自動採点が行えるプログラミング言語の検討. 信学技報 (KBSE2020-42), Vol. 120, No. 423, pp. 48–53, March 2021.

[4] 田辺良則, 萩谷昌己. 初級者を対象としたプログラミング試験のためのシステムについて. 信学技報 (KBSE2021-42), Vol. 121, No. 424, pp. 7–11, March 2022.

[5] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, Vol. 51, No. 3, 2018.

[6] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa, Warsaw, Poland*, Vol. 1929, pp. 92–101, 395, 1930.

[7] Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, Vol. 5, No. 3, p. 67–82, July 2018.

[8] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[9] Philipp Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In *Proceedings, 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Vol. 5330 of *LNCS*, pp. 274–289. Springer, 2008.

[10] Neil Fraser. Ten things we've learned from blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pp. 49–50, 2015.

[11] František Špaček, Radomír Sohlich, and Tomáš Dulík. Docker as platform for assignments evaluation. *Procedia Engineering*, Vol. 100, pp. 1665–1671, 2015. 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014.

© 2023 Information Processing Society of Japan