情報処理学会研究報告
IPSJ SIG Technical Report

空欄補充問題の自動生成による Haskell プログラミング学習支援環境

竹内　亮太郎[†1]　大久保　弘崇[†2]
粕　谷　英　人[†2]　山本　晋一郎[†2]

Haskell のプログラムから空欄補充問題を自動生成し，また回答に対して正誤判定を自動的に行う手法を提案する．空欄設定は，プログラム中の関数の重要度や，Haskell プログラミングで重要な再帰呼出構造，fold 関数の使用などを考慮して効果的な穴開け箇所を選定する．正誤判定は関数に対する単体テストを利用する．本手法に基づいて，学習者が Haskell プログラミングを自習できるシステムを Web アプリケーションとして実装した．学習者は本システムを用いることで，自ら興味があるプログラムを選んで学習できる．実装には，関数の説明文の提示，回答に誤りがある場合の対応など，学習がより効果的に行える機能が盛り込まれている．

An Learning Support Environment for Haskell Programming by Automatic Generation of Cloze Question

Ryotaro Takeuchi,[†1] Hirotaka Ohkubo,[†1]
Hideto Kasuya[†1] and Shinichiro Yamamoto[†1]

This paper proposes a method generating cloze questions from Haskell programs and checking their answers. Our method selects effective perforator points, considering importance of each function, and significant things in Haskell programming such as recursive call structure or use of fold functions, we employ unit tests to check the correctness of answers the user entered. We have implemented a self-instructional system of Haskell programming as web application. The system also provides useful facilities such as hyper-documentation of Haskell functions and error recovery tips.

1. 序　　論

プログラミング教育においてプログラミング演習の果たす役割は大きい．プログラミング演習では多くの問題を解き，多くのプログラムに触れることで学習効果が高められる．そのプログラミング演習の出題形式には，主に以下の 3 種が挙げられる．

- 記述問題: 与えられたプログラムの仕様に従いゼロから作成する
- 空欄補充問題: プログラム中に空欄が設定されていて，空欄を埋めてプログラムを完成させる
- 選択式問題: プログラム中に空欄が設定されていて，その部分に最も当てはまるものを選択肢から選んでプログラムを完成させる

記述問題はプログラムをゼロから記述する能力を強化することを目的とする．空欄補充問題と選択式問題はいずれもプログラムの大枠を示した上で補う形式である．この問題形式は様々な段階の学習に対応できる．言語の予約語や特殊記号の単位での空欄化は，文法を学習する初学者に有効である．再帰呼び出しなど言語の機能を活かした問題解決方法のスタイルを学習するには，スタイルを意識した箇所への空欄設定が考えられる．この段階の学習者は，優れたプログラムを見習うことで多くのことを学び取ることができるが，手本となるプログラムをただ読むのではなく空欄補充問題とすることでより積極的な理解が得られると期待できる．より上級者向けにはまず空欄のサイズを大きくすることで難易度を変化させることができる．また，題材とするプログラムの質をあえて下げることで，プログラムの意図をコードから読み取るプログラム読解能力を養う効果が期待できる．この読解能力はソフトウェア保守の場面で現実的に役立つ技能である．

本研究は Haskell プログラムの学習支援を目的とする．そのために Haskell のプログラムから空欄補充問題を自動生成し，また回答に対して正誤判定を自動的に行う手法を提案する．本研究の大きな目標の一つとして，効果的な空欄設定箇所を決定するアルゴリズムの開発がある．先に述べたとおり，さまざまな段階の学習者に対して効果的な空欄補充問題は異なる．それらを自動生成するためには，例えばプログラムの一部分をランダムに選ぶのではなく，何らかの指標に基づいて空欄設定箇所を決定する必要がある．

†1 愛知県立大学大学院 情報科学研究科
Graduate School of Information Science and Technology, Aichi Prefectural University
†2 愛知県立大学 情報科学部
School. of Information Science and Technology, Aichi Prefectural University

また学習者の学習意欲を向上するために，本研究で実装したシステムは学習者が入力した Haskell プログラムから自動的に空欄補充問題を作成し，正誤判定を自動的に行う機能を提供する．これにより学習者は自ら興味があるプログラムを選んで学習することができる．問題作成者が用意した問題を受動的に解いていくという従来の学習形式とは異なり，能動的な学習形式を提供できるため，学習意欲の向上に繋がる．

2. 空欄補充問題の自動生成に関する先行研究

プログラムの空欄補充問題とは,「プログラムの説明」と「空欄が設定されたプログラム」を提示し，空欄の補充を要求する問題のことである．空欄補充問題の自動生成に関する研究を紹介し，本研究に既存研究の手法が応用可能かどうかを議論する．

2.1 初級プログラミング学習のための自動作問システム

「初級プログラミング学習のための自動作問システム」[9] で試作された自動作問システムがある．対象言語は Java である．問題作成者はプログラムの説明とプログラムをシステムに登録する．システムはキーワードや識別子を自動的に抽出し，その部分に空欄を設定する．空欄は乱数によって毎回異なる場所に設定されるため，反復練習を行える．

このシステムはプログラミング初心者を対象とし，アルゴリズムをプログラムとして記述する際に必要な，プログラム言語によって規定されている書き方やキーワードの知識の学習を支援する．

2.2 学習者に合わせた C 言語演習穴埋め問題の自動生成

「学習者に合わせた C 言語演習穴埋め問題の自動生成」[10] では，プログラムを解析し，そこに問題作成者の作成意図を合わせて作問することにより，問題作成者の意図に沿った演習問題の自動生成をする．

2.2.1 問題作成意図

問題作成意図とは，問題作成者が「こんな目的で問題を作りたい」と考えている内容であり，ある学習目標に対し,「正解できれば，これが理解できている」という指針になるものである．問題作成意図の例には「出力文が分かっているのか確認したい」,「for 文の引数が分かっているのか確認したい」,「処理の流れが分かっているのか確認したい」等があげられる．

問題作成意図は問題作成ルールの組み合わせとしてコーディングされる．問題作成ルールは以下のような記述を持つものになる．

createblank(構文木の節, all|select (, 指定))

第一引数で指定した節以下を空欄とする．第二引数が “all” の場合はすべてを空欄にする．“select” の場合は第三引数で指定した部分だけを空欄にする．指定内容は，枝名で指定した部分の中でも一部分にだけ穴を開けたい場合に指定するのに用いる．

2.3 いかにプログラム空欄補充問題を作るか？

「いかにプログラム空欄補充問題を作るか？」[8] では，C 言語を対象に PDG(Program Dependence Graph)[3] を用いてプログラムの処理過程の要所を特定し，それを空欄とする方法について述べている．PDG を用いることにより，アルゴリズムの内容まで立ち入らずに，形式的に空欄を設定している．

PDG では，ノードに出入りするアークの種類・本数によって処理過程における各文の重要度を定量的に判断できる．アークの種類・数が多いノードほど処理の中心である文と考えられる．このようなノードを空欄とすることで，そのノードと依存関係にある文まで参照するように仕向けることができ，アルゴリズムを考えさせることができるとしている．

空欄設定方針の妥当性を調べるために，プログラミング経験がありアルゴリズムを熟知した理工系大学生，大学院生および大学教員 10 名にプログラム中の一文を空欄として選択させ，これと空欄設定方針を比較に妥当性を評価している．実験結果として最多あるいは 2 番目に多い文が選択されており，空欄設定方針が妥当であるという見通しを得ている．

2.4 まとめと他言語への応用

ここまで既存研究を 3 つ紹介した．これらの手法が本研究に応用できるかどうか検討する．

文献[9] の手法は文法を覚えることを目的にキーワードや識別子を対象としてランダムに空欄を設定する．キーワードや識別子を抽出しそこを空欄とする手法は言語非依性はない．我々は文法を学習する初心者ではなく中級者を対象とするため，キーワードにランダムに空欄を設定する方法は適さないと考える．

文献[10] の手法は，問題作成者が問題作成意図とプログラムを入力し，それに基づいて空欄を設定する．問題作成ルールの枝名は構文解析木の節であるため，言語依存である．しかしそれ以外は言語非依存であるため，少しの変更で他言語に応用可能である．また学習者の回答結果により，学習者の理解が不十分なところを見つけ，次の出題時に苦手な部分の問題を重点的に学習を行わせるフィードバック機能は学習に有用である．しかし本研究では学習意欲の向上のために，学習者が自ら興味があるプログラムを選びそれを空欄補充問題にでき

ることも目指すため，この手法ではその場合に対応ができない．

文献[8] の手法は PDG を用いて処理過程の要所を特定し，そこを空欄とする．処理過程の要所の特定に，Dependence Graph を用いるため，Dependence Graph に関する研究がされている言語には応用可能であると考えられる．遅延評価関数型言語向けの依存グラフというのはないので，直接はこのアイデアは使えない．ただし，プログラムの部分の重要度を独自の評価基準で判断して空欄設定箇所を選ぶという意味では我々のアプローチに最も近い研究といえる．

3. Haskell の空欄補充問題の自動生成

前章で既存研究を本研究に応用することが困難であることを述べた．そこで我々はまず，Haskell の空欄補充問題における適切な空欄設定箇所を議論する．

Haskell プログラムは関数定義の集合である．プログラムの意味を考えさせるという観点から空欄設定箇所の選定を考えるとき，問題を「どの関数に空欄を設定するか」,「関数記述のどの部分を空欄とするか」の 2 段階に分割する．本研究では前者を「マクロな空欄設定箇所」，後者を「ミクロな空欄設定箇所」と呼ぶ．

3.1 マクロな空欄設定箇所

プログラムは通常複数の関数からできている．そのいずれもがプログラムの中で同一の重要度ということはなく，プログラムの意味を理解させるための問題を作成するという目的のためには，プログラムの中でより本質的な処理を行なっている関数を対象とすべきである．この「本質的な処理」の近似として本研究では関数の複雑さを用いる．

D.E.Knuth は，プログラムが行う計算の難しさを抽象機械の実行ステップ数として形式化している[2]．Knuth が論じているプログラム及び抽象機械は手続き的なものであるが，これにヒントを得て本研究では関数を STG 言語 (Spineless Tagless G-machine[5] 上の言語) に変換したときの行数を複雑さの尺度として用いる．STG 言語とは Glasgow Haskell Compiler[7] の内部で使用されている遅延評価を行う関数型言語向けの抽象機械であり，ステップ実行により状態遷移して計算を行う．

また Haskell のプログラムでは，実行部の記述だけでなく型も重要である．Haskell ではユーザが新しい型を定義したり，ある型を別の名前で定義することができる．そこで型の定義部に空欄を設定することにより，型を理解しているかどうかを確認できる．本研究では使用されている回数が多い型ほど重要度が高いと考える．

3.1.1 空欄設定の対象外とする箇所

プログラム中の重要でない箇所は空欄設定の対象外とする．本研究では入出力などの「明らかに処理の流れの要所となっていない箇所」を対象外とする．

入出力処理　プログラムは大きく分けて (1) データの入力，(2) データの処理，(3) データの出力の 3 つに構造を分割できる[6]．本質的な機能を実現しているのは 2 の「処理」である.「入力」と「出力」の部分は，プログラムを理解する上では基本的に重要でないと考えられるので空欄設定の対象外とする．

`main` 関数　`main` 関数は完結した Haskell プログラムのエントリポイントである．プログラムを理解しようとするとき，`main` 関数から順番に処理を追っていくと考えられる．しかし，`main` 関数に空欄を設定すると開始地点において処理の流れが隠されてしまうため，ヒントが少なく空欄設定箇所として適切ではないと考えられる．

3.2 ミクロな空欄設定箇所

空欄補充問題を解くときの思考を分析する．プログラムの実行部にあけられた空欄を補充するには，与えられたプログラムの見えている部分からデータや制御の流れに関する情報を取得し，プログラマの意図を理解する必要がある．したがって，処理の流れの要所となる箇所を空欄にすることでデータや制御の流れを考える機会が増え，既存プログラムに対して理解する能力を高めることができる．

ミクロな空欄設定箇所は大きく分けて，型に対するものと関数に対するものが考えられる．本研究では関数に対するミクロな空欄設定箇所を 3 種類に分類した．

3.2.1 型に対するミクロな空欄設定箇所

Haskell において，関数の型を理解することはその関数の理解につながる．なぜなら関数の型を理解することによって，どんな値を扱う関数であるかを理解することができるからである．つまり関数に対する理解度を確認するうえで，型について理解しているかどうかを確認することは有用である．

例として，加算のみの計算式を表す `Expr` 型を図 1 に示す.【】は空欄を表す.【】の中の A,B,C は空欄のラベルを表し，( ) の中は空欄に入る解答を表す．図 1 は `Expr` 型のデータコンストラクタ `Val` と `Add` の引数を空欄としている．空欄 A を埋めるには，`Expr` 型を使用している `value` 関数の 5 行目に注目し，`(Val n)` の `n` が何かを考える．そして 4 行目の関数の型シグネチャより `Int` であることが分かる．空欄 B と C に関しても 6 行目で同様に分かる．このように型宣言のデータコンストラクタの引数を空欄にすることで，その型を使用している関数から情報を収集し考える機会を作られる．

```
1  data Expr = Val【A (Int)】
2            | Add【B (Expr)】【C (Expr)】
3
4  value :: Expr -> Int
5  value (Val n) = n
6  value (Add x y) = value x + value y
```

図 1 加算のみの計算式を表す Expr 型
Fig. 1 datatype and evaluator of simple numeric formula of addition

```
1  qsort [] = []
2  qsort (x : xs) = 【A (qsort smaller)】++ [x] ++ qsort larger
3    where smaller = [a | a <- xs, a <= x]
4          larger  = [a | a <- ys, a > x]
```

図 2 クイックソートを行う関数
Fig. 2 QuickSort function

newtype 宣言で定義された型も data 宣言で定義された型と同様に空欄を設定する．type 宣言においては型を空欄とする．

3.2.2 関数本体

再　　帰

自分自身を呼び出す関数を再帰関数という．再帰を使用している簡単な例として図 2 にクイックソートのプログラムを挙げる．3 行目の `smaller` ではリスト内包表記を用いている．これは「リスト `xs` の各要素 `a` について `a <= x` を満たすものを集める」という意味である．2 行目で `qsort` を再帰的に呼び出し，クイックソートのアルゴリズムを実装している．3,4 行目で where 節を用いて局所関数 `smaller` と `larger` を定義している．

Haskell では多くの計算が再帰を用いるとスマートに記述できる．しかし，人間にとって再帰を理解することは困難な課題であることが知られている[11]．本研究ではその「理解が困難な箇所」である再帰呼び出し箇所を空欄にすることで，その理解を促す効果を狙う．図 2 では `qsort` 関数の `qsort smaller` に空欄を設定した．図 2 の空欄 A を埋める際には，クイックソートのアルゴリズムから考える．クイックソートは `x` より小さい数を前方，大きい数を後方に移動し，分割された各々のデータをそれぞれソートしたものを結合する．空欄 A は `x` より小さい数をソートしていると想定されるため，`qsort smaller` が入るということがわかる．

```
1  filter _ []     = []
2  filter f (x:xs) = if (f x) then (x : (【A (filter f xs)】))
3                              else (filter f xs)
```

図 3 指定した関数に合致する要素だけを集めたリストを返す関数
Fig. 3 Function that returns list that collects only elements that agree with specified function

`map` 関数と `fold` 関数

Haskell では for のような繰り返しを表す構文が存在しない．そのため複数回繰り返す処理を行う場合は再帰を用いる必要がある．繰り返しは多くのアルゴリズムで頻繁に用いられる．簡単にプログラムを記述するために，Haskell では繰り返しによってリストを操作する関数があらかじめいくつか定義されている．特に，引数として与えられた関数をリストの要素すべてに適用する `map` 関数と，引数として与えられた関数でリストの畳み込みを行う `fold` 系関数が重要である．第一引数の関数を空欄とすることで，リストの各要素にどのような処理を行う必要があるのかを考えることを促す．

分岐に共通して出現する式

パターンマッチなどの分岐に共通して出現する式は条件によらず必要な処理であり，その関数の本質的な処理である可能性が高いと考えられる．例として `filter` 関数を図**??**に示す．`filter` 関数は第二引数のリストの要素 `x` のうち，`f x` が True である要素だけを集めたリストを返す関数である．2 行目と 3 行目において，`filter f xs` が共通して出現している．共通箇所を空欄化した例を図 3 に示す．

3.2.3 局所関数

局所関数を使用している例として図 2 を挙げる．局所関数はその局所関数を定義している関数でのみ使用できる関数であるので，特徴的な処理が記述されることが多い．`qsort` 関数の局所関数 `smaller` と `larger` は `x` より小さい数，大きい数を表すので，`qsort` 関数において特徴的な処理といえる．局所関数定義を空欄にすることにより，その局所関数を定義している関数における特徴的な処理の理解を問うことができる．

3.2.4 データフローの分岐

データフローが分岐している箇所では，それぞれの分岐で異なる計算を行う．分岐は条件式によって決定するので条件式を理解することは重要である．条件式に空欄を設定すること

表 1 対象プログラム
Table 1 Target program

番号 (名前)	内容	行数	関数の数
P1(Queens)	N-Queen 問題を解くプログラム	18	4
P2(IntAtBase)	10 進数を N 進数に変換するプログラム	29	4
P3(Pajama)	パジャマパーティのパズルを解くプログラム	32	4
P4(Merge)	マージソート	15	2
P5(Insert)	挿入ソート	11	2
P6(Caeser)	シーザー暗号で文字列を暗号化するプログラム	15	2
P7(Invert)	大文字・小文字を反転するプログラム	19	6
P8(Invert2)	P7 を改良したプログラム	18	5
P9(Sophie)	ソフィー・ジェルマン素数を求めるプログラム	13	4
P10(HappyNumber)	HappyNumber を判定するプログラム	13	2
P11(HappyNumber2)	P10 を改良したプログラム	11	2
P12(Ramanujan)	Ramanujan 数を求めるプログラム	19	5
P13(VM)	整数，加算，乗算からなる数式を解く仮想マシン	20	3
P14(Stack)	スタック	20	5

```
1  module Queens where
2
3  queens :: Int -> [[Int]]
4  queens number_of_queens = qu number_of_queens where
5    qu 0     = [[]]
6    qu (m+1) = [p ++ [n] | p <- qu m, n <- [1..number_of_queens], safe p n]
7
8  safe :: [Int] -> Int -> Bool
9  safe p n = all not [check (i, j) (m, n) | (i,j) <- zip [1..] p]
10   where m = 1 + length p
11
12 check :: (Num t) => (t, t) -> (t, t) -> Bool
13 check (i, j) (m, n) = j == n || (i+j == m+n) || (i-j == m-n)
14
15 q :: Int -> IO ()
16 q = putStr . unlines . map show . queens
```

図 4 N-Queen 問題
Fig. 4 N-Queen Puzzle

でデータフローについて考える機会が増える．

3.3 妥当性確認のための実験

上述の「マクロな空欄設定箇所」及び「ミクロな空欄設定箇所」の仮説の妥当性を検証するため，空欄補充問題を手作業で作成した．使用したプログラムを表 1 に示す．妥当性の確認は我々の研究室の学生学部 4 年から大学院 2 年の学生 9 名が問題を解いた後に議論する，という方法で行った．例として P1 を以下に示す．

3.3.1 作成した空欄補充問題の例 (P1)

N-Queen 問題は N 個のクイーンを，お互いにとることができないようにチェス盤の上に置くという問題である．可能な解はいくつか存在する．N-Queen 問題を解くプログラムをリスト 4 に示す．これは Hugs にデモとして付属されている．

check 関数 `check (i,j) (m,n)` は与えられたの 2 つの座標にクイーンを置くことができるかどうかを示す関数

safe 関数 `safe p n` は，`length p` 列までのクイーンの配置が `p` というリストで与えられた時，第 `1 + length p` 列の第 `n` 行にクイーンを置くことができるかどうかを示す関数

queens 関数 1 列目から第 `number_of_queens` 列まで順番に安全な配置を調べる関数

空欄設定のための考察

このプログラムの重要と思われる箇所を考える．このプログラムは 4 つの関数 (`queens`, `safe`, `check`, `q`) で構成されている．`q` 関数は出力のための関数なので重要な関数ではないと考えられる．このプログラムは，1 行目から順番にクイーンを置ける場所に置いていき，それを N 列目まで繰り返すという方法で N-Queen 問題を解いている．「1 列目から N 列目まで順番にクイーンを置いていく」のは `queens` 関数,「次の列の座標にクイーンを置けるかどうかを判定する」のは `safe`，`check` 関数となる．

`queens` 関数の特に重要な部分は `p <- qu m` だと考えられる．なぜならこの再帰によって「1 行目から順番にクイーンを置いていく」という処理を行っているからである．`check` 関数は実際にクイーンが置けるかどうかを座標で比較している関数であり重要である．`safe` 関数はの`(i, j) <- zip [1..] p` は `p` を座標に変換している箇所であり，このプログラムを理解しているかを確認する上での重要度は高くない．10 行目のリスト内包表記の `check (i, j) (m, n)` は，実際にどの座標を判定しているかを確認できるので重要であると考えられる．その中で使用されている `m` は局所的な値であり，これがある値のときだけ `check` は成り立つので，`m` を空欄とすることは有用である．

以上の考察に基づいて手作業で空欄を設定した．7 行目の局所関数 `qu` の定義部で `qu` を再帰的に呼び出している箇所 (`qu m`) と 11 行目の局所関数 `m` の定義部全体 (`1 + length p`)

表 2 手作業と STG 言語を用いた場合の比較
Table 2 Comparison between by-hand choice and STG code length

プログラム	関数名	対象 (手動)	STG の行数	対象 (自動)
P1(Queens)	queens	○	91 行	○
	safe	○	40 行	○
	check		37 行	
	q		2 行	

を空欄とした.

3.3.2 マクロな空欄設定箇所に対する妥当性確認実験

3.1 節の仮説を検証するため，作成した空欄補充問題において対象とした関数と，STG 言語の行数を用いた場合に空欄設定対象となった関数を比較した．一つのプログラムにつき空欄設定の対象とする関数は 2 個とする．例として P1 の比較結果を表 2 に示す．P1 では手作業の場合に対象とした関数と，STG 言語を用いた場合に対象となる関数が完全に一致していることがわかる．表 1 の全てのプログラムを用いて実験したところ，85% (26 個中 22 個) で一致し,「本質的な処理」の近似として妥当であることを確認した.

3.3.3 ミクロな空欄設定箇所に対する妥当性確認実験

空欄補充問題を手作業で作成し，我々の研究室の学生 9 名で解いたところ，ほぼ全ての問題において 10 分から 20 分程度考えて回答することができた．また，その後に口頭でプログラムを理解しているかを確認したところ，理解していることが確認できた．よって空欄設定箇所の妥当性は確認できたと考える.

3.4 空欄設定箇所の優先度

空欄設定箇所に優先度を設定し，その優先度に基づいて空欄とするとよいのではないかという仮説を立てた．例えば図 4 の queens 関数では「再帰」と「リスト内包表記のガード」があるがここでは前者のほうが重要であると考えたため「再帰」の優先度を高いとした．実験結果に基づいて決定した優先度を表 3 に示す．表の上のほうが優先度が高いことを表す.

4. 正誤判定の自動化

空欄補充問題では別解が存在する場合がある．空欄補充問題の正誤判定を自動化するには，文字列としての判定ではなく別解にも対応した正誤判定が必要である.

ここでは単体テストを用いることによって別解へ対応する．問題作成者が各関数に対して単体テストを事前に作成しておき，回答者が空欄を補充したプログラムがその単体テストを通ったら正解とする．空欄設定をする前のプログラムである正答プログラムと，回答者が空欄を補充したプログラムに同一データを入力し，出力が同じ場合単体テストは成功とする.

また，単体テストを通らない回答は不正解であるが，このときどのようなテスト入力に対して正しい出力と異なるのかを問題回答者に提示することで回答者へのフォローとなる.

表 3 空欄設定の優先度
Table 3 Priority of blank setting

カテゴリ	内容
関数本体	再帰
	map 関数, fold 系関数
	分岐に共通して出現する式
局所関数	where で定義された局所関数
	let で定義された局所関数
条件式	if の条件式
	case の条件式
	ガード式
	リスト内包表記のガード

5. 実　　装

5.1 Haskell Automatic Generation system of cloZe questions

Haskell Automatic Generation system of cloZe questions (以下 HAGZ) は，3 章で提案した手法を実装した Web システムである．PHP をモジュールとして組み込んだサーバ上で動作する．ブラウザから HAGZ にアクセスすることで空欄補充問題の作成，回答を行うことが出来る．本システムは空欄生成器と正誤判定器から構成されている.

5.2 空欄生成器

空欄生成器では，入力された Haskell プログラムから「空欄設定済みプログラム」と「答え」の生成を行う．実装は Haskell で行った．空欄設定箇所は 3 章の空欄設定箇所の優先度に基づく．空欄設定箇所の個数はオプションとすることで，難易度の調節を可能とする．入力された Haskell プログラムの解析には，Haskell-src-exts[4] を使用した.

5.3 正誤判定器

正誤判定器では，Haskell のプログラムを自動的にテストするツールである QuickCheck[1] を用いた．プログラマは，プログラムが満たさなくてはならない性質を与えることで仕様を

定義する．QuickCheck はランダムに生成したさまざまなテストケースによって，全ての性質が常に保たれているかをテストする．QuickCheck のプログラムは問題登録者が作成する必要がある．

空欄を補充した結果，無限ループを含むプログラムになる可能性がある．しかし QuickCheck でテストをする際に，無限ループを含むプログラムだと終了しない．時間がある程度経過してもテストが終了しない場合は，QuickCheck のプログラムを強制終了して「誤答」とする．

5.3.1 安全のために使用禁止とするモジュール

単体テストを行う際にプログラムを実行する．悪意のある入力からシステムを守るためにファイルの全削除などの「危険なコードを含むプログラム」は対象外とする必要がある．ファイル操作に関するモジュール，ネットワーク操作に関するモジュール，外部プログラムの実行に関するモジュールを含むプログラムを問題として登録できないようにする．

6. 評　　価

本論文では，Haskell プログラムの空欄補充問題を自動生成することで，プログラミング学習を支援するシステムを実装した．本章では実装したシステムを評価する．

「本システムは Haskell の学習に役立つか」と「本手法による空欄設定箇所は適切か」の 2 点に対する評価を行った．前者は本学の学生を対象に，後者は Haskell プログラミングについて，より専門的な知識を有する人を対象にアンケートを実施した．

6.1 有用性の評価

愛知県立大学情報科学部情報科学科 2 年生 59 名と我々の研究室の学生 6 名を対象に，本システムの試用及び無記名アンケートを行った．情報科学科 2 年生は講義で，研究室の学生はゼミで Haskell プログラミングについて学習済みである．**表 1** の P1–P5，P8，P13，P14 の 8 個のプログラムを用いた．「Haskell プログラミングの学習に役立つと思いますか？」という質問に対するアンケート結果を**表 4** に示す．有用性に関しては好意的な回答が 84.6%を占めた．主観的な評価ではあるが一定の評価を得ているといえる．

6.2 空欄設定箇所の評価

Haskell プログラミングについてより専門的な知識を有する人として，Haskell を用いて仕事・研究を行っている人を対象にシステムの試用及び空欄設定箇所の妥当性に関するアンケートを行った．Twitter を用いて呼びかけを行った．各問題ごとのアンケートに回答した延べ人数を**表 5** に示す．プログラムは 6.1 節と同じものを用いた．空欄設定箇所の妥当性に関する質問として，「空欄として選択された箇所は適切だったと思いますか?」という質問に対するアンケート結果を**表 6** に示す．これは空欄ごとに質問した．好意的な回答が 86.2%を占めており，一定の評価を得ているといえる．

表 4　有用性に関する質問
Table 4　Question concerning utilityg

回答	数	割合 (%)
はい	55	84.6
いいえ	1	1.5
わからない	9	13.9
合計	65	100

表 5　問題ごとのアンケートに回答した延べ人数
Table 5　Number of total answered questionnaire of each problem

番号 (名前)	延べ人数
P1(Queens)	2
P2(IntAtBase)	2
P3(Pajama)	3
P4(Merge)	4
P5(Insert)	5
P8(Invert2)	6
P13(VM)	2
P14(Stack)	1

表 6　空欄設定箇所の妥当性に関する質問
Table 6　Question concerning validity of empty column setting part

回答	数	割合 (%)
そう思う	50	77.0
ややそう思う	6	9.2
ややそう思わない	3	4.6
そう思わない	6	9.2
合計	65	100

6.3 その他の意見

6.3.1 学部 2 年生と研究室の学生からの意見

「本システムに関する意見，要望などを書いてください」という問いに対する代表的な回答を以下に示す．

好意的回答

- 空欄の答えが周囲から推測できる場所だったのでよかったと思う

非好意的回答

- 全体的に難易度が高い。
- 実際に解こうと思っても、ある程度の知識がないと解くことができなかったため、とても基礎的な問題も作って欲しい．

建設的回答

- 片方が正解していたとき，そこだけでも正解と記してほしい．そのほうがモチベーションもあがると思う．
- 問題によっては、Haskell で書く利点や欠点、C 言語でかくとどうなるかなどがあると便利だと思った。
- 答えだけでなく解説ものせてもらえるとわかりやすくなると思います。

考　　察

難易度が高いという回答が多かった．しかし，マージソートや挿入ソートが解けていない学生が大半を占めていた．これ以上に問題のプログラムの難易度を下げるのは困難であるので，ヒントを与える機能などを提供する必要があると思われる．建設的な回答として解説などを充実することで，より学習効果の向上が期待できる可能性があることを指摘している．

6.3.2 専門的な知識を有する人からの意見

Haskell プログラミングについてより専門的な知識を有する人からメールや Twitter を通して意見を頂いた．本節ではその一部を紹介する．

(1) 問題文は英語にしたほうがよい
(2) 「違います」のページはもっと encourage してほしい．あと何回目の挑戦なのかはわかるとうれしい
(3) 例題のコードが汚い．=の右側に if,case,let が不必要に現れるのは美しくない．($) が多いのもいただけない．

6.3.3 考　　察

1 個目の意見は，英語のページと日本語のページを用意することで，今後対応する予定である．2 個目の意見は，学習者の学習意欲を減少させないために重要であるので，対応する必要がある．3 個目の意見は評価で用いたプログラムのコーディングの美しさに関する意見である．コーディングの美しさは，絶対的な部分と好みの部分がある．本研究の目的の一つは，空欄補充問題の自動生成であり，その対象はあらゆるプログラムとなる．ここで指摘されたコーディングの美しさは，判断が難しいところではあるが好みの部分だと考える．

7. 結　　論

本研究では関数型言語 Haskell を対象に，空欄補充問題を自動生成し，また回答に対して答え合わせを自動的に行う手法を提案した．そして本手法に基づいて，学習者が Haskell プログラミングを自習できるシステムを Web アプリケーションとして実装した．

本研究で提案した手法の評価として，本システムを試用した後にアンケートを行った．その結果，システムの有用性と空欄設定箇所の妥当性に関して一定の評価を得ることができた．また，アンケートの記述式回答から，本システムに不足している機能も明確になった．

7.1 今後の課題

アンケートの結果から解説などを提供する必要があることが判明した．間違えた回数によって表示するヒントを多くしていくなどの方法により，回答者のレベルに応じた対応が可能であると考える．また，本研究ではテストケースの生成に QuickCheck を用いる．QuickCheck を用いることでテストケースを人手で作成するコストは軽減されたが，QuickCheck のプログラムを記述する必要がある．自習の際にその作業は障害となるため支援を行う必要がある．

参 考 文 献

1) Claessen, K. and Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs, pp.268–279 (2000).
2) E.Knuth, D.: The Art of Computer Programming Volume 1–4 Third Edition (2004).
3) Ferrante, J., Ottenstein, K.J. and Warren, J.D.: The program dependence graph and its use in optimization, *ACM Trans. Program. Lang. Syst.*, Vol.9, pp.319–349 (1987).
4) haskell-src-exts: Manipulating Haskell source: abstract syntax, lexer, parser, and pretty-printer: `http://hackage.haskell.org/package/haskell-src-exts/`.
5) Jones, S. L.P.: Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine - Version 2.5, *Journal of Functional Programming*, Vol.2, pp.127–202 (1992).
6) Myers, G.J.: Composite/Structured Design (1978).
7) The Glasgow Haskell Compiler: `http://www.teu.ac.jp/media/~earth/FK/`.
8) 柏原昭博，寺井淳裕，豊田順一：いかにプログラム空欄補充問題を作るか?，Vol.99, No.81, 社団法人電子情報通信学会 (1999-05-21).
9) 内田保雄：初級プログラミング学習のための自動作問システム，Vol.2007, No.123, 一般社団法人情報処理学会 (2007-12-07).
10) 有安浩平，池田絵里，岡本辰夫，國島丈生，横田一正：学習者に合わせた C 言語演習穴埋め問題の自動生成 (2009).
11) 林創：再帰呼び出しを含む手続きの処理の難しさ， Vol.6, pp.389–405.