K-007

プログラミングの不理解点を特定できる空欄補充問題の生成
Generating Fill-in-Blank Assignments to Identify Understanding Level of Programming

浅井 創 † 山内 義晴 ‡ 島川 博光 †
So Asai Yoshiharu Yamauchi Hiromitsu Shimakawa

1. はじめに

プログラミングには単に知識だけでなく，ソースコードが書けることが必要である．とくにプログラミングのコードを簡潔に書けることはプログラミングの理解が十分に得られているという点で重要である．簡潔なコードを書くことができる学習者はプログラミングを正しく理解している．反対に，理解せずに簡潔なコードを書かないで次の段階へ進んだ学習者は，今後の学習で躓くおそれがある．そのような学習者の理解度は早い段階で明らかにしなければならない．本論文では，C 言語のプログラミングについて，学習者が知識を正しく理解して簡潔なプログラムを書くことができることを瞬時に判別可能な手法を提案する．判別のために空欄補充問題を用いる．これにより，指導者は学習者の不理解を早期に発見し，個別対応ができる．

2. プログラミングの理解度

2.1 プログラミングの授業における問題点

大学における C 言語プログラミングの授業は毎回異なる知識や技法を学習する．それらの単元はいずれも C 言語の知識として必要不可欠な内容である．それぞれの単元には関連があり，学習の順序も一貫性がある．プログラミングの演習においてソースコードを書くときはこれまで学習したすべての知識を駆使しなければならない．したがって，特定の単元において学習者に不理解があると，以降の単元も必要な理解を得られず，ソースコードを書くことができなくなる．このようなことが連鎖的に起こると，学習者のプログラミングに対する忌避を招く．そのような学習者一人ひとりに対して不理解が生じた段階まで後戻りし，軌道修正を図ることは困難である．指導者は忌避に陥る可能性のある学習者を早期に発見し，学習者が落ちこぼれることを防止する必要がある．

2.2 理解度を判定するための空欄補充問題

大学においては座学で得た知識をプログラミング演習にて，学習者がどの程度プログラムを書くことができるかを確かめている．演習では学習者に問題を読んでもらい，内容をソースコードに落とし込ませる能力を見る．入力に対して指定された出力を生成するプログラムの構成方法は一通りではない．学習者が習得するべき構文を理解していない場合，別の方法で指定された出力を生成できてしまう．しかし，このようなコードを採点する場合，指導者はソースコードの内容を読み通して，学習者の理解できていない箇所を判別しなければならない．これには指導者にとって大変な時間と労力を要する．長い時間が経過すると習得するべき構文を理解していない学生を指導してその構文を理解させるのは難しい．

そこで，別のプログラミング能力の測定手段として空欄補充問題がある．プログラミングにおける空欄補充問題は，ソースコードの文中に空欄を設けて問題とし，プログラムの流れに沿うコードを回答するという形式である．空欄の大きさは任意に決定できるので，空欄の大きさに応じたピンポイントな理解度を測ることが可能である．空欄に入力しうるコードの内容は構造の範疇では有限であり，意図したコードだけを書かせることができる．また，採点は空欄部分の回答の正誤だけを見ればよいので非常に容易である．しかし，一部分だけで意図している事柄について学習者が理解しているかどうかを明らかにしなければならないので，空欄をソースコードのどこに設けるかを決定することは難しい．指導者は理解できている学習者とそうでない学習者を切り分けられる箇所を選ばなければならない．空欄補充問題は，理解のない学生を明らかにできる空欄を設定することが重要になる．

柏原らの研究 [2] では，プログラム依存グラフを用いてソースコードの中から最も理解を問うのに適切な空欄補充問題の生成を行っている．しかし，ソースコードにおける不作法な部分は複数あると考えられるが，この手法では一つのソースコードの中に空欄を一箇所だけしか設けることができない．また有安らの研究 [1] では，プログラムの構文解析による指導者の出題意図に合致する部分の空欄の自動生成を支援している．しかし，この手法では出題意図は指導者自身が決定しなければならず，学習者の理解が判別可能な空欄を生成することはできない．

2.3 簡潔なコードと不作法なコード

指導者が作成する模範コードは，習得するべき構文を無駄なく利用している．指導者が教えた構文を十分に習得しており、演習問題の意図を理解している学習者は，教えられた構文を正しく使い模範を遵守してコードを簡潔に書くことができる．本論文において簡潔なコードであるということは構文の出現度や順序が模範コードに類似することである．反対に，コードを簡潔に書くことができない学習者のソースコードには模範に則っていない部分が存在する．このようなソースコードには指定された出力は生成できるが，奇異な方法で出力が生成されていたり，プログラムとして必要がない余計な処理が含まれていたりするといった不作法なコードが現れる．不作法コードは学習者が構文やデータ構造などのプログラミングを正しく理解できていないことに起因する．本論文においては，空欄補充問題のために使用する演習問題の模範コードを簡潔なコードとし，習得するべき構文を適切に使えていないコードを不作法なコードと表現する．

3. 不理解点を明らかにする空欄補充問題

3.1 過去のコードに基づく空欄の設定

本研究では簡潔なコードと過去の学生が作成した不作法なコードから，現在の学習者がプログラミングの不理解のために不作法なコードを書いているのかを判定できる空欄補充問題を生成する手法を提案する．手法の全体像を図 1 に示す．本手法では，大学の C 言語のプログラミング演習の授業にて，過去と現在の授業で同一の演習問題が課されていると仮定する．学生が作成したソー

† 立命館大学情報理工学部
‡ 立命館大学大学院理工学研究科

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

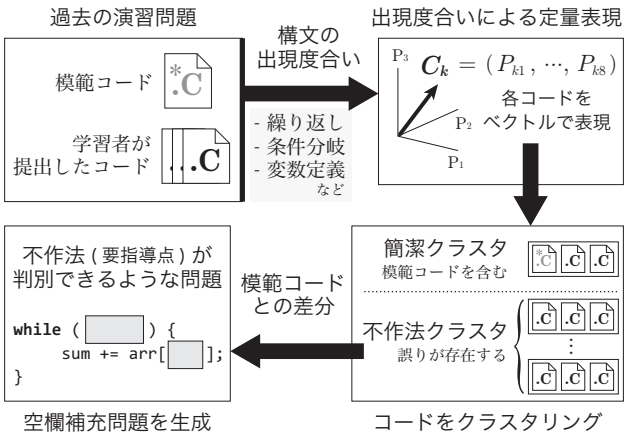

図 1: 手法全体図

スコードを分類するために，ソースコードの中に現れる構文の出現度合いをベクトルで表現した座標空間を用いる．演習問題の模範コードと，過去の学生が答案として提出したソースコードを比較して，答案コードから不作法な部分を同定する．つぎに，不作法な部分について模範コードとの差分をとる．これにより学習者の不理解が判定可能な空欄に設定すべき部分が抽出される．本手法により生成された空欄補充問題は，空欄に付随する部分を現在の学習者が理解しているかを明らかにする．指導者はそのような学習者の要指導点を早期に発見し，現在の学習者の，今後の学習における躓きを防止できる．

3.2 キーワード出現確率によるコードの特徴抽出

模範を遵守して書かれたソースコードと不作法なものを分類するために，各々のソースコードがどのように記述されているかを定量的に表現する．

本手法ではプログラムの構文がソースコードの中にどの程度出現するかで分類する．本研究では，構文を構成するキーワードに着目する．例えば，繰り返しの構文のキーワードには for，while，do が該当する．同じ構文が同程度使われている 2 つのソースコードでは，その構文のキーワードの出現確率は似たものになると考えられる．対象とする構文は，C 言語のプログラムを書くためには理解が必須であり，初期段階のプログラミングの授業においても扱われる，変数，定数，条件分岐，繰り返し，関数，配列，ポインタ，構造体の 8 項目に限定する．ソースコードから構文を抽出するために，ソースコードの形態素解析を行う．形態素解析には MeCab[3] を利用する．通常 MeCab は一般的な文章の解析をするために利用されるので，C 言語の解析には不向きである．よって事前に C 言語固有のキーワードが含まれる辞書を作成する．使われた構文の類似性を求めるために，解析の結果から 8 項目の構文を構成するキーワード数をそれぞれ計上する．ソースコード C_k に現れる構文 S_i の出現確率 P_{ki} は，ソースコード内の単語の数と構文に該当するキーワードの数をそれぞれ $n(S_i)$，$n(C_k)$ とすると次式で求められる．

$$P_{ki} = \frac{n(S_i)}{n(C_k)}$$

ソースコードの特徴を 8 項目の構文の出現確率 P_{ki} を要素に持つベクトル $\boldsymbol{C_k} = (P_{k1}, \cdots, P_{k8})$ で定量的に表現する．指定された出力は生成できるが，奇異な方法や余計な処理が含まれている無作法コードが含まれたプログラムのベクトル $\boldsymbol{C_k}$ は，模範コードのベクトルからは大きく乖離すると考えられる．

3.3 簡潔と不作法の分類

本研究では，同じような不理解点をもつ過去と現在の学習者は，同じような不作法コードを書くと捉える．学習者が作成する不作法コードにどのようなものがあるかを明らかにするために，過去に実施された演習問題から，学習者が書いたソースコードと模範コードを教師なし学習アルゴリズムにより分類する．各ソースコードを構文に基づきベクトル化したソースコードの集合を Ward 法でクラスタリングする．本研究では，不作法コードのクラスタを 8 として，テンドログラムを用いクラスタを区切る．生成されたクラスタについて，模範コードが含まれているひとつのクラスタを簡潔クラスタ，それ以外のクラスタを不作法クラスタとする．簡潔クラスタに分類されたコードは模範コードとの類似度が高く，このようなコードを書いた学習者は習得すべきプログラミング技法を理解したうえでコードを書いたと推定できる．一方，不作法クラスタに分類されたコードには，不作法とされる部分が存在する．不作法クラスタに分類されるコードを書いた学習者には特定の不理解が伴っている．その不理解の原因を推定して，空欄補充問題を生成する．

3.4 差分による空欄の生成

不作法クラスタに属する学習者にどのような不理解点があるかを判別できるような空欄を生成する．本手法においては，不作法クラスタから一つソースコードを選択して，模範コードとの差分をとり，一致しない部分を空欄に設定する．簡潔なコードであるためには模範コードに近づかなければならず，差分による不一致な箇所は，学習者が正しい記述を理解したうえで改めなければならない部分である．差を考えるうえで，3.2 節で述べたプログラムのベクトル表現を用いる．不作法のソースコードと模範コードの差をとり，ベクトルの構文の各要素間でもっとも差が大きい構文を空欄に設定する．例えば，繰り返しの要素に大きく差がみられたときは模範コードの中の繰り返しの箇所を空欄に設定する．このようにすべての不作法のクラスタについて，差分を取り模範コード中のそれぞれの場所に空欄を設け，各空欄の正誤で何が理解できていないのかが判る空欄補充問題を生成する．

4. おわりに

本論文では過去に実施された演習問題のソースコードから不作法コードを分類することで，学習者が抱く不理解を判定できる空欄補充問題を生成する手法を提案した．今後は，本手法の有用性を実験により検証する．

参考文献

[1] 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，DEIM Forum (2009).

[2] 柏原 昭博，久米井 邦貴，梅野 浩司，豊田 順一： プログラム空欄補充問題の作成とその評価，人工知能学会論文誌，16 巻 4 号 C (2001).

[3] Taku Kudo, Kaoru Yamamoto, Yuji Matsumoto: Applying Conditional Random Fields to Japanese Morphological Analysis, Proceedings of EMNLP, pp.230 – 237 (2004).

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.