プログラミング演習における解答の構文に基づく自動判別手法

情報科学科　山田 誠也　　指導教員：山本 晋一郎

1　はじめに

高度なプログラミング能力を持つ技術者を養成するために，多くの情報系の大学ではプログラミング演習を実施している．本学でもプログラミング入門として，純粋関数型言語 Haskell を利用した演習を実施している．

学生が基本的な構文を学習する段階では，プログラムの動作だけではなく，学習した特定の構文や組み込み関数を利用していることを演習問題の採点基準に盛り込む必要がある．

答案プログラムの動作を，ソフトウェアテストの観点で自動的に確認する方法は確立され，実際に本学で運用されている．しかし，構文的な条件という観点での自動的な判定方法はまだない．

本論文では，答案プログラムの構文的な条件を自動的に判定するために，構文解析を利用することで，プログラムが利用している構文や関数を判別する手法の提案を行う．

2　構文解析によるプログラムの判別

プログラムは，同じ結果を返すものでも実装方法が複数存在する．本手法では，プログラムの情報を構文解析を利用して取得し，その結果に基づいて判別を行う．

2.1　構文解析

本論文では，Haskell プログラムの構文解析を行う際に，haskell-src-exts[1] を利用する．構文解析の例として，以下のプログラム断片を解析した結果を示す．

プログラム断片

```
if not (x `elem` xs) then x else y
```

解析結果

```
(If                                                    — if
  (App (Var (UnQual (Ident "not")))                    — not
    (Paren                                             — ()
      (InfixApp (Var (UnQual (Ident "x"))) (QVarOp (UnQual (Ident "elem")))  — x elem
        (Var (UnQual (Ident "xs"))))))                 — xs
    (Var (UnQual (Ident "x")))                         — then x
  (Var (UnQual (Ident "y"))))                          — else y
```

特定の構文や関数を利用しているプログラムの解析結果には，必ず対応するキーワードとなる値コンストラクタが存在する．上記の解析結果では，if 式に対応する値コンストラクタ If が解析結果に含まれていることが確認できる．本手法ではこのキーワードが解析結果に含まれているか判定することで，プログラムの実装方法を確認する．

2.2　判別項目

今年度の本学の演習では演習問題を 106 問出題している．そのうち，構文上の指定をした問題は 31 問であり，指定された条件を分類すると 9 種類におよぶ． 表 1 に示すように，本手法では解析結果にキーワードが存在する構文の判別は全て可能である．where は対応するキーワードが存在しないが，where で局所定義を行った情報を格納した箇所を検出することで判別を行う．関数定義の仮引数に対するパターンマッチを指定した問題は最も多く 9 問出題されていた．パターンマッチを用いたプログラムの解析結果は，1 パターン毎にパターンマッチの情報を格納するためのリストの要素として返される．このため，このリストが空リストでないことを確認することで，パターンマッチを利用したプログラムだと判別できる．

表 1　構文上の条件の分類

条件	問題数	判別	キーワード
ラムダ式	1	可	lambda
セクション	2	可	RightSection
リスト内包表記	2	可	ListComp
let	2	可	Let
case	2	可	Case
where	2	可	-
再帰	4	可	Ident 関数名
組み込み関数	7	可	Ident 関数名
パターンマッチ	9	可	-

3　評価

評価にあたり，前節の提案手法を用いた答案プログラムの自動採点システムを試作した．プログラムの動作の確認は Haskell プログラムのランダムテストツール QuickCheck[2] を用いて行う．今回は評価のために，以下の演習問題と解答例を用意した．

演習問題

リスト内包表記と length を利用し，リスト xs と要素 w を与えたとき，xs の中にある w と等しい要素の数を返す関数を作成せよ．

解答例

```
func1 xs w = length [() | x <- xs, x == w]
func2 xs w = length $ filter (== w) xs
func3 xs w = sum [1 | x <- xs, x == w]
```

- func1　リスト内包表記と length を利用できている．
- func2　リスト内包表記を利用できていない．
- func3　length を利用できていない．

3 つの解答例は全て問題の仕様を満たしている．これらの解答例を試作したシステムで採点した結果を表 2 に示す．

表 2　自動採点結果

採点基準	配点	func1	func2	func3
動作	50	○	○	○
リスト内包表記	30	○	×	○
関数 length	20	○	○	×
合計点	100	100	70	80

表 2 の結果より，答案プログラムの動作だけでなく，構文的な条件を自動的に判定できるようになったといえる．

4　おわりに

本論文では，Haskell プログラムの構文解析を行い，プログラムの利用している構文や関数の判別を行う手法を提案した．この結果，答案プログラムの動作だけでなく，構文的な条件を自動的に判定できるようになった．

今後の課題としては，プログラミング演習で運用するためのシステムの構築と，判別項目の追加を考えている．判別項目を増やすことで，よりきめ細かい構文的な条件を演習問題に指定できるようになるため，細部まで学生の知識の定着度が確認できるようになると考えられる．

参考文献

[1] The haskell-src-exts package.
`https://hackage.haskell.org/package/haskell-src-exts-1.13.5`

[2] Koen Claessen and John Hughes, “QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs” ,ICFP, pp.268–279 ACM, 2000.