2K-1

Python から Ruby へとプログラムの自動変換を図るシステムの構築

清水 崇之　　小川 翔二郎　　松原 俊一　　Martin J. Dürst
青山学院大学理工学部情報テクノロジー学科

1　はじめに

今日，多くのプログラミング言語が使われており，多くのライブラリがそれぞれに作られている．だが，それらはユーザーが使いたい言語で利用できるとは限らない．使いたい言語へ変換することもできるが，手動での変換は非常に手間がかかる．類似点が多い Python [1] と Ruby [2] でさえ，その手間は少なくない．その点に着目し，Tomson は Python から Ruby への自動変換プログラム translate_Python_to_Ruby [3] を開発した．しかし，正規表現を主体としているため，変換範囲や拡張性の点で十分とは言えない．そこで本研究では，Python から Ruby への変換をより効率的かつ広範囲で行うシステムを構築する．制御構造のブロック書式などの基本的な部分から，関数名やリスト内包表記などの特徴的な部分も対応する．また，対応できない部分には行末にコメントを挿入することでユーザーに警告している．これにより，ユーザーはより少ない負担でプログラムを変換できる．

2　システム構成

2.1　原始言語と目的言語

本システムを使用する上で，原始言語は正常動作する Python とする．また，目的言語は Ruby である．ただし目的言語には警告としてのコメントが挿入されているため，ユーザーはその部分や関連する部分への対応が必要になる．

2.2　変換の流れ

本システムでは，Python を二段階で構文解析する．Python のブロックは，インデントを用いて定義されている．そのため，第一段階でインデントのみを対象として構文解析する．第二段階で Python の文法 [4] に沿った構文解析を行う．基本的な構造は構文木に沿って変換する．

A System Attempting Automatic Program Translation from Python to Ruby
Takayuki Shimizu, Shojiro Ogawa, Shunichi Matsubara and Martin J. Dürst
Department of Integrated Information Technology, College of Science and Engineering, Aoyama Gakuin University
5–10–1 Fuchinobe, Chuo-ku, Sagamihara, Kanagawa 252-0206, Japan
{shimizu,shojiro}@sw.it.aoyama.ac.jp,
{matsubara, duerst}@it.aoyama.ac.jp

Python の構文解析器は生成系 Treetop [5] で生成される．Treetop は構文解析に字句解析を含む．これを利用し，空白を保持したまま変換する．以下に，様々な構文を含んだコードについて変換を行った例を記述する．図 1 が元の Python プログラムであり，図 2 が変換後の Ruby プログラムである．

```
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
import sys
if num:
    print("true")
elif num == False:
    print([x*y for x in vec1 for y in vec2])
else:
    disp = sys.stdout.write
    disp(num)
```

図 1: Python プログラム

```
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
require 'sys'
if num
    puts "true"
elsif num == false
    puts vec1.product(vec2).collect{|x,y| x*y}
else
    disp = sys.stdout.write #変数へ関数を挿入
    disp(num) #不明な関数
end
```

図 2: 変換後の Ruby プログラム

3　ライブラリの関数名の変換

Python と Ruby では，関数機能に差異が存在する．同名の関数が違う機能を有する場合や，別名の関数が

同じ機能を有している場合がある．さらに，場合によってはメソッドに置き換える必要がある．

本研究では標準ライブラリの関数名の変換に対応するため専用の Ruby 内部ドメイン特化言語を作り，その処理系を組み込んでいる．ドメイン特化言語は，各要素を図 3 のように記述する．

```
1 py_module (:built_in) {
2   py_def (:print) {
3     r_name :puts
4     delete_argument :end
5     delete_parentheses false
6     type :func
7   }
8 }
```

図 3: 専用ドメイン特化言語の記述例

図 3 中の `py_module` と `py_def` は，Python でのモジュール名と関数名を示している．これにより，個別の関数についての設定ができる．さらに，コード中の識別子がモジュールか，または変換可能な関数名であることを判別できる．図 3 の 3 行目から 6 行目は Ruby について記述している．変更後の関数名は `r_name` で指定し，関数内に削除したい引数の有無は `delete_argument` で指定する．`delete_parentheses` は関数をメソッドへ変換する際などの括弧の削除・追加の指定に使用する．識別子の種類は `type` で指定する．これにより出力の際に識別子の位置が変更される．これらの指定は処理系によって自動的に処理される．

標準ライブラリについては以上のような仕様で用意するため，ユーザーが記述を行う必要はない．自作ライブラリについて特に必要だと感じた際，図 3 のように記述することで拡張できる．

4 リスト内包表記

Python では，リスト内包表記が用意されている．これは，”[”と”]”の間に一つ以上の `for` 節と，0 以上の `if` 節によって記述される [4]．これを Ruby に変換する場合，イテレータへ置き換えることで表現できる．本システムでは，3 つのパターンについて対応することで，変換を実現している．まず，`for` 節が一つの場合はメソッド `collect` へ置き換える．`for` 節が複数の場合は `product` で組み合わせる．また，`if` 節を含む場合は `select` を用いる．これにより，全てのリスト内包表記へ対応できる．表 1，2 に変換の例を示す．

また，リスト内包表記をネストさせることもできるが，同様の方法で対応できる．

リスト内包表記の例
`[3*x for x in vec]`
`[x*y for x in vec1 for y in vec2]`
`[3*x for x in vec if x>3]`

表 1: 変換前のリスト内包表記

変換の例
`vec.collect{ |x| 3*x }`
`vec1.product(vec2).collect{ |x,y| x*y }`
`vec.select{ |x| x>3 }.collect{ |x| 3*x }`

表 2: 変換後のイテレータ

5 まとめ

本システムは，Python の文法規則に則った構文木を作ることで，多様なプログラムの解析を行い，Python から Ruby への自動変換を可能としている．これにより，ユーザーは手動に比べ数倍少ない手間で変換を行うことが可能となった．

今後の課題としては，`built_in` や `sys` などのモジュールにより対応するためのライブラリの整理や，現在のドメイン特化言語で対応しきれない関数への適応，タプルへの対応などが挙げられる．これらに対応していくことで，より多くのプログラムが容易に変換できるようになり，利便性が向上していくと考えられる．

参考文献

[1] David Ascher Mark Lutz. 初めての Python. オライリージャパン, 11 月 2004.

[2] Dave Thomas with Chad Fowler and Andy Hunt. プログラミング Ruby 1.9 言語編. オーム社, 5 月 2010.

[3] Phil Tomson. translate_python_to_ruby. http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/71961/, May 21 2003.

[4] Python Software Foundation. Python Programming Language - Official Website - Full Grammar specification. http://docs.python.org/py3k/index.html/, Dec 02 2010.

[5] Pivotak Labs. Treetop. http://treetop.rubyforge.org/index.html/, Dec 21 2010.