7J-08

Python 標準ライブラリ関数の Ruby への自動変換

多田 悠太朗　　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

筆者らの所属研究室では Python プログラムから Ruby プログラムへ自動変換するシステムを開発している．両言語は独立して開発されたため，完全な自動化は困難である．そのため，できる限り手間の少ない変換を目指している．より具体的には，Python のライブラリを Ruby で使用可能にすることを目的としている．2010 年に当システムの開発を開始し，基本構文の解析，条件式の対応，デコレータの変換に取り組んできた [1].

本研究では，Python の標準ライブラリ関数 [2] に着目してその変換機能を追加する．関数の変換パターンを分析し実装する．また，開発の負担を減らすために CPython のテストを利用する手法を提案し実践する．標準ライブラリ関数を自動変換することにより，ユーザの変換の手間を大幅に削減できる．

2　標準ライブラリ関数の変換

Python の標準ライブラリ関数を手作業で Ruby に変換するには，その関数の仕様を調査し，それに対応したメソッドに書き換える作業が必要になる．そこで本研究では，そのような作業の負担を軽減するために以下の方法を提案し実装した．

2.1　標準ライブラリ関数の変換方法の提案

Python の組込み関数およびユニットテストライブラリの主要な関数を Ruby のメソッドに対応付けた．その結果をもとに各関数の変換パターンを分析し，パターン項目を抽出した．各パターンごとの変換動作を定義し，各関数のパターン項目を内部ドメイン特化言語で記述することで実装した．変換は，Python の文法規則を適用した構文解析器，内部ドメイン特化言語，出力部で構成される．

2.2　内部ドメイン特化言語

当システムには，Python の関数を変換するための内部ドメイン特化言語が，清水らの研究 [3] によって一部実装された．このドメイン特化言語は，Python の関数ごとに，その変換パターンについての情報を記述する．例えば，変換後のメソッド名や括弧を削除するかどうかなどである．本研究では，このドメイン特化言語を大幅に拡張することで標準ライブラリ関数の変換を実装した．

2.3　変換パターン項目

本研究で実装した変換パターン項目について説明する．これらの項目のほかに，メソッド名，キーワード引数の削除，括弧の削除に関する項目がある．

変換後のタイプ　関数，メソッド，演算子のいずれかを指定する．関数であれば識別子の後ろに引数を出力し，メソッドであればレシーバ，ピリオド，識別子の順に出力する．演算子の場合は演算子を用いた式の形で出力する．

引数の順序変更・削除・追加　引数の順序変更，削除及び追加をする場合に記述する．例えばユニットテストの場合，Python の `assertEqual()` と Ruby の `assert_equal` は同じ機能であるが，第 1 引数と第 2 引数を入れ替える必要がある．

ライブラリの追加　追加するライブラリがあれば，ライブラリの重複がないかを確認し，ファイルの先頭に `require` 文を出力する．例えば，Python では標準で集合型を扱えるが，Ruby ではライブラリをロードする必要がある．

複数のメソッドの組合せ　複数のメソッドを組み合わせることで対応する場合に記述する．例えば，Python の `frozenset()` は Ruby の集合クラスのコンストラクタと `freeze` メソッドを組み合わせることで実装する．

2.4　変換の例

Python の組込み関数の変換パターンを裏面の表 1 に示す．

3　開発の手法

開発を効率よく進めることと，より正確にテストするために以下の方法を提案する．テストを変換するために，Python のユニットテストライブラリ関数から変換する．そして，CPython のテスト[1]を当システム

Automatic Translation of the Python Standard Library Functions to Ruby
Yuutarou Tada, 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 252-5258, Japan
duerst@it.aoyama.ac.jp

[1] cpython/Lib/test at master · python/cpython · GitHub, https://github.com/python/cpython/blob/master/Lib/test

Copyright ©2018 Information Processing Society of Japan.
All Rights Reserved.

表 1: Python 関数 bin() の変換の例

bin(10) ⇒ sprintf("%#b", 10)	
変換パターン項目	項目の内容
メソッド名	sprintf
キーワード引数の削除	なし
括弧の削除	なし
変化後のタイプ	関数
引数の順序変更	第 1 引数 ⇒ 第 2 引数
引数の削除・追加	第 1 引数に "%#b" を追加
ライブラリの追加	なし
複数のメソッドの組合せ	なし

で変換する．この変換による利点は二つある．一つ目は，変換のテストができる点である．二つ目は，変換後のプログラムを実行できるように修正し，それを実行することでテストができる点である．

ソースコード 1 は CPython のテストから abs() 関数のテストを一部抜粋したものである．これを当システムで変換した結果がソースコード 2 である．ユーザによる手作業での修正を必要とする部分には警告としてコメントが挿入される．assertEqual() などが変換されていることが確認できる．これを正しく実行できるように修正したものがソースコード 3 である．

ソースコード 1: Python の abs() 関数のテスト

```
1  import unittest
2
3  class BuiltinTest(unittest.TestCase):
4     def test_abs(self):
5        # int
6        self.assertEqual(abs(-1234), 1234)
7        # float
8        self.assertEqual(abs(-3.14), 3.14)
9        # str
10       self.assertRaises(TypeError, abs, 'a')
11       # bool
12       self.assertEqual(abs(True), 1)
```

ソースコード 2: 変換後のコード

```
1  require 'test/unit'
2  require './py_lib'
3
4  class BuiltinTest < .TestCase # possibility
       of unknown method # possibility of
       function assignment
5     def test_abs()
6        # int
7        assert_equal(1234, (-1234).abs)
8        # float
9        assert_equal(3.14, (-3.14).abs)
10       # str
11       assert_raise(TypeError, abs, 'a')
12       # bool
13       assert_equal(1, (true).abs)
14
15    end
16
17 end
```

ソースコード 3: 修正後のコード

```
1  require 'test/unit'
2  require './py_lib'
3
4  class BuiltinTest < Test::Unit::TestCase
5     def test_abs()
6        # int
7        assert_equal(1234, (-1234).abs)
8        # float
9        assert_equal(3.14, (-3.14).abs)
10       # str
11       assert_raise(NoMethodError) do
12          'a'.abs
13       end
14       # bool
15       # assert_equal(1, (true).abs)
16
17    end
18
19 end
```

4 評価

本研究で組込み関数は 68 個中 51 個，文字列メソッドは 44 個中 19 個が変換可能になった．ユニットテストライブラリ関数は 73 個の内，主要な関数 4 個を Ruby に対応付けた．複数のメソッドの組合せに変化するパターンは現在未対応である．その他の対応できていない関数の例として，memoryview() などの Ruby への対応付けが困難な関数や引数の数が不定の関数などが挙げられる．

ソースコード 1 とソースコード 3 を比較すると，関数の変換や self の削除などの主な変更点は 26 箇所ある．そのうち 22 箇所は当システムで自動で変換されるため，ユーザが手作業で修正する必要はない．例外やブロック引数などは手作業による修正が必要である．

5 まとめ

本研究では，Python の標準ライブラリ関数の変換パターンを分析し，変換パターン項目を抽出することで，内部ドメイン特化言語を拡張した．また，CPython のテストを利用し，開発を効率よく進める手法を提案し実践した．これにより，標準ライブラリ関数の変換に対応する作業の負担を減らすことができる．

今後の課題として，Python の 関数機能に対応する Ruby メソッドがブロック引数を取る場合や，組込み例外やクラスの入れ子の変換などが挙げられる．これにより，今後さらに多くの Python プログラムの変換が可能になると考えられる．

参考文献

[1] 中山竜一, 松原俊一, Martin J. Dürst. デコレータの Python から Ruby への自動変換. 第 77 回全国大会講演論文集, No. 1, pp. 301–302, 2015.

[2] Python 標準ライブラリ - Python 3.6.3 ドキュメント. https://docs.python.jp/3/library/index.html.

[3] 清水崇之, 小川翔二郎, 松原俊一, Martin J. Dürst. 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. 第 73 回情報処理学会全国大会論文集, 2011.

Copyright ©2018 Information Processing Society of Japan.
All Rights Reserved.