初学者の学習支援のための Python と Ruby 間の翻訳

2014SE020 廣瀬 隼大　2014SE003 赤羽 里帆
2011SE286 渡邊 将匡
指導教員：横山 哲郎

1　はじめに

1.1　背景

今日では，第 4 次産業革命や，プログラミング言語の義務教育化政策により，社会にプログラミング言語が多く普及し，かつてよりも，プログラミング言語が一般の人に認知されてきている．そこで今回我々は，プログラミング言語の初学者のための言語を翻訳する翻訳ソフトウェアを作成する．その中で，現在プログラミング言語の中で最も人気のある Python[1]と，需要が高く，軽量スクリプト言語の中で最もメジャーな言語である Ruby を，翻訳する言語として採用した．また，Python と Ruby は基本的な概念や使用用途において非常に類似している．しかし，様々な差異が存在するため[2]，そのようなソースコードを翻訳し，比較することで，初学者の効率的な学習に繋がると考えた．ここでの初学者とは，Python を既に学習しており，新しく Ruby を学習する者とする．

1.2　課題

我々は，新しくプログラミング言語を学習するにあたって，多くのコストがかかる．そこで私たちは，初学者の学習にかかるコストを削減する為，コストの中にある学習時間の短縮が期待できる翻訳ソフトウェアを作成する．翻訳の精度をあげ，内訳の分からないプログラムも翻訳できるように，抽象構文木を使用して翻訳システムを作成する．

1.3　アプローチ

Python の既存の標準ライブラリを使用して，Python のソースコードを抽象構文木に変換を行う．変換された Python の抽象構文木を，Ruby の抽象構文木へ翻訳し，プリティプリンタを使用して，Ruby のソースコードへと翻訳を行う．また，我々の翻訳の範囲は，南山大学のプログラミング基礎の授業の範囲である．

1.4　期待される効果

翻訳ソフトウェアで自動的に言語を翻訳することによって，ユーザーが手動で言語を翻訳するよりも，時間的コストを大幅に削減できる．それにより，ユーザーにかかるの労力についても，自動的に言語が翻訳されるので，負担を大きく減らすことが可能となる．ソースコードの質については，手動で翻訳するとユーザーの能力や経験によって，質の差異が生まれるが，自動的に翻訳することにより，ソースコードの質が一定に保たれる．そこで学習面において，事前に動きがわかっているソースコードを翻訳し，翻訳前と翻訳後のソースコードを比較することによって，初学者がソースコードの中身を理解しやすくなることが期待される．また，Python は WEB 系の言語であり，ネット上に多くのソースコードが存在するため，Ruby のソースコードがない場合，自動翻訳を使用することにより，初学者にとって効率的な学習に繋げることが出来る．

2　関連ソフトウェア

2.1　類似研究

本研究の翻訳ソフトウェアと類似ソフトウェアとの比較を表した表は以下の表 1 の通りである．

表のコンパイラ[3]は様々な開発環境で使用されている場合も含む．

2.2　関連技術

2.2.1　構文

構文は，規則的な字句の並びである [3]．例えば if 文では if 条件式 ‘:’ のように続く．

ソースコードの記述をする時に構文に沿った記述をしないと，ソースコードが実行されない．

2.2.2　BNF 記法

BNF 記法はバッカス・ナウア記法の略称であり，構文を定義するメタ言語である[3]．そのため，コンピュータへの命令を行う言語の定義にも使用されている．

BNF 記法には表現力において問題があり，その問題を解決するために拡張 BNF 記法が作成された．

Python と Ruby は拡張 BNF 記法によって構文が定義されており，その定義から解析木が生成される．そのため，本研究では拡張 BNF 記法を使用する．

2.2.3　字句解析

字句解析はソースコードを最小単位の字句に分割する解析である[3]．字句解析の入力は文字列であり，出力は字句である．

また，字句解析を実行するプログラムを字句解析器と呼び，字句解析器で字句解析が実行される．

字句解析器は文字を読み取る文字読み取りと，読み込んだ文字を字句に分割する字句読み取りで構成されている．

2.2.4　構文解析

構文解析は字句からソースコードの構成を明らかにする解析である[3]．構文解析の入力は字句であり，出力は

表 1 関連ソフトウェアの比較

	対象言語	方法	使用言語
青山学院大学[2]	PythonをRubyに翻訳する.	外部のソフトウェアを利用して構文解析し, 得られた中間語を翻訳する.	不明
コンパイラ[3]	高級プログラミング言語を低級プログラミング言語に翻訳する.	字句解析を行った後,構文解析を行い, それによって得られた中間語から目的コードを生成し, 目的コードを出力する.	複数
mysystem[4]	C言語をJavaに翻訳する.	他のソフトウェアで中間語を生成し, その中間語をJavaのソースコードに翻訳	不明
本稿の翻訳ソフトウェア	PythonをRubyに翻訳する.	標準ライブラリを使用し,それによって生成される抽象構文木を翻訳する.	Python

解析木である. プログラムの意味上で不要な字句を除いた解析木を抽象構文木と呼ぶ.

また, 構文解析を実行するプログラムを構文解析器と呼ぶ.

2.2.5 LR 構文解析法

LR 構文解析法は構文解析器が文における一つの構造を読み終わった時, BNF 記法で与えられた生成文法によって, ソースコードのトークン列を還元しながら, 最終的に閉路とならずに一つの生成規則になるような還元の流れを描く解析木を作る解析方法である.

また, Python と Ruby の構文解析器では, LR 構文解析法の一種である LALR 構文解析法が使用されている. それらの構文解析器等を用いて, 抽象構文木を生成する標準ライブラリが Python と Ruby で提供されている. その標準ライブラリは Python では ast モジュール, Ruby では Ripper クラスである.

3 製作部分

中田育男の方法[3]を参考とし, 本研究において我々がどの部分に携わるのかを以下の図 1 に記述する.

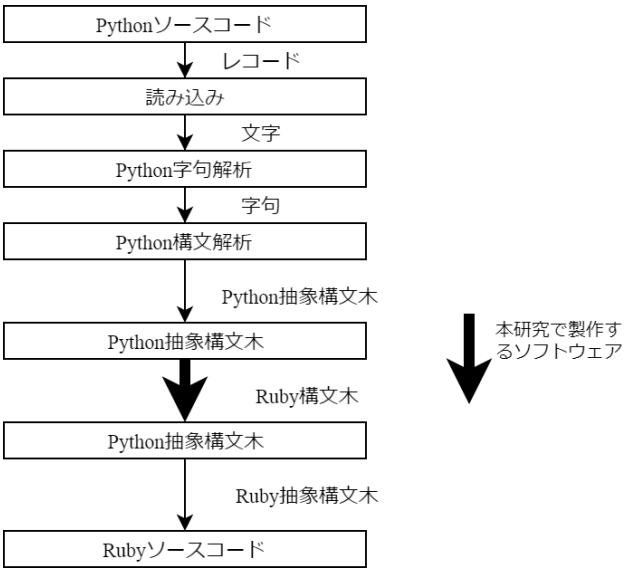

図 1 本研究で製作する部分

4 翻訳ソフトウェアについて

図 1 から本研究においての翻訳ソフトウェアの役割は Python の抽象構文木を Ruby の抽象構文木に翻訳することである.

なぜならば, Python と Ruby の抽象構文木は, それぞれ Python と Ruby の標準ライブラリを使用して作成されるからである. そのため, 我々が製作する翻訳ソフトウェアの役割は上記の通りであることが分かる.

5 翻訳規則の設計

本研究では, Python と Ruby の構文の中でも四則演算と if 文に着目し, Python から Ruby への翻訳規則を構築する.

標準ライブラリで生成された抽象構文木を使用し, それを使用した翻訳をするため, 以下の順で翻訳規則を構築する.

(a) Python と Ruby のソースコードを比較する.
(b) 同じ意味のソースコードを抽出する.
(c) 同じ意味のソースコードが複数ある場合, 意味として妥当なものを翻訳先のソースコードとする.
(d) それぞれの抽象構文木を比較する.
(e) 一般化する.

この方法により, Python から Ruby への四則演算と if 文における翻訳規則が出来ることが期待される.

5.1 Python と Ruby の拡張 BNF 記法

ここでは, Python の構文について拡張 BNF 記法で記述する. Ruby については終端記号のみを記述する.

5.1.1 Python の if 文と四則演算の構文

Python3.6.1 ドキュメント[5]から, 四則演算と if 文を表す構文は

```
if_stmt: 'if' test ':' suite ('elif' test ':' suite)* ['else' ':' suite]
test: or_test ['if' or_test 'else' test]
or_test: and_test ('or' and_test)*
and_test: not_test ('and' not_test)*
not_test: 'not' not_test | comparison
comparison: expr (comp_op expr)*
comp_op: '<'|'>'|'=='|'>='|'<='|'!='|'is'|'is' 'not'
and_expr: shift_expr ('&' shift_expr)*
```

shift_expr: arith_expr (('<<'|'>>') arith_expr)*
arith_expr: term (('+'|'-') term)*
term: factor (('*'|"/"|'//'|'%') factor)*
factor: ('+'|'-') factor
suite: NEWLINE INDENT stmt+ DEDENT
stmt: if_stmt | simple_stmt
simple_stmt: test (';'test)* [';'] NEWLINE
simple_stmt: small_stmt (';' small_stmt)* [';'] NEWLINE
small_stmt: (expr_stmt)
expr_stmt: testlist_star_expr (annassign | augassign (testlist))
である.

5.1.2 Ruby の終端記号の集合

Ruby リファレンスマニュアル 2.4.0[6]から, 本研究での翻訳に必要な終端記号の集合は
Ruby ={'if','elsif','else','end','numeric',
'words','and','or','not',',','+','-',
'*','/','>','>=','<','<=','==','!=','%'}
である.

5.2 翻訳解説

ここでは 5.1.1 節の構文と 5.1.2 節の終端記号の集合を元に Python から Ruby への四則演算と if 文のソースコードの翻訳例から, Python と Ruby の四則演算と if 文の違いを解説する.

5.2.1 翻訳例

Python から Ruby への四則演算と if 文の翻訳例は以下の図 2 となる.

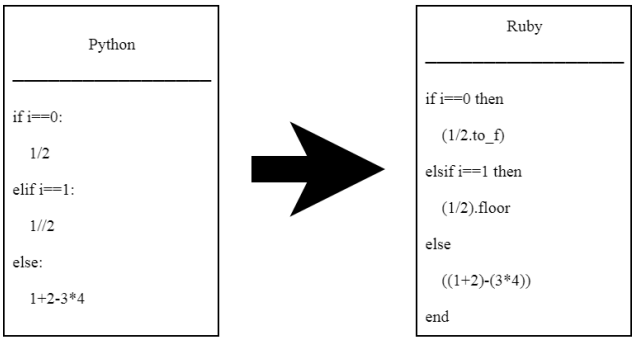

図 2 翻訳例

5.2.2 翻訳例のソースコードの違いについて

ここでは 4.2.1 節で記述した各ソースコードの違いを四則演算と if 文に分けて記述する.
(I)四則演算の違い
(a)式における括弧の有無の違い
Python の ast モジュールで生成した抽象構文木に括弧は含まない. しかし, Ruby の Ripper で生成した, Ruby の抽象構文木を表す S 式は括弧を含んでいる. そのため Ruby の抽象構文木から Ruby ソースコードへの翻訳において, 括弧を含まない状態で翻訳をすると, Ruby の抽象構文木としては正しいかも知れないが, Rubyソースコードの意味が変わる可能性がある.

それを防ぐために, Ruby ソースコード内で式に括弧を付けている.
(b)割り算の違い
Python では除算の演算子が二通りある. /演算子は, 数値を**浮動小数点**型に変換してから計算を行う. //演算子は, 商の小数点以下を切り捨てる除算を行う. そのため, /演算子の意味は, to_f メソッドで浮動小数点型に変換してから計算を行うことにより, 保存される. //演算子の意味は, floor メソッドで商の小数点以下の切り捨てを行うことにより, 保存される.
(II)If 文の違い
Python の if 文では, 終端に空行を挿入することにより, 実行が終了したことを表す.
Ruby の if 文では, 終端に end を記述することにより, 実行が終了したことを表す.

5.3 一般化

5.2.1 節の例と 5.2.2 節を元に Python から Ruby へと翻訳する四則演算と if 文の規則を一般化すると, 以下の通りとなる.

$$T\left[\left['\text{if}' test\ ':'\ suite(\text{`elif'}\ test\ suite) * [\text{`else'}\ ':'\ suite]\right]\right]^{Python} =$$
$$(\text{'if'} T\left[[test]\right]^{Python}\ then\ T\left[[suite]\right]^{Python}$$
$$(\text{'elsif'}\ T\left[[test]\right]^{Python}\ then\ ) *$$
$$[\ \text{'else'}\ T\left[[suite]\right]^{Python}]\ \text{'end'})$$
$$T\left[[\text{NEWLINE INDENT}\ stmt+\ \text{DEDENT}]\right]^{Python} = (T\left[[stmt]\right]^{Python})$$
$$T\left[[test\ (';'\ test) * [';'\ ]\ NEWLINE\right]^{Python} = T\left[[test]\right]^{Python}(T\left[[test]\right]^{Python}) *$$
$$T\left[[or\_test\ [\text{'if'} or\_test\ \text{'else'}\ test]]\right]^{Python} = T\left[[or_{test}]\right]^{Python}\text{'if'} T\left[[or_{test}]\right]^{Python}$$
$$\text{'else'}\ T\left[[test]\right]^{Python}$$
$$T\left[[and\_test\ (\text{'or}\ '\ and\_test) *]\right]^{Python} =$$
$$T\left[[and\_test]\right]^{Python}(\ '\ \text{or}'\ T\left[[and\_test]\right]^{Python}) *$$
$$T\left[[not\_test\ (\text{'and}\ '\ not\_test) *]\right]^{Python} = T\left[[not\_test]\right]^{Python}(\ '\ \text{and}'\ T\left[[not\_test]\right]^{Python}) *$$
$$T\left[[\text{'not}\ '\ not\_test *]\right]^{Python} =$$
$$(\ '\ \text{not}'\ T\left[[not\_test]\right]^{Python}) *$$
$$T\left[[\ expr\ (comp\_op\ expr) *]\right]^{Python} = T\left[[expr]\right]^{Python}(\ T\left[[comp\_op]\right]^{Python}\ T\left[[expr]\right]^{Python}) *$$
$$T\left[[\ term\ ((\text{'+'}\ |\text{'}-\text{'})term) *]\right]^{Python} = T\left[[term]\right]^{Python}(\ (\text{'+'}|\ '-\text{'})\ T\left[[term]\right]^{Python}) *$$
$$T\left[[\ factor\ ((\text{'}*\text{'}\ |\text{'}\%\text{'})factor) *]\right]^{Python} = T\left[[factor]\right]^{Python}(\ (\text{'}*\text{'}|\ \text{'}\%\text{'})\ T\left[[factor]\right]^{Python}) *$$
$$T\left[[\ factor\ (\text{'}/\text{'}factor) *]\right]^{Python} = T\left[[factor]\right]^{Python}(\ \text{'}/\text{'}T\left[[factor]\right]^{Python}, \text{to\_f}) *$$

$$T\left[\left[\, factor\ (\text{'//'} factor)\ *\right]\right]^{Python}$$
$$= T\left[\left[factor\right]\right]^{Python}(\text{'/'} T\left[\left[factor\right]\right]^{Python}, \text{.floor}) *$$
$$T\left[\left[\ (\text{'+'}|\text{'-'}) factor|power\right]\right]^{Python}$$
$$= (\text{'+'}|\text{'-'}) T\left[\left[factor\right]\right]^{Python} | T\left[\left[power\right]\right]^{Python})$$
$$T\left[\left[\ atom\ [\text{'**'}\ factor]\right]\right]^{Python} = T\left[\left[atom\right]\right]^{Python}, * * \text{'} T\left[\left[factor\right]\right]^{Python})$$
$$T\left[\left[\ \text{NAME}|\ \text{NUMBER}|\ \text{'true'}|\text{'false'}\ |\ \right]\right]^{Python}$$
$$= \text{'words'}|\text{'}\ \text{numeric'}|\ \text{'true'}|\ \text{'false'})$$
$$T_{comp\_OP}\left[\left[\text{'}<\text{'}\right]\right]^{Python} = \text{'}<\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}>\text{'}\right]\right]^{Python} = \text{'}>\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}==\text{'}\right]\right]^{Python} = \text{'}==\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}>=\text{'}\right]\right]^{Python} = \text{'}>=\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}<=\text{'}\right]\right]^{Python} = \text{'}<=\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}<>\text{'}\right]\right]^{Python} = \text{'}!=\text{'}$$
$$T_{comp\_OP}\left[\left[\text{'}!=\text{'}\right]\right]^{Python} = \text{'}!=\text{'}$$
$$T\left[\left[\ \text{':'}\ test[\text{'}=\text{'}\ test]\right]\right]^{Python} =$$
$$T\left[\left[expr\right]\right]^{Python}(\ T\left[\left[comp\_op\right]\right]^{Python}\ T\left[\left[expr\right]\right]^{Python}) *$$

上記ではアルファベットを NAME で，数字を NUMBER で表している．また，NEWLINE で改行を，INDENT で字下げを，DEDENT で字上げを表している．

これより，Python から Ruby への翻訳例自体で Python と Ruby で意味を保存した翻訳が出来た．

5.4 比較表

5.3 節で記述した翻訳規則の比較表をまとめると表 2 となる．

表 2 比較表

	Python	Ruby
加算	1+2	1+2
減算	1-2	1-2
乗算	1*2	1*2
除算	3/1	3/1.to_f
切り捨て除算	3//2	3/2 (3/2).floor
剰余演算	3%2	3%2
小なり，大なり	<, >	<, >
または，かつ	|,or #または &,and #かつ	|,||,or #または &,&&,and #かつ
同値	3==4,3 is 4	3==4,3===4
同値ではない	3!=4,3 is not 4	3!=4
否定	not	!,not
構文の中身の判断	インデントで判断	endで判断

if	if i==0:pass
	elif i==1:pass
	else:pass
	#戻り値はなし
		if i==0 then
		elsif i==1 then
		else
		end
		#戻り値は最後に評価した式の値 or nil

6 まとめ

本研究では，標準ライブラリを使用して抽象構文木を生成し，それを翻訳する翻訳ソフトウェアの構成について述べて，四則演算と if 文を Python から Ruby へと翻訳する翻訳規則が出来た．

7 今後の課題

本研究では，四則演算と if 文の翻訳規則を構築したが，我々の目標は南山大学のプログラミング基礎の授業で学んだ範囲を翻訳することである．

そのため，今後は繰り返し構文や配列等の抽象構文木の翻訳を実装する．

8 参考文献

[1]"The 2017 Top Programming Languages".IEEE SPECTRUM,〈https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages〉(accessed 2017-10-4)

[2]清水崇之，小川翔二郎，松原俊一，Duerst, M.: 情報学広場:情報処理学会電子図書館，情報処理学会（オンライン）〈https://ipsj.ixsq.nii.ac.jp/ej/index.php?active_action=repository_view_main_item_detail&page_id=13&block_id=8&item_id=107791&item_no=1〉(2011)

[3]中田育男:コンパイラ，pp.11-139，産業図書(1981)

[4]mtSystems（Migration Technology Systems）GmbH,mtSystems,〈https://www.mtsystems.com〉(accessed 2017-09-27)

[5]Python Software Foundation:Python 3.6.1 ドキュメント，Python,〈https://docs.python.jp/3/index.html〉(参照 2017-09-27)

[6]まつもとゆきひろ:Ruby2.4.0 リファレンスマニュアル，オブジェクト指向スクリプト言語 Ruby リファレンスマニュアル，〈https://docs.rubylang.org/ja/latest/doc/index.html〉(参照 2017-09-27)