自然言語処理を用いた初学者向けプログラミング支援

秋信 有花　　多田 拓　　倉光 君郎
日本女子大学理学部, 横浜国立大学大学院理工学府
m1616003ay@ug.jwu.ac.jp，kuramitsuk@fc.jwu.ac.jp

1 はじめに

初学者がプログラミングを始めたとき，戸惑うことの 1 つが構文エラーや名前エラーなどのエラーである [2]．プログラミング言語は，自然言語と異なり，厳密な構文定義と意味論から成り立ち，少しでも規則に外れた書き方を認めてくれない．そして，このようなプログラミング言語特有の厳密さが，プログラミングを始めた初学者の障壁となることも少なくない．

日本語プログラミング言語は，我々が日常使っている言語でプログラミングすることで，初学者の障壁を下げようとする取り組みといえる．しかし，日本語プログラミング言語も，独自の言語構文を覚える必要があり，自然言語のような柔軟さがあるわけではない．

本論文では，プログラミング言語中で自然言語叙述を認め，初学者の感じる障壁を緩和する手法の提案を行なう．既存のプログラミング言語の中での自然言語叙述を認め，自然言語処理技法を導入することで，自然言語記述部のコード変換を行なう．ユーザに構文や語彙などの制限のない叙述を認めることで，プログラミングの自由度の向上が期待できる．また，初学者のモチベーション低下の要因でもあるエラーを減らすことで，プログラミング初学者の継続的な学習のサポートに繋がると考えている．

本論文の構成は以下の通りである．2 節では，本研究の問題を定義する．3 節では，コード変換のアルゴリズムについて述べる．4 節では，実装について述べる．5 節では，本論文を総括する．

2 動機と問題設定

まず，本研究の動機となった Puppy を紹介し，課題をまとめる．

2.1 Puppy

Puppy[1] (図 1) は現在我々が開発しているプログラミング入門言語である．Puppy はプログラミング教育で定評のある Python の簡易版であり，本格的なプログラミングへのステップアップとして設計された．また，物体オブジェクトを簡単に記述でき，この物体オブジェクトの動きから，プログラムの動作を直感的に理解しやすくなるよう配慮されている．

以下の例は，床の上に赤いボールが跳ねている様子を記述したものである．実行すると，図 1 のような実行結果が得られる．

Listing 1: 物体オブジェクトの生成例

```
1 Rectangle(0, -500, 1100, 50, isStatic=True)
2 Circle(0, 100, 100, fillStyle='#e60033',
      restitution=1.0)
```

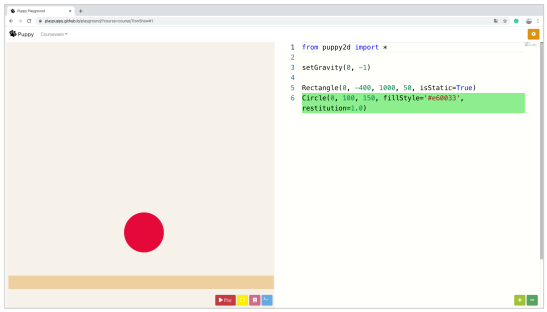

図 1: Listing 1 の実行結果

物体オブジェクトはコンストラクタで生成することができ，(x,y) 座標，また四角形 (Rectangle) には幅と高さ，円 (Circle) には半径を数値で与えることで画面上に配置ができる．さらに，物体の物理的な特性はプロパティとして簡単に与えることができるようになっている．Listing 1 の例では，”isStatic=True” で物体 (Rectangle) が固定されることを，”restitution=1.0” で物体 (Circle) の跳ね返り係数が 1.0 であることを記述している．

2.2 自然言語による記述

我々が目指すものは，Puppy ソースコード内で自然言語叙述を部分的に認め，自然言語記述部をコードに変換することで，プログラミングを支援することである．

もう一度，先ほどの例に戻る．床の上に赤いボールが跳ねている様子を，自然言語による記述で書いてみると，例えば，以下のように記述される．

Listing 2: 物体オブジェクトの生成例

```
Rectangle(0, -500, 1100, 50, 固定)
Circle(0, 100, 100, 色は赤，よく跳ねる)
```

この場合，自然言語部分は以下のように変換されることが望まれる．

- 固定 → isStatic=True
- 色は赤 → fillStyle='#e60033'
- よく跳ねる → restitution=1.0

2.3 プロパティ変換問題

ここで，これまでの例を一般化し，プロパティ変換問題として定義したい．

プロパティは，キー k と値 v からなる組である．本稿では，(k, v) と書く．つまり，先ほどの例にあった `fillStyle='#e60033'` は，$(\mathit{fillStyle}, '\#\mathtt{e60033}')$ となる．読みやすさのため，キーはイタリック体，値は等幅フォントで書く．

プロパティ集合 P は，プログラム意味論で解釈可能な 0 個以上のプロパティからなる集合である．

$$P = \{(k_1, v_1), (k_2, v_2), ...\}$$

P に含まれるプロパティのキーは交わらない，つまり P 上の任意の 2 つのキーに対して $k_i \neq k_j$ である．

今まで見てきた通り，我々は「自然言語で記述された入力文字列 x を P へ変換」したい．ここで，P がプログラム上で正しく解釈可能される必要がある．型安全性の観点から解釈を保証するため，型環境 Γ を導入する．型環境 Γ は，プログラム上で解釈可能なキーの集合を K，型の集合を T とすると，次の通りの写像となる．

$$\Gamma : K \to T$$

型環境 Γ は，キーの型付けをサポートするもので，直感的にはキーから型 t への辞書と考えてよい．

以上より，プロパティ変換問題は以下の通り定義できる．

定義 1 (プロパティ変換問題) ある文字列 x を与えられた型環境 Γ のもと，*0* 個以上のプロパティ集合 P に変換する

3 アルゴリズム

本節では，我々が開発しているプロパティ変換問題アルゴリズムを示す．

3.1 全体像

プロパティ変換問題アルゴリズムの概要は次の通りである．

- 入力文字列 s を，抽象構文木 e に変換する
- 次の 2 つの場合について，再帰的に考える
  - 構文木 e からキーが明示されている場合は，キー k と値 v に相当する部分をそれぞれ変換する
  - 構文木 e からキーが明示されていない場合は，必要に応じキー k を推定し，値 v を変換する．

以下，まず構文木 e のデータ表現，変換用辞書などの道具を定義し，アルゴリズム本体について述べる．

3.2 抽象構文木

抽象構文木は，以下の通り，再帰的に定義されるものとする．

e	::=	Let(e, e)	定義
	|	Ad(e, e)	修飾
	|	Not(e)	否定
	|	'x'	字句

以下に例を示す．

赤	’赤’
色は赤	Let(’色’, ’赤’)
跳ねる	’跳ねる’
よく跳ねる	Ad(’よく’, ’跳ねる’)
色は明るい赤	Let(’色’, Ad(’明るい’,’赤’))

本アルゴリズムにおいて，構文解析器が鍵となるが，この詳細は実装の節で述べる．

3.3 変換辞書

我々は，変換辞書 D を用意する．変換辞書は，字句 x に対して，(k, v) が登録された写像である．次は，辞書の定義例である．

$$
\begin{aligned}
D('色') &= (fillStyle, \_) \\
D('赤') &= (\_, '\#\texttt{e60033}' : str) \\
D('跳ねる') &= (restitution, \texttt{0.8} : float) \\
D('固定') &= (isStatic, \texttt{True} : bool)
\end{aligned}
$$

_ は特殊な記号で定義なしと解釈する．型は，対応のプログラミング言語に依存する．本稿では，文字列 str, 数値 $float$, 論理値 $bool$ と仮定する．

3.4 構文木からの変換

$\mathcal{T}(e) = (k, v)$ は，構文木からプロパティへの変換する関数である．関数は，図 2 に示す変換規則で再帰的に定義される．

以下，引き続き同じ例を用いて簡単に説明する．「色は赤」の場合，構文解析を行なうと，Let(’色’, ’赤’) になる．これに変換規則 (T-LET) と (T-TOKEN) を適用すると，($fillStyle$, #e60033) へ変換される．

また，「よく跳ねる」の場合，構文解析を行なうと，Ad(’よく’, ’跳ねる’) になる．これに変換規則 (T-AD) と (T-TOKEN) を適用すると，($restitution$, 1.0) へ変換される．「よく」は変換規則 (T-AD) に現れる ϕ 関数を適用する．ϕ 関数によって，値が増加する．

3.5 Word2vec を用いた類似度計算によるマッピング

辞書のエントリ数，つまり $dom(D)$ は有限である．

今まで見てきた通り，変換関数 $\mathcal{T}$ は，変換辞書 D に基づいて変換を行なっている．しかし，自然言語による表現の方法は無限に考えられるため，変換辞書 D に登録されていない場合も多い．この問題を解決するため，字句 x が $x \notin dom(D)$ の場合，word2vec による類似度計算を行ない，近似的にマッピングする．

変換辞書 D の自然言語と入力された自然言語との類似度計算結果が大きいものを変換候補として返すようにしている．この類似度計算により，変換辞書 D の量も最小限に抑えることができ，ユーザの自然言語記述の制約もなくすことができる．

$$\boxed{Let(e_1, e_2)}$$

$$\frac{\mathcal{T}(e_1) = (k_1, \_) \qquad \mathcal{T}(e_2) = (\_, v_2)}{\mathcal{T}(Let(e_1, e_2)) = (k_1, v_2)} \text{ (T-LET)}$$

$$\boxed{Ad(e_1, e_2)}$$

$$\frac{\mathcal{T}(e_2) = (k, v) \qquad \phi\text{は，値関数}}{\mathcal{T}(Ad(e_1, e_2)) = (k, \phi(v))} \text{ (T-AD)}$$

$$\boxed{Not(e)}$$

$$\frac{\mathcal{T}(e) = (k, v)}{\mathcal{T}(Not(e)) = (k, \neg v)} \text{ (T-NOT)}$$

$$\boxed{\texttt{'x'}}$$

$$\frac{D('\texttt{x}') = (k, v)}{\mathcal{T}('\texttt{x}') = (k, v)} \text{ (T-TOKEN)}$$

図 2: 変換規則

4 実装

本節では，プロパティ変換問題アルゴリズムの実装，また Puppy との統合方法について述べる．

4.1 アルゴリズムの実装

変換関数 $\mathcal{T}$ の実装は，Python3 を用いて行なった．入力文字列から構文木への変換は，PEGPY と CJ 文法 [3] を使用した．図 3 は，CJ 文法による構文解析の例である．このように出力されるタグ情報から，構文木を構築している．変換辞書 D は，現在のところ，50

エントリーほど登録されている．また，word2vec の類似度計算には，東北大学乾研究室の日本語 Wikipedia エンティティベクトルを使用した．

```
>>> 色は赤でよく跳ねる
[#S
  [#NChunk
    [#Noun '色']
    [#Subject 'は']
  ]
  [#NChunk
    [#Noun '赤']
    [#Object 'で']
  ]
  [#Adverb 'よく']
  [#VChunk
    [#Verb1 '跳ね']
    [#Base '']
  ]
]
```

図 3: CJ による構文解析の例

4.2 Puppy の統合

プロパティ変換問題は Monaco Editor[1]を介して Puppy へ統合した．Monaco Editor は Microsoft 社が提供するコードエディタライブラリである．Monaco Editor の API である CodeAction によって変換したプロパティを提案する UI を実装した．

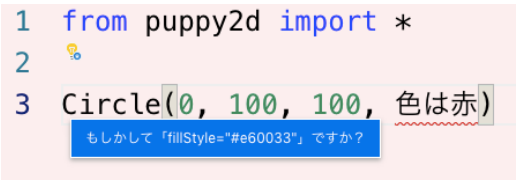

図 4: Monaco Editor によるプロパティの提案

図 4 にプロパティの提案の例を示す．コードエディタに記述された「色は赤」という文字列に対してプロパティ変換を行ない，結果である「fillStyle = "#e60033"」を CodeAction の UI によって提案している．さらに提案のポップアップをクリックすることで，文字列を提案されたコードに置き換えることが可能である．また，プロパティ変換処理は API サーバとしてスタンドアローンな仕様にしている．これは，プロパティ変換処理に word2vec の学習済みモデルが必要であり，サーバレスアーキテクチャを採用している Puppy 本体との親和性が低いためである．そのため，Puppy から API を利用する形でプロパティ変換を統合している．

今後，Puppy の評価実験に合わせて，プロパティの提案による学習効果も検証していきたいと考えている．

[1]https://microsoft.github.io/monaco-editor/

5 まとめ

本稿では，自然言語処理を用いたプログラミング支援について紹介した．構文解析や word2vec を用いた類似度計算によるマッピング処理を導入することによって，ユーザに構文や語彙などの制限のない叙述を認めることが可能になる．この支援によって，多くのプログラミング初学者が，プログラミング言語特有の厳密さに苦しむことなく，スムーズなプログラミング体験ができるようになると期待している．

今後は，より自然言語に近い表現を記述しても，プログラミング言語への変換ができるようにしていきたい．現在は単語ベクトルによる類似度計算を行なっているが，文単位での意味や性質も捉えることで実現できると考えている．また，日本語プログラミング言語も設計予定であり，本支援と組み合わせることで，より記述しやすい日本語プログラミング言語を目指したい．

参考文献

[1] Taku Tada, Yuka Akinobu, Makoto Sakane, and Kimio Kuramitsu. Puppy: an educational simplification of python with live playground. In *Proc. of the ACM SPLASH/LIVE2019*, 2019.

[2] 岡本雅子. ぺた語義 : はじめてのプログラミングとつまずき. 情報処理, Vol. 56, No. 6, pp. 580–583, may 2015.

[3] 若杉祐依, 秋信有花, 渡邉遥輔, 倉光君郎. 解析表現文法による CNL 日本語文法の試作. 言語処理学会第 26 回年次大会予稿集, 2020.