初学者のためのレゴマインドストーム EV3 を用いた C プログラムから Python プログラムへの翻訳器の設計

2017SE040 久保井彩香　2017SE080 諏訪貴大　2017SE113 吉山大輔
指導教員：横山哲郎

1　はじめに

1.1　背景

組込みエンジニアは，近年 IoT 技術の進歩によって需要が高まっている反面，人材の確保が求められている [1]．現在 100 種類以上ある各プログラミング言語は得意な問題解決分野が異なっているため，その利用目的やシステムによってプログラミング言語を使い分ける必要がある．プログラミング学習者が新たにプログラミング言語を学ぶ場合において，一から学習する際に時間や費用，労力といった学習コストがかかるといった問題が生じる．実際，現代のプログラミング教育において，一つ目のプログラミング言語を学ぶ時点で生徒がその難しさで挫折してしまうことも少なくない [2]．

1.2　アプローチ

手続き型は分かりやすいというメリットがある反面，コードの再利用性が低いというデメリットがある．一方，オブジェクト指向型はコードの再利用ができる他，クラス化されているためコードの可読性，プログラムの品質が向上するというメリットもある．手続き型からオブジェクト指向型に翻訳を行うことで手続き型のメリットを保ちながらデメリットを克服し，手続き型で入力を行い，オブジェクト指向型で出力することで学習難易度の高さも克服されると考える．また本研究では，C 言語と Python の 2 言語を使用する．C 言語は組込みソフトウェア開発に使われる言語の 6 割を占めている [3]．Python は近年話題になっている人工知能の分野で使われており，これからの学習希望人口が多くなるのではないかと考えられている．またソースコード内では，C 言語と Python での言語差異が大きく，かつ初学者がつまづきやすい反復と関数を中心にコード変換を行う [4]．前述した理由からレゴマインドストーム EV3 のサンプルコードを用いることによって翻訳範囲を組込み系上に制限した．

1.3　目的

本研究の目的は学習効率の向上を目的とした組込み系上の初学者学習支援ソフトウェアである翻訳器の設計である．選択・反復・関数の一部を対象とし，ソース言語を C のターゲット言語を Python として設計する．つまり，設計する翻訳器は手続き型からオブジェクト指向型へ自動翻訳を行うものである．そのために Trudel らの提案した C から Eiffel の自動翻訳の方法である AutoOO[5] を参考にした．Trudel らの研究ではモダンな環境でレガシーコードを再利用することを目的としている．そのためにはオブジェクト指向リエンジニアリングを行う必要がある．オブジェクト指向リエンジニアリングとは再設計されたオブジェクト指向プログラムを構築することであり，Trudel らの研究では AutoOO が使われている．本研究においても Trudel らの研究と同様に手続き型抽象構文木からオブジェクト指向型抽象構文木の変換に AutoOO を使用する．

1.4　研究課題

本研究の研究課題を 5 つ挙げる．1 つ目はビルトイン型や制御フローに関する基本的な構文の一部について C 言語から Python への翻訳規則を示すことである．2 つ目は C 言語の構造体の間の関係を解析して Python のクラス間の適切な継承関係に変換することである．3 つ目は C API から Python API への変換が可能か評価を行うことである．4 つ目は翻訳器の作成において中間言語のある/なしの評価を行うことである．5 つ目は Lego Mindstroms EV3 を対象にしたプログラムにおける頻出のプログラムパターンを用いた C2Python の有効性の評価を行うことである．

2　関連研究

2.1　字句解析・構文解析

字句解析とは，広義な意味での構文解析における前半の処理部分のことで，プログラムを最小単位に分割する処理のことである．構文解析とは，字句解析によって得られた語彙の列の構造を分析し，構文木を生成することである．構文木には構文解析木と抽象構文木がある．まず構文解析木とは，入力した情報を全て表示した具体的な構文木である．抽象構文木とは，構文解析木を抽象化したものである．括弧や終端記号などの不必要な情報を取り除いたものである．しかし，コンパイラを実行する上で必要な情報は残す必要がある．

2.2　レガシーコードの自動リエンジニアリング

C 言語のような低級言語から高級言語への変換は，低級言語よりも先進的な環境においてレガシーコードを再利用することができるというメリットがある点において実用的な影響があるといえる．本文献 [5] の具体的なアプローチは，C のソースコードを CIL によって中間処理を行い，出力された C を C2Eif によって手続き型の Eiffel に変換する．さらに出力された手続き型の Eiffel を AutoOO を用いてオブジェクト指向型の Eiffel に変換，Eiffel コンパイラによってバイナリーコードを出力する．そのための

C2Eif，AutoOO の設計を行った．AutoOO はクラスのコレクションで構成されるオブジェクト指向設計を作成する．生成されるクラスは目的が異なる 2 種類でありデータ型クラスとバンドルクラスである．AutoOO は，1. ソースファイル分析 2. 関数シグネチャ分析 3. コールグラフ分析 4. 継承分析 の 4 つのステップでデータ型とバンドルクラスを生成する．また，AutoOO はリエンジニアリングで生成されたクラスの可読性を向上させるためにコントラクトと例外を導入する．結論として，現代のプログラミング言語にオブジェクト指向機能が広く採用されていることとレガシー言語の手続き的な性質から現代のオブジェクト指向への順応を考慮するとリエンジニアリングの必要性は保証されるといえる．

3 設計

3.1 概要

C2Python は，C プログラムを MicroPython プログラムに変換するコンパイラであり，学習効率の向上を目的とした組込み系上の初学者学習支援ソフトウェアである．

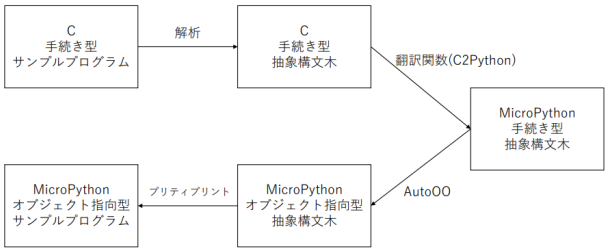

図 1 C2Python の動作概要

図 1 は，本研究の翻訳器の動作の全体像を示している．C2Python は，字句解析器・構文解析器で解析された C 言語（手続き型）のサンプルプログラムを入力とし，字句解析・構文解析を行い，C プログラムを抽象構文木に変換する．C2Python は，C 言語のプログラムの抽象構文木を MicroPython の抽象構文木に変換し，文献 [5] の AutoOO を参考にオブジェクト指向型へと変換する．最後に MicroPython の抽象構文木から，プリティプリンタを用いてサンプルプログラムを生成する．

3.2 ビルトイン型の対応

翻訳規則を示すために，まずビルトイン型の対応を行った．ビルトイン型とは，あらかじめ用意されているデータ型（*int* 型など）である．C 言語と MicroPython のビルトイン型の対応を以下の表 1 にまとめた．表 1 は C 言語と MicroPython のビルトイン型の対応の一部を表している．

3.3 制御フロー

制御フローを指示する命令の変換 $\mathcal{T}_{CF}$ について記述する（図 2）．選択や反復は C と MicroPython では似ている部分が多く，変換が容易である．ここで，$\mathcal{T}$ は命令文を変換する関数，I は命令，c は条件式，TB，EB，LB は制御フロー内の命令文である．

表 1 ビルトイン型の対応

C Type	MicroPython Class
short, long, int	int
float, double	float

$$
\begin{aligned}
\mathcal{T}_{CF}\,(I1; I2) =& \mathcal{T}(I1)\\
& \mathcal{T}(I2)\\
\mathcal{T}_{CF}\,(\texttt{if}(c)\ TB\ \texttt{else}\ EB) =& \texttt{if}\ \mathcal{T}(c):\\
& \quad \mathcal{T}(TB)\\
& \texttt{else}:\\
& \quad \mathcal{T}(EB)\\
\mathcal{T}_{CF}\,(\texttt{while}(c)\ LB) =& \texttt{while}\ \mathcal{T}(c):\\
& \quad \mathcal{T}(LB)
\end{aligned}
$$

図 2 変換関数 $\mathcal{T}_{CF}$ の定義

3.4 継承の解析

MicroPython では構造体という概念がないため，C プログラムの構造体を継承を用いた対応付けを記述した．C プログラムの struct r を親クラス class $\mathcal{T}_{capitalize}[\![r]\!]$ とし，struct s を class $\mathcal{T}_{capitalize}[\![s]\!](\mathcal{T}_{capitalize}[\![r]\!])$ のように，親クラスに子クラスの継承を行っている．struct r を変換する際，class $\mathcal{T}_{capitalize}[\![r]\!]$ のようにクラス名は構造体名を大文字にする形で記述される．

3.5 PD 制御によるライントレースプログラムの変換

以下に示した図 3 と図 4 のプログラムは，PD 制御によってライントレースをするプログラムである．C 言語で定義した型を以下の様に MicroPython の型に変換した．

```
int r = 40; → target = 40 //目標値
int e1; → p_error //前回の偏差
int e2; → error //今回の偏差
float mv; → turn //制御量
float kp = 1.5; → KP = 1.5 //比例ゲイン
```

C プログラムでは while(1) によって無限ループの処理を設定しているため， while Ture に変換する．

```
while(1) → while Ture
```

図 3 では操作量 mv と if 文を用いて左右のモーターパワーを制御することによってライントレースを行っている．一方，図 4 の MicroPython プログラムでは robot.drive で角速度を制御する turn を用いることによってライントレースを行う．

```
mv = (kp * e2) + (kd * (e2 - e1) / 0.010);
 →
turn = error * KP + (error - p_error) * KD
//制御量を計算
```

```
void main_task(intptr_t unused)
{
    int r = 40;
    int e1;
    int e2 = 0;
    float mv;
    float kp = 1.5;
    float kd = 1;
    int lpower, rpower;

    ev3_motor_config(EV3_PORT_B, LARGE_MOTOR);
    ev3_motor_config(EV3_PORT_C, LARGE_MOTOR);
    ev3_sensor_config(EV3_PORT_3, COLOR_SENSOR)
        ;

    while (1) {
        e1 = e2;
        e2 = r - ev3_color_sensor_get_reflect(
            EV3_PORT_3);

        mv = (kp * e2) + (kd * (e2 - e1) /
            0.010);

        if (mv > 0) {
            lpower = 30;
            rpower = 30 * ((100 - mv)/100);
        } else {
            lpower = 30 * ((100 + mv)/100);
            rpower = 30;
        }

        ev3_motor_set_power(EV3_PORT_B, lpower)
            ;
        ev3_motor_set_power(EV3_PORT_C, rpower)
            ;
    }
}
```

図 3 ライントレースを行う C 言語プログラム（[6] を参考に作成）

turn は error * KP が目標値との差にゲインをかけた値を角速度に代入し制御する P 制御である．そこに (error - p_error) * KD を足すことで，1 時刻前の偏差と今の時刻の偏差を比較し偏差の増減に合わせ制御量を変化させている．そうすることでより滑らかな動作になる．図 3 と図 4 のようなプログラムを利用して，動作の振る舞いが同じコードの翻訳が可能となる翻訳器の作成を試みた．

4 評価

4.1 C の API から Python の API への変換の可能性

我々の研究した C2Python において，C の API から MicroPython の API への変換が可能であるかを評価した．選択，反復，関数を対象とした EV3 のプログラムを例に API の変換を行った．それぞれ個々のプログラムの中においては問題なくプログラムの実行できたという点で API 自体の記述変換は間違ってはいないために大多数の API においては変換が可能であると評価できるが，図 5 に示すモータータイプの設定 API においては人の手によって繋がれたポートを機械がどのようにして右か左かを判断して left_motor もしくは right_motor と判断するのかという点において，このままの翻訳例での翻訳は自動変換において理想的な動作が見込めないのではないだろうかという考察結果となった．

```
#!/usr/bin/env pybricks-micropython

from common import *
import time

left_motor  = Motor(Port.B)
right_motor = Motor(Port.C)
wheel_diameter = 56
axle_track = 123
robot = DriveBase(left_motor, right_motor,
    wheel_diameter, axle_track)

color_sensor = ColorSensor(Port.S3)

KP = 1.5
KD = 1.0
target = 40

p_error = 0

while True:
    p_error = error
    bright = color_sensor.reflection()
    error = target - bright
    turn = error * KP + (error - p_error) * KD

    robot.drive(70, turn)
```

図 4 ライントレースを行う MicroPython プログラム

```
    ev3_motor_config(PORT,LARGE_MOTOR)
                        →
    left_Motor = Motor(Port.port)
```

図 5 自動変換が見込めない API 変換例

4.2 中間言語あり

中間言語とは文献 [7] より言語翻訳時に変換前の言語と目的言語の間に挟む言語のことであり，本来であれば一対一で翻訳する状況を一対多の関係で翻訳することが可能になる．C 言語に字句解析を行い文字列を token 列に分解し，構文解析によって token 列を抽象構文木に変換した後，意味解析として抽象構文木をベースとして，変数の型や文などの意味情報を精査し，構文木を中間言語に変換するという手順だ．例えば，本研究においては C 言語から MicroPython に翻訳することが目的であるが，上記のように C 言語を中間言語に変換しておくことで今後の発展研究として，その中間言語の状態から MicroPython 以外の言語に変換が可能だ．これは翻訳言語が増えるほど有効であり，4 言語間の翻訳であれば中間言語を使用しない場合 6 通りの変換を行わなければいけないが，中間言語を設定した場合の翻訳では各言語から中間言語への 4 通りの変換で済むというメリットがある．

4.3 中間言語なし

プログラミング言語間の翻訳について中間言語を設定しないという場合は，言語間を直接翻訳するという方法がある．プログラミング言語は種類によって関数定義やコンパイル方法が異なるため，直接翻訳するには各言語を 1 つ 1 つ定義していく必要がある．そのため，言語を直接翻訳するコンパイラを作成することは，中間言語を作成することよりも労力は増える．例えば，3 つの言語間で 3 通り，4 つの言語間で 6 通りであり，言語の数が n から $n+1$ へ 1 つ増えると翻訳数が $n-1$ 通り増える．

4.4 C2Python の有効性の評価

列挙型におけるクラスのフィールドを連結させるような変換を自動翻訳によって行うことは，我々が参考にした MarcoTrudel らの C2Eiffel においては，列挙型におけるクラスのフィールドを連結させるような変換は不可能であった．我々の提案した LEGO MINDSTORM を利用したライブラリの変換においても列挙型におけるクラスのフィールドを連結させるような変換は適応されないのではないかと考察される．単に変換規則によって変換を行うとそれぞれの API の持つ意味を無視してクラスを生成してしまうことになりクラスとして意味を成さないものとなってしまうため，C 言語の API での複数の構造体を MicroPython の API では一つのクラスにまとめる変換を自動翻訳で行う場合において，現状の設計では有効性は低いといえる．

4.5 考察

上記で述べたこのような評価方法から，本研究のように 2 言語間の翻訳は中間言語なしの場合の直接翻訳する評価の方が最適だと考える．しかし，今後の発展研究のことを考えると中間言語を作成し翻訳をする方が最適であると考える．中間言語の作成にあたり，言語で異なる記述能力の違いなどそれぞれのパラダイムに合わせていく必要であるため難しい点は多くある．しかし，中間言語が作成でき実現できたならば，複数の言語に対応可能となり汎用性は高く，発展研究により高度なものが作成できるであろう．言語に依存しない翻訳器が作成できるのではないかと考えた．

5 おわりに

本研究をまとめると，新たにプログラミングを学ぶうえで時間や費用，労力といった学習コストがかかるという問題を解決するため翻訳器の設計を行った．本研究の成果として 5 つ挙げられる．1 つ目は，ビルトイン型や制御フローに関する基本的な構文の一部について C 言語から Python への翻訳規則を示した．2 つ目は，C 言語の構造体の間の関係を解析して Python のクラス間の適切な継承関係に変換を行った．C プログラムの構造体を継承を用いた対応付けを例として説明を行った．3 つ目は，C API から Python API への変換が可能か評価した．対象範囲を選択・反復・関数とし 9 つの例を挙げ C と Python それぞれサンプルプログラムを用いて評価を行った．4 つ目は，翻訳器の作成において中間言語のある/なしの評価を行った．コンパイラを用いて翻訳を行う場合，インタプリタを用いて翻訳を行う場合，中間言語を使用せずに翻訳を行う場合，中間言語を使用して翻訳を行う場合それぞれ場合分けをしメリット・デメリットを挙げ評価した．その結果，本研究の 2 言語間での翻訳では中間言語を使用せず直接翻訳する方だと考えた．こうした結果になったのは，評価方法を 4 つ提案し比較したためである．5 つ目は，Lego Mindstroms EV3 を対象にしたプログラムにおける頻出のプログラムパターンを用いた C2Python の有効性の評価を行った. 列挙型におけるクラスのフィールドを連結する自動翻訳は適応できない．本研究では翻訳器の作成という実装部分ができておらず，さらに本研究で行った設計も一部のみである. また翻訳器の作成ができていないため評価方法の確認にも至っていない．今後の発展研究のためにも実装を行い評価方法の確認及び検討を行う必要がある．さらに，自動翻訳でそれぞれの API が持つ意味を考慮してクラスの連結が行えるように拡張することも可能であると考える．以上が今後の課題として残っている．

参考文献

[1] 情報処理推進機構：2017 年度組込みソフトウェア産業の動向把握等に関する調査事業「組込みソフトウェアに関する動向調査」，調査報告書 (2018)． 入手先〈https://www.ipa.go.jp/files/000065314.pdf〉(参照 2020-10-03).

[2] 小松香爾：プログラミング教育の問題と対策，文京学院大学総合研究所経営論集，Vol.25, No.1, pp.83–104 (2015).

[3] 情報処理推進機構：2012 年度「ソフトウェア産業の実態把握に関する調査」，調査報告書 (2013)． 入手先〈https://www.ipa.go.jp/files/000026799.pdf〉(参照 2020-10-03).

[4] 間辺広樹，長島和平，並木美太郎，長　慎也，兼宗　進：C の学習経験を持つ高校生への Python の授業導入事例，情報教育シンポジウム論文集，pp.256–262 (2019).

[5] Trudel, M.: Automatic Translation and Object-Oriented Reengineering of Legacy Code, Doctoral thesis, ETH Zurich (2013).

[6] アフレル：ロボットで学ぶ C 言語プログラミング，アフレル (2018).

[7] 松澤芳昭，坂本一憲，大畑貴史，筧　捷彦：プログラミング教育のための多言語間プログラミング言語翻訳システム，情報教育シンポジウム 2015 論文集，pp.223–230 (2015).