Python の静的型検査器を活用したコレオグラフィックプログラミング言語の実装

岐阜大学大学院 自然科学技術研究科 知能理工学専攻　草刈研究室　恩田晴登

1. はじめに

一般的に，並行・分散プログラムは，デッドロック等の並行性に起因するエラーや非決定性問題の発見と修正が困難であるため構築が難しい．この問題の解決手法の一つとして，コレオグラフィが提唱されている．コレオグラフィは並行に動作する複数参加者の連携手順をまとめたプログラムであり，コレオグラフィックプログラミング言語によって記述される．コレオグラフィに準ずる各参加者のプログラムは，**エンドポイント射影**と呼ばれる操作によって自動的に生成される．生成された各参加者のプログラムはデッドロック等の並行性に起因するエラーがないことが理論的に保証されている．先行研究である Choral [1] は，Java を拡張したコレオグラフィックプログラミング言語の一つである．しかし，機械学習や IoT の分野で盛んに使用されている Python を拡張したコレオグラフィックプログラミング言語は著者が知る限りはまだ提案されていない．

本研究では，Python をベースとしたコレオグラフィックプログラミング言語 **PyChoral** を構築し，有用性を確かめる．これにより，機械学習や IoT を扱うプロジェクトにおいて、Python で並行・分散プログラムが実装可能となることを目指す足がかりとする．本研究の特徴は Python の型検査器である **Mypy** の型検査結果を活用することである．PyChoral の構文は Python と同一であることから，Python の IDE やライブラリを使うことができる．よって，Python の標準的な仕組みを保ったまま，単一のプログラムで並行・分散プログラムを記述できるという点で可用性をもつ．

2. PyChoral によるプログラミング

PyChoral は Python の構文や型システムを踏襲して構築されている．PyChoral を使ったコレオグラフィはクラスを用いて構築する．参加者間では，コレオグラフィの中で生成するチャネルを用いて通信する．Code 1 は，入力した整数が 2 で割り切れる回数をカウントするコレオグラフィ Div2 である．A と B は div2 メソッドを呼ぶ．A に引数で渡される整数値 n が 2 で割り切れる限り，n を 2 で割って，割った回数をカウントする．もし割り切れない場合は，それまでにカウントした回数を A から B へ送る．例えば，ユーザーコード側で A のメソッド div2 に $(8,0)$ を渡すと B は 3 を受け取る．2,3 行目はクラスのコンストラクタを表す．ここでは A から B へ object 型の値を送るチャネル chAB を生成している．

4-10 行目は A と B が連携して動作するメソッド div2 である．5 行目の if 文で，A がもつ整数 n が 2 で割り切れるかどうかで分岐する．この分岐の結果はまだ B には知らされない．6,7 行目は割り切れる場合，9,10 行目は割り切れない場合である．6,9 行目にある select は条件分岐の結果を示す列挙型の値を送るメソッドである．com は送信者の情報を含む任意の型をもつ値を送るメソッドである．

```
1  class Div2(Choreograhy2[A,B]):
2    def __init__(self) -> None:
3      self.chAB = Channel[object,A,B]()
4    def div2(self,n:At[int,A],c:At[int,A]) -> At[
         int,B]:
5      if (n % 2)@A() == 0@A():
6        self.chAB.select(OddEven.EVEN@A())
7        return self.div2((n//2)@A(),(c+1)@A())
8      else:
9        self.chAB.select(OddEven.ODD@A())
10       return self.chAB.com(c@A())
```

Code 1　PyChoral によるコレオグラフィ

クラス Div2(Choreograhy2[A,B]) からエンドポイント射影により，クラス Div2_A,Div2_B が生成される．参加者 A,B はプロセスから Div2_A,Div2_B のメソッド div2 を呼び出すことにより連携して動作する．Code 2 は，Code 1 から射影された B の Python プログラムである．型注釈や B が関連しない式や文は除去される．

```
1  class Div2_B():
2    def __init__(self):
3      self.chAB = Channel[object,A,B]()
4    def div2(self):
5      match self.chAB.select():
6        case OddEven.EVEN:
7          return self.div2()
8        case OddEven.ODD:
9          return self.chAB.com()
10       case _:
```

Code 2　参加者 B の Python プログラム

分岐を含むコレオグラフィのエンドポイント射影で重要なのは，ある参加者で発生した条件分岐をいかにして他の参加者に伝えるか，という仕組みを実現することである．これを実現するのが**マージ**である．PyChoral におけるエンドポイント射影およびマージの理論は Choral を模倣している．if 文におけるマージは，if 文の条件式を判断した参加者以外が条件式の結果に相当する列挙型の値を受信して分岐することにより実現する．PyChoral では Python3.10 から導入された match 文を使用している．match 文のケースは Code 1 の 6,9 行目にある select メソッドで送信し

た列挙型の値で分ける．Code 1 の 6 行目から射影により Code 2 の 5-7,10 行目が生成される．Code 1 の 9 行目からは射影により Code 2 の 5,8-10 行目が生成される．`case _` は共通のケースであるため，統合されて一つになる．`OddEven.EVEN`,`OddEven.ODD` は独立したケースであるため，マージ後の match 文にそのまま加える．これにより，条件分岐に関係ない参加者も分岐を考慮した形でエンドポイント射影される．生成された `A,B` の Python プログラムは，エンドポイント射影およびマージによりデッドロックが起こることなく連携して動作することが可能である．

3. PyChoral の設計と実装

PyChoral はエンドポイント射影において型情報を活用する．これは，すべての式や文が射影される参加者と関連するかどうかに基づいて射影をするためである．例えば，関数呼び出しの射影 $(\!|f(\bar{e}) : \tau|\!)^A$ は引数と戻り値の型に，射影される参加者の情報が含まれているかどうかで場合分けされる．ここで，$(\!|\ |\!)^A$ は参加者 A に対しての射影であることを示す．τ は返値の型情報である．引数 $\bar{e}$ は式のリストである．このようなエンドポイント射影を実現するために，PyChoral は Python の基本型やクラスに参加者の情報を含める形で拡張している．具体的な手法は次の 2 つである．

■ 型の拡張　参加者の情報を含む型として `At[T,R]` や `Choreograhy2[R1,R2]`，`Choreograhy3[R1,R2,R3]` を定義する．`At[T,R]` は参加者 `R` に割り当てられる値の型 `T` を表す型である．PyChoral における `At[T,R]` は Python の基本型 `T` と同様に扱える．`Choreograhy2[R1,R2]`，`Choreograhy3[R1,R2,R3]` は PyChoral プログラムでクラス宣言をする際に継承する基底クラスの型である．型パラメータに参加者の情報を持ち，参加者の人数で継承するクラスを指定する．

■ 式の拡張　Python のすべての式が属する型であるオブジェクト型に`@`演算子を追加で定義する．これにより，`exp@R()` のように式 `exp` に参加者 `R` を割り当てることができる．ここで，`exp` の型が `T` であれば `exp@R()` の型は `At[T,R]` となる．

`@`演算子および型クラス `At`,`Choreograhy2`,`Choreograhy3` はエンドポイント射影で参加者の情報を得るために用いられる．これらを実体を持たない Mypy の型宣言の形で定義することにより，各参加者のプログラムでは`@`演算子，型クラス `At`,`Choreograhy2`,`Choreograhy3` が消去される．

4. アプリケーションによる評価

Code 3 は Choral のプログラム例の一つである Parallel MergeSort を PyChoral で実装したものである (可読性向上のために型情報は省略する)．クラス `MergeSort` の参加者である Merger と 2 つの Sorter はパラメータ `M,S1,S2` として扱う．2,3 行目ではコンストラクタを記述している．`chS2M,chS1S2,chS2M` は型 `T` の値を双方向 (`R1`⇄`R2`) に送り合えるチャネル `SymChannel[T,R1,R2]` の型をもつ．4 行目はメソッド `sort` の定義であり，リストを分割する必要性を確認する条件式で構成される．5-15 行目は分割する場合である．10 行目でリストの中心 (`pivot`) を見つけ，それを境目にリストを分割して得られるサブリストを 2 つの Sorter に送信する．8,9 行目では分割されたリストをさらにソートするために，Merger と Sorter の役割を入れ替えながら MergeSort を再帰的に呼び出す．このように Merger と Sorter の役割を代えながらリストの分割ができなくなるまでソートを行う．各 Sorter によって順序付けられたリストは 15 行目にあるメソッド `merge` の呼び出しによって結合される．20 行目にある `merge` は Sorter がもつ 2 つのリストを Merger が結合するメソッドである．このメソッドはローカルであり，リストの結合を逐次的なアルゴリズムのように実行する．MergeSort のコードは，参加者の型情報により，リストのソートを並列して実行可能である Python プログラムを生成できる．さらに，それらは理論的にデッドロックフリー性をもつ．

```
1  class Mergesort(Choreograhy3[M,S1,S2]):
2    def __init__(self,m,s1,s2,chMS1,chS1S2,chS2M):
3      # constructor #
4    def sort(self,a):
5      if len(a)@self.m() > 1@self.m():
6        self.chMS1.select(MChoice.L@self.m())
7        self.chS2M.select(MChoice.L@self.m())
8        mb = Mergesort[S1,S2,M]( # arguments # )
9        mc = Mergesort[S2,M,S1]( # arguments # )
10       pivot = float(floor(len(a)/2))@self.m()
11       b = a[0:int(pivot)]
12       lhs = mb.sort(self.chMS1.com(b@self.m()))
13       c = a[int(pivot):len(a)]
14       rhs = mc.sort(self.chS2M.com(c@self.m()))
15       return self.merge(self.chMS1.com(lhs),self.
             chMS1.com(rhs))
16     else:
17       self.chMS1.select(MChoice.R@self.m())
18       self.chS2M.select(MChoice.R@self.m())
19       return a
20   def merge(self,lhs,rhs):
21       ...
```

Code 3　MergeSort.py

5. まとめ

本研究は Python を拡張したコレオグラフィックプログラミング言語 PyChoral を構築した．エンドポイント射影の際に参加者の情報を含む型情報を活用することで，不必要な式や文が排除された各参加者の Python プログラムが生成できる．PyChoral を IoT や機械学習を扱う Python プログラムに応用することが今後の課題である．

参考文献

[1] S. Giallorenzo et al. Choreographies as objects. *CoRR*, Vol. abs/2005.09520, , 2020.