2024 年度 卒業/修士論文

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

1224525023　恩田晴登

指導教員　草刈圭一朗

2024 年 2 月 5 日

目次

第 1 章　はじめに 1

第 2 章　前提知識・先行研究 2
2.1 コレオグラフィとエンドポイント射影 2
2.2 Choral 2
2.3 Mypy 3

第 3 章　PyChoral によるプログラミング 4

第 4 章　PyChoral の設計と実装 7
4.1 PyChoral の構文とエンドポイント射影の定義 7
4.2 エンドポイント射影の実装 9

第 5 章　PyChoral を利用したアプリケーションによる評価 13

第 6 章　まとめ，今後の課題 15

参考文献 16

付録 A　Projection, Merging, Normalizer の定義 17
A.1 Projection to Python 17
A.2 Merging 18
A.3 Normalizer 19

付録 B　射影後のデータ構造 21

付録 C　3 者間のコレオグラフィのランタイム 23

第 1 章　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．well-formed なコレオグラフィからは、**エンドポイント射影**と呼ばれる操作により、各参加者のプログラムを生成できる．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 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．

本論文の構成を以下に示す．まず，2 章で本研究の前提知識となるコレオグラフィックプログラミングと Mypy の概要を述べる．3 章では本研究で構築したコレオグラフィック言語である PyChoral について述べ，プログラミング例を示す．4 章では PyChoral の設計と実装について述べる．5 章では PyChoral を用いたアプリケーションを示し，Pychoral の有用性を確かめる．6 章で結論と今後の課題を述べる．付録 A には PyChoral におけるエンドポイント射影，マージ，正規化の全体像を示す．

PyChoral の実装，プログラミング例，アプリケーションのソースコードは以下から入手可能である．

`https://github.com/onharu/mypytest`

第 2 章　前提知識・先行研究

2.1　コレオグラフィとエンドポイント射影

コレオグラフィ [3] とは，並行に動作する複数の参加者の連携手順をまとめたプログラムである．コレオグラフィに従った各エンドポイントのプログラムは，デッドロック等の並行性起因のエラーが起こる恐れを排除できる．各エンドポイント (通信の参加者) のプログラムはエンドポイント射影 [5] により，自動的に抽出される．

エンドポイント射影 [5] とは，コレオグラフィから各参加者のプログラムを導出する操作である．コレオグラフィックプログラミング言語にはコンパイラが付属しており，コンパイラはエンドポイント射影理論によって並行に動作する実行可能なプログラムを出力する．

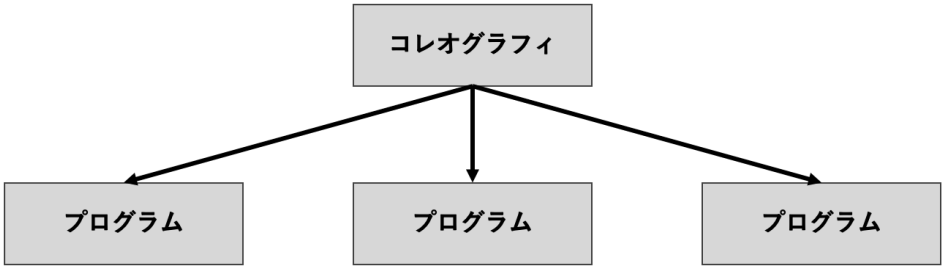


図 2.1　エンドポイント射影

2.2　Choral

Choral [4] は Java 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である．Choral は，並行・分散システムに従わせたいプロトコル全体を単一のプログラムとして作成できるため，デッドロックが発生しないようにコーディングする必要があったプログラマの負担が軽減される．Choral のオブジェクトには $T@(R_1, \cdots, R_n)$ という形式の型があり，Java の基本的なオブジェクトインターフェイス T に各参加者の情報となるパラメータ $R_1, \cdots, R_n$ が付随する．これにより，Choral のコンパイラは独自のパーサーを使ってエンドポイント射影をする．この際に参加者の情報を含む型をもとに，各参加者のプログラムをコレオグラフィに従った形で自動的に生成することができる．それぞれの Java プログラムはコレオグラフィに従っているため，相互関係によるバグがないことが保証されている．これは，マルチパーティなプログラムを記述する Java プログラマにとっては大きなメリットである．

2.3　Mypy

Mypy [2] は Python の静的型検査器であり，既存の Python コードに型注釈を追加することで，プログラムの実行前に型エラーを検出する．これにより，コンパイル時にバグの検出が可能になり，安全なコーディングが可能となる．

```
def greeting(name):
  return 'Hello ' + name

greeting(123)
greeting(b'Alice')
```

Code 2.1　型注釈のない Python コード

```
def greeting(name: str) -> str:
  return 'Hello ' + name

greeting('World')
greeting(3)
# Argument 1 to 'greeting' has incompatible type 'int'; expected 'str'
greeting(b'Alice')
# Argument 1 to 'greeting' has incompatible type 'bytes'; expected 'str'
```

Code 2.2　型注釈のある Python コード

図 2.2 は Mypy を使用した際の型検査のプロセスである．Mypy はプログラムを一度抽象構文木に変換し，構文木を探索しながら型エラーがないか検査を行う．型検査の結果はコンパイル終了時にテキストエディタに表示される (Code 2.2).

図 2.2　Mypy プロセス

第 3 章　PyChoral によるプログラミング

PyChoral は Python ベースのコレオグラフィックプログラミング言語である．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 を使用するコレオグラフィはプログラムのトップレベルにクラス定義を置き，それを用いて構築する．通信の参加者間では，コレオグラフィの中でコレオグラフィの中で生成するチャネルを用いて通信をする．

Code 3.1 は，参加者 A から参加者 B に入力した整数値が 2 で割り切れる回数を伝達するコレオグラフィである．A と B はそれぞれ div2 メソッドを呼ぶ．A には整数値 num と割った回数をカウントする値 count を引数として送る．整数値 num が 2 で割り切れる限り，num を 2 で割って，割った回数をカウントする．割り切れなくなった場合は，それまでにカウントした回数を A から B へ伝える．このコレオグラフィをユーザー側で使用するときは以下のようなプログラムになる．回数を数える変数 count は 0 に指定する．n は任意の整数値を代入できる．例えば，$n = 8$ の場合は B に整数値 3 が伝わる．$n = -3$ の場合はに整数値 0 が伝わる．

```
def run_A():
    div_a = Divide2_A()
    div_a.divide_by_two(n,0)
def run_B():
    div_b = Divide2_B()
    div_b.divide_by_two()
```

1 行目はクラス定義であり，型パラメータ A および B が通信の参加者となる．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．このプログラムでは A から B へ列挙型の OddEven と int 型のカウント数 count を送る場合があるため，どちらの型も送信できるように object 型でコンストラクタを生成する．

4-10 行目は A と B が連携して動作するメソッド divide_by_two である．メソッドの引数 num,count は参加者 A がもつ int 型の値であり，戻り値は参加者 B がもつ int 型の値である．5 行目の if 文では，A がもつ整数 num 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 行目では select メソッドにより条件分岐の結果を示す列挙型の値を A から B へ送っている．6 行目では num が偶数であるということを知らせる列挙型の値 EVEN を B へ送っている．9 行目では num が奇数であるということを知らせる列挙型の値 ODD を B へ送っている．7 行目では，メソッド divide_by_two の引数に，num を 2 で割った値と 1 だけ加算されたカウント数を引数として再帰呼び出しをしている．10 行目では com メソッドによりカウントした整数値を A から B へ送っている．

```
1 class Divide2(Choreograhy2[A,B]):
2     def __init__(self) -> None:
3         self.chAB : Channel[object,A,B] = Channel[object,A,B]('A','B')
4     def divide_by_two(self,num:At[int,A],count:At[int,A]) -> At[int,B]:
5         if (num % 2@A())@A() == 0@A():
6             self.chAB.select(OddEven.EVEN@A())
7             return self.divide_by_two((num // 2)@A(),(count + 1)@A())
8         else:
```

```
9           self.chAB.select(OddEven.ODD@A())
10          return self.chAB.com(count@A())
```

Code 3.1　参加者 A,B のコレオグラフィ

クラス Divide2(Choreograhy2[A,B]) からエンドポイント射影により, クラス Divide2_A,Divide2_B が生成される．参加者 A,B はプロセスから Divide2_A,Divide2_B のメソッド divide_by_two を呼び出すことにより連携して動作する．Code 3.2 と Code 3.3 は，Code 3.1 からエンドポイント射影された各参加者の Python プログラムである．型注釈，それぞれの参加者が関連しない式や文は除去される．

```
1  class Diveide2_A():
2      def __init__(self):
3          self.chAB = Channel[object,A,B]('A','B')
4      def divide_by_two(self,num,count):
5          if (num % 2) == 0:
6              Unit.id(self.chAB.select(OddEven.EVEN))
7              return Unit.id(self.divide_by_two(num // 2, count + 1))
8          else:
9              Unit.id(self.chAB.select(OddEven.ODD))
10             return Unit.id(self.chAB.com(count))
```

Code 3.2　参加者 A の Python プログラム

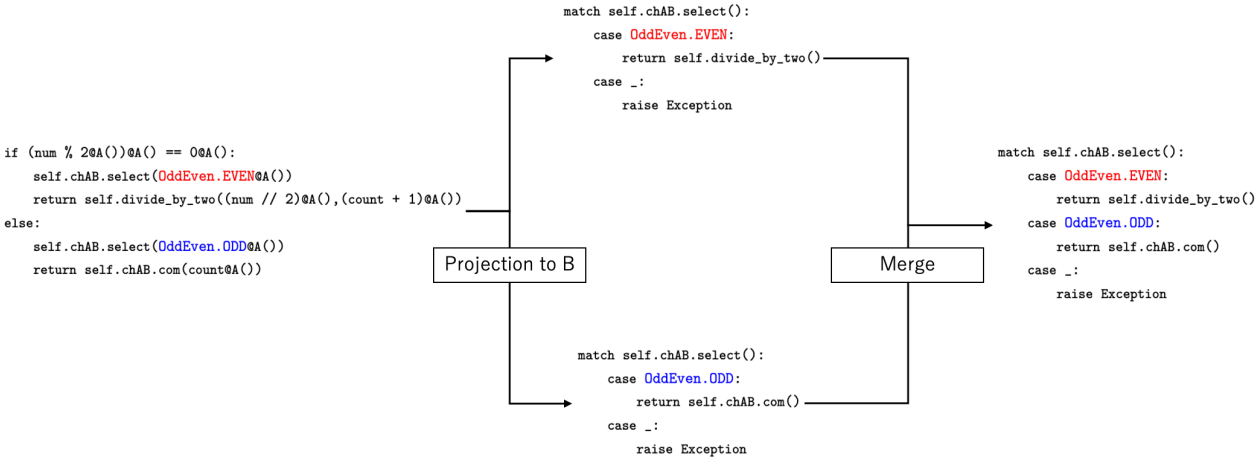
```
1  class Divide2_B():
2      def __init__(self):
3          self.chAB = Channel[object,A,B]('A','B')
4      def divide_by_two(self):
5          match self.chAB.select():
6              case OddEven.EVEN:
7                  return self.divide_by_two()
8              case OddEven.ODD:
9                  return self.chAB.com()
10             case _:
11                 raise Exception
```

Code 3.3　参加者 B の Python プログラム

参加者 A はチャネル chAB を介して B に値を送るため，select や com などのメソッド呼び出しでは戻り値がない．そのため，Code 3.2 の 6 行目以降に出てくるメソッド呼び出しは全て Unit 値となっている．

条件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 3.1 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

3.1 の 6 行目から射影により Code 3.3 の 5-7,10 行目が生成される．Code 3.1 の 9 行目からは射影により Code 3.3 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．

第 4 章　PyChoral の設計と実装

この章では PyChoral の構文およびエンドポイント射影の設計について述べる．その後，設計をもとにした本研究の実装について述べる．

4.1　PyChoral の構文とエンドポイント射影の定義

図 4.1 は Python の構文定義である．PyChoral の構文は Python と同一であり，プログラムの構文木はトップレベルからクラス定義 ($Class$)，関数およびメソッド定義 ($FuncDef$)，文 (Stm)，式 (Exp) のいずれかの構文要素で構成される．PyChoral では全ての式に参加者の情報が割り当てられる．参加者の情報は Python の基本型とともに Typed Annotation で表現されるか，リテラル値に@をつけて表現される．オーバーラインが引かれている Exp や Stm などはそれらのリストを表す．

Program	P	$::= P \cdot Class \mid P \cdot FuncDef \mid P \cdot Stm \mid P \cdot Exp \mid P \cdot EOF$
Class	$Class$	$::= \mathsf{class}\ id\ (\mathrm{Choreograhy}[\overline{A}], \overline{Exp})$
Function	$Func$	$::= \mathsf{def}\ id\ (\overline{id(:\tau)})$
Annotation	AN	$::= @id$
Type Annotation	τ	$::= Exp : \tau \mid id : \tau$
Literals	lit	$::= \mathrm{None} \mid \mathrm{True} \mid \mathrm{False} \mid "a" \mid \ldots \mid 1 \mid \ldots$
Expression	Exp	$::= lit@\overline{A()} \mid Exp.id \mid f(\overline{Exp}) \mid Exp.f(\overline{Exp}) \mid C[\overline{id}](\overline{Exp})$ $\mid Exp_1\ BinOp\ Exp_2$
Statement	Stm	$::= \mathbf{pass} \mid \mathbf{return}\ \ Exp \mid Exp\ ; \overline{Stm} \mid id = Exp\ ; \overline{Stm}$ $\mid Exp_1\ AsgOp\ Exp_2\ ; \overline{Stm} \mid \mathbf{if}\ \ Exp : Stm_1\ ; \mathbf{else} : Stm_2\ ; \overline{Stm}$ $\mid \mathbf{raise}\ \ Exp \mid \mathbf{assert}\ \ Exp$
Assign Op.	$AsgOp$	$\in \{\texttt{=}, \texttt{+=}, \texttt{-=}, \texttt{*=}, \texttt{/=}, \texttt{\%=}, \texttt{//=}\}$
Binary Op.	$BinOp$	$\in \{\texttt{|}, \texttt{\&}, \texttt{==}, \texttt{!=}, \texttt{<}, \texttt{>}, \texttt{<=}, \texttt{>=}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{//}, \texttt{\%}\}$

図 4.1　Syntax of PyChoral

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．エンドポイント射影の設計は構文木の項 Term と射影先の参加者 R をとり，同じ構文をもつ項へ移す写像とする．とあるコレオグラフィの参加者 A に対する PyChoral の項のエンドポイント射影は $(\!|\mathsf{Term}|\!)^A$ と記し，参加者 A の振舞いを表す Python の項となる．PyChoral の構文に無く，安全でないプログラムを射影する場合は未定義としてエラーが出力される．例えば，メソッド呼び出し ($Exp_1.f(\overline{Exp_2})$) はレシーバオブジェクト Exp_1，引数 $\overline{Exp_2}$ およびメソッド呼び出しの戻り値の型情報に射影される参加者の情報があるかどうかで場合分けをする．以

下はメソッド呼び出しのエンドポイント射影の定義である．

$$(\!|Exp_1.f(\overline{Exp_2}) : \tau|\!)^A = \begin{cases} (\!|Exp_1|\!)^A.f(\overline{(\!|Exp_2|\!)^A}) \\ \qquad\qquad \text{if} \quad A \in \text{rolesOf}(Exp_1) \wedge A \in \text{rolesOf}(\overline{Exp_2}) \\ \qquad\qquad\qquad\qquad \wedge A \in \text{rolesOf}(Exp_1.f(\overline{Exp_2})) \\ \texttt{Unit.id}((\!|Exp_1|\!)^A.f(\overline{(\!|Exp_2|\!)^A})) \\ \qquad\qquad \text{if} \quad A \in \text{rolesOf}(Exp_1) \wedge A \notin \text{rolesOf}(Exp_1.f(\overline{Exp_2})) \\ \texttt{Unit.id}((\!|Exp_1|\!)^A, \overline{(\!|Exp_2|\!)^A}) \qquad \text{otherwise} \end{cases}$$

戻り値の型情報に射影先の参加者が含まれていない場合は Unit 値を返す．レシーバオブジェクトや引数はそれぞれ再帰的に射影される．例えば，Code 3.1 の 10 行目にあるメソッド呼び出し `self.chAB.com(count@A())` は参加者 A,B に対して，以下のように射影される．

$$\begin{aligned} (\!|\texttt{self.chAB.com(count@A())}|\!)^A &\Longrightarrow \texttt{Unit.id}((\!|\texttt{self.chAB}|\!)^A\texttt{.com}((\!|\texttt{count@A()}|\!)^A)) \\ &\Longrightarrow \texttt{Unit.id(self.chAB.com(count))} \end{aligned}$$

$$\begin{aligned} (\!|\texttt{self.chAB.com(count@A())}|\!)^B &\Longrightarrow (\!|\texttt{self.chAB}|\!)^A\texttt{.com}((\!|\texttt{count@A()}|\!)^A) \\ &\Longrightarrow \texttt{self.chAB.com(Unit.id)} \end{aligned}$$

メソッド呼び出しの場合分けに出てくる rolesOf( ) とは，式の型情報を参照し，その型から参加者の情報を文字列の集合として返す関数である．`self.chAB.com(count@A())` の場合では，$\text{rolesOf}(Exp_1) = \{\text{A,B}\}$, $\text{rolesOf}(\overline{Exp_2}) = \{\text{A}\}$, $\text{rolesOf}(Exp_1.f(\overline{Exp_2})) = \{\text{B}\}$ となる．

次に，3 章で述べた条件分岐によるエンドポイント射影とマージの定義について述べる．文 Stm は，文中に現れる式や文をそれぞれ射影する形式を取る．例えば return 文のエンドポイント射影は $(\!|\textbf{return}\ Exp|\!)^A = \textbf{return}\ (\!|Exp|\!)^A$ となり，戻り値である式 Exp のエンドポイント射影が再帰で行われた return 文が新しく Python プログラムとして生成される．しかし，select メソッド呼び出しの式文と if 文は，他の文とエンドポイント射影の形式が異なる．

メソッド呼び出し以外，またはメソッド名が select 以外のメソッド呼び出しである式文は，他の文と同じように再帰的に射影されていく．select メソッド呼び出しの式文である場合は，match 文に射影される．select メソッドは引数に列挙型の値 id をとり，その値は射影された match 文での場合分けに使用する．この場合を満たさない場合は，ワイルドカードを用いて例外とする．

$$(\!|Exp\ ; \overline{Stm}|\!)^A = \begin{cases} \textbf{match}\ (\!|Exp|\!)^A : \\ \quad \textbf{case}\ id : \overline{(\!|Stm|\!)^A}; & \text{if} \quad Exp = Exp_1\texttt{.select}(id\texttt{@}A()),\ id : \texttt{Enum} \\ \quad \textbf{case}\ \_ : \text{raise Exception}; \\ (\!|Exp|\!)^A; \overline{(\!|Stm|\!)^A} & \text{otherwise} \end{cases}$$

if 文は，条件式 Exp の型情報に射影先の参加者が含まれる場合，そのまま if 文の構文を保ったまま再帰的に射影が行われていく．そうでない場合は，条件分岐の結果がどうなろうと振舞えるようにするために，then 節と else 節に存在する後続の文を正規化し，マージする必要がある．

$$(\!|\textbf{if}\ Exp : Stm_1\ ; \textbf{else} : Stm_2\ ; \overline{Stm}|\!)^A = \begin{cases} \textbf{if}\ (\!|Exp|\!)^A : (\!|Stm_1|\!)^A\ ; \textbf{else} : (\!|Stm_2|\!)^A\ ; \overline{(\!|Stm|\!)^A} & \text{if} \quad \text{rolesOf}(Exp) = A \\ (\!|Exp|\!)^A\ ; [\![(\!|Stm_1|\!)^A]\!] \sqcup [\![(\!|Stm_2|\!)^A]\!]\ ; \overline{(\!|Stm|\!)^A} & \text{otherwise} \end{cases}$$

マージ演算子 ⊔ は，分岐のプログラムを結合する演算子である．match 文以外のマージは，結合される Python プログラムの式，文は等価であるとする．2 つの match 文のマージは，Exp のマージを条件式とする match 文となる．各 case に関して，両方に存在する各ケースは元のケースに続く文 $(Stm_a, ...)$ をマージしたケースを得る．片方にしかないケースはマージ後の match 文にそのまま加える．

match Exp :		**match** Exp' :		**match** $Exp \sqcup Exp'$:
case $id_a : Stm'_a$;		**case** $id_a : Stm''_a$;		**case** $id_a : Stm'_a \sqcup Stm''_a$;
...		...		...
case $id_x : Stm'_x$;	⊔	**case** $id_x : Stm''_x$;	=	**case** $id_x : Stm'_x \sqcup Stm''_x$;
case $id_y : Stm'_y$;				**case** $id_y : Stm'_y$;
		case $id_z : Stm'_z$;		**case** $id_z : Stm'_z$;
case _ : Stm'_{ex};		**case** _ : Stm''_{ex};		**case** _ : $Stm'_{ex} \sqcup Stm''_{ex}$;

4.2　エンドポイント射影の実装

4.1 節で述べたエンドポイント射影を実現するために，PyChoral は Python の基本型や型クラスに参加者の情報を含める形で拡張する．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[T1,R]** や **Choreograhy2[R1,R2]**,**Choreograhy3[R1,R2,R3]** を定義する．`At[T1,R]` は参加者 R に割り当てられる値の型 `T1` を表す型である．PyChoral における `At[T1,R]` は Python の基本型 `T1` 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もち，参加者の人数で継承するクラスを指定する．`At`,`Choreograhy2`,`Choreograhy3` の型パラメータはジェネリック型をとり，コレオグラフィでは任意の型をとれるプレースホルダとして機能する．

```
class Choreograhy2(Generic[R1,R2]):
class Choreograhy3(Generic[R1,R2,R3]):
class At(Generic[T1,R],T1):
```

式の拡張

Python のすべての式が属する型クラスである object 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可能となる．ここで，`Exp` の型が `T1` であれば `Exp@R()` の型は `At[T1,R]` と解析される．

```
class object:
    def __matmul__(self:Self, _:R1) -> At[Self,R1]: ...
```

@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．

Channel,SymChannel は 2 者間の通信をするメソッドをもつ型クラスである．Channel は型パラメータ R1 から R2 へ com メソッドか select メソッドを使って通信するクラスである．SymChannel は@overload デコレータを使用してオーバーロードを定義する．これにより，R1 と R2 との双方向通信を実現する．com は送信者の情報を含む任意の型 At[T1,R1] をもつ値 msg を受信者に送るメソッドである．select は送信者がもつ条件分岐の結果を示す列挙型の値を受信者へ送るメソッドである．

```
class Channel(Generic[T1,R1,R2]):
    def com(self,msg:At[T1,R1]) -> At[T1,R2]:
    def select(self,msg:At[T1,R1]) -> At[T1,R2]:

class SymChannel(Generic[T1,R1,R2]):
    @overload
    def com(self,msg:At[T1,R1]) -> At[T1,R2]:
    @overload
    def com(self,msg:At[T1,R2]) -> At[T1,R1]:
    ...
```

図 4.2 は PyChoral プログラムから各参加者の Python プログラムが生成されるプロセスである．

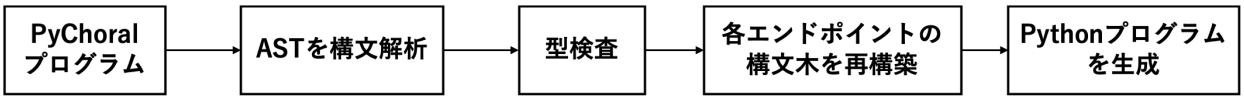


図 4.2　PyChoral プロセス

PyChoral で記述されたプログラムは一度抽象構文木に変換されて保存される．この構文木に対して型検査を行った後に，エンドポイント射影により各参加者の構文木を再構築する．この構文木をもとに各参加者の Python プログラムが生成される．Mypy は，Python の標準的なライブラリや型情報を提供するリポジトリである typeshed を参照して型検査を行う．本研究では，typeshed の中に上記で述べた At や Choreograhy2，Choreograhy3，@演算子などの定義を追加した型検査器で参加者の情報を得る．

式の型情報あるいは変数の型注釈に含まれる参加者の情報は関数 rolesOf(),rolesOf_t() によって取得する．Code 4.1 は関数 rolesOf の実装コードである．

```
1  def rolesOf(e:Expression, typeChecker:TypeChecker) -> list[str]:
2      t0 = get_type(typeChecker, e)
3      if isinstance(t0,mypy.types.ProperType):
4          if get_typename(t0) == "At":
5              return [get_typearg(t0,1)]
6          elif get_typename(t0) == "Channel":
7              return [get_typearg(t0,1),get_typearg(t0,2)]
8          elif get_typename(t0) == "SymChannel":
9              return [get_typearg(t0,1),get_typearg(t0,2)]
10         else:
11             return []
```

Code 4.1　rolesOf

`rolesOf()` は引数に式 (e) と型検査器 (typechecker) をとり，取得した参加者の情報を文字列のリストとして返す関数である．Typechecker は Mypy が型検査を行うための型クラスである．式 e の型情報は 2 行目にある get_type 関数によって得る．4-10 行目では得られた型情報に対して場合分けをしている．4,6,8 行目にある get_typename は型クラスの名前を文字列として返す関数である．例えば `At[int,A]` という型を get_typename に渡すと`"At"`が返ってくる．5,7,9 行目にある get_typearg は型クラスのパラメータから指定されたリストの要素を get_typename によって文字列で返す関数である．例えば `Channel[str,A,B]` という型を get_typearg に渡すと `["A","B"]` が返ってくる．`rolesOf_t()` は型注釈としてプログラムに書かれている型情報から参加者名を取り出す関数である．`rolesOf()`,`rolesOf_t()` は受け取る引数は違うが，動作は同じである．

射影後に再構築される構文木は新たなデータ構造を取り，それに従った Python プログラムが生成される．クラス定義 (ClassDef)，関数定義 (FuncDef) および PyChoral の構文に存在する文 (Stm) は射影後，独自のデータ構造で生成される．新しく定義したデータ構造は Python のデータ構造と同じだが，PyChoral プログラムの式はエンドポイント射影により文字列に変換されているため，生成されるコンストラクタの型が異なる．マージや正規化は新しく定義したデータ構造を利用して行う．

親クラス Stmt は抽象的な構文木であり，クラス継承を使って AST の具体的なノードを子クラス (Block,ClassDef,...) として定義している．新しく定義したデータ構造は Python のデータ構造と同様にコンストラクタを生成する．エンドポイント射影，マージ，正規化を施した構文木は関数 stmt_to_string によって文字列に変換される．この文字列を新しく生成される各参加者のファイルに印字することで Python プログラムを生成している．Code 4.2 は新しく定義したデータ構造および印字関数 stmt_to_string の一部である．

```
1  class Stmt:
2    pass
3
4  class Pass(Stmt):
5    pass
6  class Return(Stmt):
7      def __init__(self, exp:str):
8          self.expr = exp
9  class ClassDef(Stmt):
10     def __init__(self,name:str,rolename:str,base_type_vars:list[str],defs:Block):
11         self.name = name
12         self.rolename = rolename
13         self.base_type_vars = base_type_vars
14         self.defs = defs
15 ...
16
17 def stmt_to_string(s:Stmt,indent:int) -> str:
18     if isinstance(s,Pass):
19             return ' '*indent + 'pass'
20     elif isinstance(s,Return):
21         return ' '*indent + 'return ' + s.expr
```

```
22      elif isinstance(s,ClassDef):
23          return ' '*indent + 'class ' + s.name + '(' + ','.join(s.base_type_vars) +
                ')':\n' + stmt_to_string(s.defs,indent+4) + '\n'
24      ...
```

Code 4.2　data.py

18,20,22 行目に出てくる関数 `isinstance(object,T)` は，Python に標準で備わっている関数である．この関数は，指定されたクラスまたは型 `T` に `object` が属しているかどうかを判定する．本研究では，関数 `isinstance` をクラスまたは型の制限に使用している．関数 `stmt_to_string` は引数に構文木 `s` とインデントの深さを調整するための値 `indent` を引数にとり，文字列を返す関数である．生成された各参加者のファイルへ正常な Python プログラムとして印字するために，改行やインデントを考慮している．

各参加者のプログラムの実行には Python の標準ライブラリに含まれる multiprocessing モジュールの機能を利用する．multiprocessing モジュールが提供する Process クラス，Pipe を使用することで複数のプログラムの並列実行ができる．Code 4.3 は参加者が 2 人のコレオグラフィを実行するランタイムである．参加者 A,B の異なるプロセスを起動し，それぞれが connect 関数を呼び出す．Pipe を使用してプロセス間通信を行い，各プロセスは異なる関数 run_A,run_B を呼び出す．

```
1  parent_pipe = None
2
3  def start_processes(f_A, f_B):
4      pipe_A, pipe_B = Pipe()
5      pa = Process(target=connect, args=(f_A,pipe_A,))
6      pb = Process(target=connect, args=(f_B,pipe_B,))
7      pa.start()
8      pb.start()
9      pa.join()
10     pb.join()
11
12 def connect(f,pipe):
13     global parent_pipe
14     parent_pipe = pipe
15     f()
```

Code 4.3　runtime

1 行目の `parent_pipe` はプロセス間でデータをやり取りするための Pipe オブジェクトを格納するためのグローバル変数である．3-10 行目では，関数 start_process によって 2 つのプロセスを作成し，それぞれ異なる関数 f_A および f_B を実行させる．各プロセスは Pipe オブジェクトを生成し，関数 connect に渡す．start メソッドで関数 f_A,f_B を実行し，join メソッドでプロセスの終了を待機する．11-14 行目は Pipe を繋ぐ関数 connect である．Pipe オブジェクトをグローバル変数 `parent_pipe` に設定する．これにより，関数 f 内で `parent_pipe` を使用してプロセス間の通信を行うことができる．

第 5 章　PyChoral を利用したアプリケーションによる評価

Code 5.1 は Choral のプログラム例の一つである Parallel MergeSort [1] を PyChoral で実装したものである．これは，PyChoral を用いた並列アルゴリズムを示し，通信の参加者のインスタンス化が分散ロジックを記述する際にどれだけ Python プログラマにとって役に立つのかを示すものである．

```
1  class Mergesort(Ch3[M,S1,S2]):
2      def __init__(self, m, s1, s2, chMS1, chS1S2, chS2M):
3          self.m:Type[M] = m
4          self.s1:Type[S1] = s1
5          self.s2:Type[S2] = s2
6          self.chMS1:SymChannel[object,M,S1] = chMS1
7          self.chS1S2:SymChannel[object,S1,S2] = chS1S2
8          self.chS2M:SymChannel[object,S2,M] = chS2M
9
10     def sort(self,a:At[list[int],M]) -> At[list[int],M]:
11         if len(a)@self.m() > 1@self.m():
12             self.chMS1.select(MChoice.L@self.m())
13             self.chS2M.select(MChoice.L@self.m())
14             mb = Mergesort[S1,S2,M](self.s1,self.s2,self.m,self.chS1S2,self.
                   chS2M,self.chMS1)
15             mc = Mergesort[S2,M,S1](self.s2,self.m,self.s1,self.chS2M,self.
                   chMS1,self.chS1S2)
16             pivot:At[float,M] = float(floor(len(a)/2@self.m())@self.m())@self.m()
17             b:list[int] = a[0:int(pivot)]
18             lhs:At[list[int],S1] = mb.sort(self.chMS1.com(b@self.m()))
19             c:list[int] = a[int(pivot):len(a)]
20             rhs:At[list[int],S2] = mc.sort(self.chS2M.com(c@self.m()))
21             return self.merge(self.chMS1.com(lhs),self.chMS1.com(rhs))
22         else:
23             self.chMS1.select(MChoice.R@self.m())
24             self.chS2M.select(MChoice.R@self.m())
25             return a
26
27     def merge(self,lhs:At[list[int],S1],rhs:At[list[int],S2]) -> At[list[int],M]:
28         ...
```

Code 5.1　MergeSort.py

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．2-8 行目ではコンストラクタを生成している．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．10-25 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．11-21 行目は分割する場合である．16 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．14,15 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 21 行目にあるメソッド merge の呼び出しによって結合される．27 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．PyChoral で記述した MergeSort のコードは参加者の型情報により，Python 言語に類似したプログラム一つで，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．

第 6 章　まとめ，今後の課題

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．PyChoral におけるエンドポイント射影の理論により PyChoral プログラム中の式は文字列として，その他は抽象クラス Stmt を親クラスとした新しいデータ構造として射影される．射影された新たな構文木は生成される各参加者のファイルに印字され，Python プログラムとして実行可能なコードで出力される．生成された各参加者の Python プログラムはコレオグラフィに従っているため，デッドロック等の並行性に起因するエラーが生じないことが理論的に保証されている．また，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．これにより，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・分散プログラミングができるようになる．

生成される Python プログラムのデッドロックフリー性を証明することと，Python 言語が多く使用されている IoT や機械学習を扱うプロジェクトに PyChoral を応用して更なる有用性を示すことが今後の課題である．

参考文献

[1] Choral example. `https://github.com/choral-lang/examples/tree/master`.

[2] mypy. `https://mypy.readthedocs.io/en/stable/`.

[3] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. A formal theory of choreographic programming. *J. Autom. Reason.*, Vol. 67, No. 2, p. 21, 2023.

[4] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. Choreographies as objects. *CoRR*, Vol. abs/2005.09520, , 2020.

[5] Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, Vol. 3, No. POPL, pp. 30:1–30:29, 2019.

付録 A　Projection, Merging, Normalizer の定義

A.1　Projection to Python

$$(Class)\quad (\!|\mathsf{class}\ id\ (Ch[\overline{R}], \overline{Exp}) : \overline{Stm}|\!)^A = \begin{cases} \mathsf{class}\ id\_A\ (\overline{(\!|Exp|\!)^A}) : \overline{(\!|Stm|\!)^A} & \text{if}\quad A \in \overline{R} \\ \text{absent} & \text{if}\quad A \notin \overline{R} \end{cases}$$

$$(Func)\quad (\!|\mathsf{def}\ id\ (\overline{id : TE}) : \overline{Stm}|\!)^A = \mathsf{def}\ id\ (\overline{id}) : \overline{(\!|Stm|\!)^A}$$

$$(Exp)\quad (\!|lit@(\overline{R()}) : \tau|\!)^A = \begin{cases} lit & \text{if}\quad A \in \overline{R} \\ \texttt{Unit.id} & \text{otherwise} \end{cases}$$

$$(\!|Exp.id : \tau|\!)^A = \begin{cases} (\!|Exp|\!)^A.id & \text{if}\quad A \in \text{rolesOf}(Exp.id) \\ \text{absent} & \text{otherwise} \end{cases}$$

$$(\!|f(\overline{Exp}) : \tau|\!)^A = \begin{cases} f(\overline{(\!|Exp|\!)^A}) & \text{if}\quad A \in \text{rolesOf}(f(\overline{Exp})) \\ \texttt{Unit.id}(f(\overline{(\!|Exp|\!)^A})) & \text{if}\quad A \in \text{rolesOf}(\overline{Exp}) \wedge A \notin \text{rolesOf}(f(\overline{Exp})) \\ \texttt{Unit.id}(\overline{(\!|Exp|\!)^A}) & \text{otherwise} \end{cases}$$

$$(\!|Exp_1.f(\overline{Exp_2}) : \tau|\!)^A = \begin{cases} (\!|Exp_1|\!)^A.f(\overline{(\!|Exp_2|\!)^A}) & \\ \qquad \text{if}\quad A \in \text{rolesOf}(Exp_1) \wedge A \in \text{rolesOf}(\overline{Exp_2}) & \\ \qquad\qquad \wedge A \in \text{rolesOf}(Exp_1.f(\overline{Exp_2})) & \\ \texttt{Unit.id}((\!|Exp_1|\!)^A.f(\overline{(\!|Exp_2|\!)^A})) & \\ \qquad \text{if}\quad A \in \text{rolesOf}(Exp_1) \wedge A \notin \text{rolesOf}(Exp_1.f(\overline{Exp_2})) & \\ \texttt{Unit.id}((\!|Exp_1|\!)^A, \overline{(\!|Exp_2|\!)^A}) \qquad \text{otherwise} & \end{cases}$$

$$(\!|C[\overline{R}](\overline{Exp}) : \tau|\!)^A = \begin{cases} (\!|C[\overline{R}]|\!)^A(\overline{(\!|Exp|\!)^A}) & A \in \overline{R} \\ \texttt{Unit.id}(\overline{(\!|Exp|\!)^A}) & \text{otherwise} \end{cases}$$

$$(\!|Exp_1\ BinOp\ Exp_2|\!)^A = (\!|Exp_1|\!)^A\ BinOp\ (\!|Exp_2|\!)^A$$

$$\text{rolesOf}(\_ : \tau@(\overline{R()})) = \overline{R}$$

$$\text{rolesOf}(\overline{Exp}) = \bigcup_i \text{rolesOf}(Exp_i)$$

$$(\!|\overline{Exp}|\!)^A = Exp'_1, Exp'_2, \cdots, Exp'_n \quad \text{where}\quad Exp'_i = (\!|Exp_i|\!)^A$$

(Stm)　$(\!|\textbf{pass}|\!)^A = \textbf{pass}$

$(\!|\textbf{return}\ Exp;|\!)^A = \textbf{return}\ (\!|Exp|\!)^A$

$$(\!|Exp; \overline{Stm}|\!)^A = \begin{cases} \textbf{match}\ (\!|Exp|\!)^A : \\ \quad \textbf{case}\ id_2 : (\!|\overline{Stm}|\!)^A; & \text{if}\ \ Exp = Exp'.\texttt{select}(id_1\texttt{@}A.id_2) : \texttt{Enum@}A \\ \quad \textbf{case}\ \_ : \text{assert False}; \\ (\!|Exp|\!)^A; (\!|\overline{Stm}|\!)^A & \text{otherwise} \end{cases}$$

$$(\!|id : TE = Exp\ ; \overline{Stm}|\!)^A = \begin{cases} id = (\!|Exp|\!)^A; (\!|\overline{Stm}|\!)^A & \text{if}\ \ A \in \text{rolesOf}(TE) \\ (\!|Exp|\!)^A; (\!|\overline{Stm}|\!)^A & \text{otherwise} \end{cases}$$

$(\!|Exp_1\ AsgOp\ Exp_2\ ; \overline{Stm}|\!)^A = (\!|Exp_1|\!)^A AsgOp\ (\!|Exp_2|\!)^A\ ; (\!|\overline{Stm}|\!)^A$

$$(\!|\textbf{if}\ Exp : Stm_1\ ; \textbf{else} : Stm_2\ ; \overline{Stm}|\!)^A = \begin{cases} \textbf{if}\ (\!|Exp|\!)^A : (\!|Stm_1|\!)^A\ ; \textbf{else} : (\!|Stm_2|\!)^A\ ; (\!|\overline{Stm}|\!)^A & \text{if}\ \ \text{rolesOf}(Exp) = A \\ (\!|Exp|\!)^A\ ; [\![(\!|Stm_1|\!)^A]\!] \sqcup [\![(\!|Stm_2|\!)^A]\!]\ ; (\!|\overline{Stm}|\!)^A & \text{otherwise} \end{cases}$$

$(\!|\textbf{raise}\ Exp|\!)^A = \textbf{raise}\ (\!|Exp|\!)^A$

$(\!|\textbf{assert}\ Exp|\!)^A = \textbf{assert}\ (\!|Exp|\!)^A$

$(\!|\overline{Stm}|\!)^A = Stm'_1, Stm'_2, \cdots, Stm'_n$　where　$Stm'_i = (\!|Stm_i|\!)^A$

A.2　Merging

Statement

$\dot{\sqcup}\ \overline{Stm} = \dot{\sqcup}\ (Stm_1, \cdots, Stm_n) = [\![Stm_1]\!] \sqcup \cdots \sqcup [\![Stm_n]\!]$

$\textbf{return}\ Exp \sqcup \textbf{return}\ Exp' = \textbf{return}\ Exp \sqcup Exp'$

$\textbf{raise}\ Exp \sqcup \textbf{raise}\ Exp' = \textbf{raise}\ Exp \sqcup Exp'$

$$(Exp_1\ AsgOp\ Exp_2; \overline{Stm}) \sqcup (Exp'_1\ AsgOp\ Exp'_2; \overline{Stm'}) \\ = (Exp_1 \sqcup Exp'_1)\ AsgOp\ (Exp_2 \sqcup Exp'_2); (\overline{Stm} \sqcup \overline{Stm'})$$

$(Exp; \overline{Stm}) \sqcup (Exp'; \overline{Stm'}) = (Exp \sqcup Exp'); (\overline{Stm} \sqcup \overline{Stm'})$

$$
\begin{array}{l}
\textbf{if } Exp_1: \\
\quad Stm_1; \\
\cdots \\
\textbf{elif } Exp_n: \\
\quad Stm_n; \\
\textbf{else}: \\
\quad Stm_e; \\
\overline{Stm}
\end{array}
\sqcup
\begin{array}{l}
\textbf{if } Exp'_1: \\
\quad Stm'_1 \\
\cdots \\
\textbf{elif } Exp'_n: \\
\quad Stm'_n \\
\textbf{else}: \\
\quad Stm'_e \\
\overline{Stm'}
\end{array}
=
\begin{array}{l}
\textbf{if } Exp_1 \sqcup Exp'_1: \\
\quad Stm_1 \sqcup Stm'_1 \\
\cdots \\
\textbf{elif } Exp_n \sqcup Exp'_n: \\
\quad Stm_n \sqcup Stm'_n \\
\textbf{else}: \\
\quad Stm_e \sqcup Stm'_e \\
\overline{Stm} \sqcup \overline{Stm'}
\end{array}
$$

$$
\begin{array}{l}
\textbf{match } Exp: \\
\quad \textbf{case } id_a: Stm'_a; \\
\quad \cdots \\
\quad \textbf{case } id_x: Stm'_x; \\
\quad \textbf{case } id_y: Stm'_y; \\
\quad \textbf{case } \_: Stm'_{ex}; \\
\overline{Stm}
\end{array}
\sqcup
\begin{array}{l}
\textbf{match } Exp': \\
\quad \textbf{case } id_a: Stm''_a; \\
\quad \cdots \\
\quad \textbf{case } id_x: Stm''_x; \\
\quad \textbf{case } id_z: Stm'_z; \\
\quad \textbf{case } \_: Stm''_{ex}; \\
\overline{Stm'}
\end{array}
=
\begin{array}{l}
\textbf{match } Exp \sqcup Exp': \\
\quad \textbf{case } id_a: Stm'_a \sqcup Stm''_a; \\
\quad \cdots \\
\quad \textbf{case } id_x: Stm'_x \sqcup Stm''_x; \\
\quad \textbf{case } id_y: Stm'_y; \\
\quad \textbf{case } id_z: Stm'_z; \\
\quad \textbf{case } \_: Stm'_{ex} \sqcup Stm''_{ex}; \\
\overline{Stm} \sqcup \overline{Stm'}
\end{array}
$$

$$
\overline{Stm} \sqcup \overline{Stm'} = [\![Stm_1]\!] \sqcup [\![Stm'_1]\!]\ ,\ [\![Stm_2]\!] \sqcup [\![Stm'_2]\!]\ ,\ \cdots\ , [\![Stm_n]\!] \sqcup [\![Stm'_n]\!]
$$

Expression

$$
Exp \sqcup Exp' = \begin{cases} Exp & \text{if } Exp = Exp' \\ \text{error} & \text{if } Exp \neq Exp' \end{cases}
$$

A.3　Normalizer

Statements

$$
[\![\textbf{pass}]\!] = \textbf{pass}
$$

$$
[\![\textbf{return } Exp]\!] = \textbf{return } [\![Exp]\!]
$$

$$
[\![Exp; \overline{Stm}]\!] = \begin{cases} [\![\overline{Stm}]\!] & \text{if } \textsc{noop}([\![Exp]\!]) = [blank] \\ [\![Exp]\!]; [\![\overline{Stm}]\!] & \text{otherwise} \end{cases}
$$

$$[\![Exp_1\ AsgOp\ Exp_2; \overline{Stm}]\!] = \begin{cases} [\![Exp_1]\!]; [\![\overline{Stm}]\!] & \text{if } \textsc{noop}([\![Exp_2]\!]) = [blank] \\ [\![Exp_2]\!]; [\![\overline{Stm}]\!] & \text{if } \textsc{noop}([\![Exp_1]\!]) = [blank] \\ [\![\overline{Stm}]\!] & \text{if } \textsc{noop}([\![Exp_1]\!], [\![Exp_2]\!]) = [blank] \\ [\![Exp_1]\!] AsgOp [\![Exp_2]\!]; [\![\overline{Stm}]\!] & \text{otherwise} \end{cases}$$

$$[\![\textbf{if}\ Exp : Stm_1\ ;\ \textbf{else} : Stm_2\ ;\ \overline{Stm}]\!] = \textbf{if}\ [\![Exp]\!] : [\![Stm_1]\!]\ ;\ \textbf{else} : [\![Stm_2]\!]\ ;\ [\![\overline{Stm}]\!]$$

$$[\![\textbf{match}\ Exp :\ \overline{\textbf{case}\ id : Stm}\ ;\ \overline{Stm}]\!] = \textbf{match}\ [\![Exp]\!] :\ \overline{\textbf{case}\ id : [\![Stm]\!]}\ ;\ [\![\overline{Stm}]\!]$$

Expressions

$$\textsc{noop}(Exp) = \begin{cases} [blank] & \text{if } Exp \in \{\texttt{Unit.id, None}\} \\ Exp & \text{otherwise} \end{cases}$$

$$[\![None]\!] = None \quad [\![id]\!] = id$$

付録 B　射影後のデータ構造

```
class Stmt:
    pass

class Block(Stmt):
    def __init__(self, body:list[Stmt], indent:int):
        self.body = body
        self.indent = indent

class Pass(Stmt):
    pass

class Return(Stmt):
    def __init__(self, exp:str):
        self.expr = exp

class Raise(Stmt):
    def __init__(self, expr:str):
        self.expr = expr

class Assert(Stmt):
    def __init__(self, expr:str):
        self.expr = expr

class Es(Stmt):
    def __init__(self, exp:str):
        self.expr = exp

class Es2(Stmt):
    def __init__(self, exp:str, stmt:list[Stmt]):
        self.expr = exp
        self.stmt = stmt

class Asg(Stmt):
    def __init__(self, lv:str, rv:str, type:Type):
        self.lvalues = lv
        self.rvalue = rv
        self.type = type

class OpAsg(Stmt):
    def __init__(self, lv:str, rv:str, op:str):
        self.lvalue = lv
```

```
            self.rvalue = rv
            self.op = op

    class If(Stmt):
        def __init__(self, exp:str, body:Block, else_body:Block):
            self.expr = exp
            self.body = body
            self.else_body = else_body

    class Match(Stmt):
        def __init__(self, sub:str, pat:list[str], bodies:list[Block]):
            self.subject = sub
            self.patterns = pat
            self.bodies = bodies

    class FuncDef(Stmt):
        def __init__(self, name:str, arguments:list[str], body:Block):
            self.name = name
            self.arguments = arguments
            self.body = body

    class ClassDef(Stmt):
        def __init__(self, name:str, rolename:str, base_type_vars:list[str], defs:
            Block):
            self.name = name
            self.rolename = rolename
            self.base_type_vars = base_type_vars
            self.defs = defs

    class Import(Stmt):
        def __init__(self, ids:list[str]):
            self.ids = ids

    class ImportFrom(Stmt):
        def __init__(self, id:str, relative:int, names:list[str]):
            self.id = id
            self.relative = relative
            self.names = names

    class ImportAll(Stmt):
        def __init__(self, id:str, relative:int):
            self.id = id
            self.relative = relative
```

付録 C　3 者間のコレオグラフィのランタイム

```
parent_pipeA = None
parent_pipeB = None
parent_pipeC = None

roledict = {"A":parent_pipeA,"B":parent_pipeB,"C":parent_pipeC}

def start_processes(f_A, f_B, f_C):
    pipe_AtoB, pipe_BtoA = Pipe()
    pipe_BtoC, pipe_CtoB = Pipe()
    pipe_CtoA, pipe_AtoC = Pipe()
    pa = Process(target=connectA, args=(f_A,pipe_AtoB,pipe_AtoC,))
    pb = Process(target=connectB, args=(f_B,pipe_BtoC,pipe_BtoA,))
    pc = Process(target=connectC, args=(f_C,pipe_CtoA,pipe_CtoB,))
    pa.start()
    pb.start()
    pc.start()
    pa.join()
    pb.join()
    pc.join()

def connectA(f,pipe_AtoB,pipe_AtoC):
    global parent_pipeA
    global roledict
    roledict = {"A":None,"B":pipe_AtoB,"C":pipe_AtoC}
    print("pipe_AtoB,pipe_AtoC are connected")
    f()

def connectB(f,pipe_BtoC,pipe_BtoA):
    global parent_pipeB
    global roledict
    roledict = {"A":pipe_BtoA,"B":None,"C":pipe_BtoC}
    print("pipe_BtoC,pipe_BtoA are connected")
    f()

def connectC(f,pipe_CtoA,pipe_CtoB):
    global parent_pipeC
    global roledict
    roledict = {"A":pipe_CtoA,"B":pipe_CtoB,"C":None}
    print("pipe_CtoA,pipe_CtoB are connected")
    f()
```