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

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 [4] により，自動的に抽出される．

エンドポイント射影とは，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．

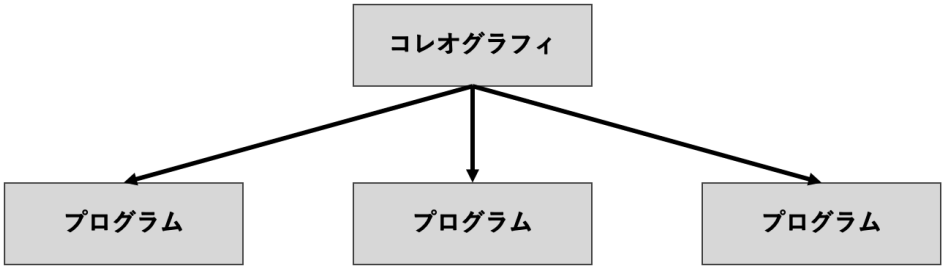


図 2.1　エンドポイント射影

2.2　Choral

Choral [1] 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') # No error
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．

```
# A
div_a = Divide2_A()
div_a.divide_by_two(n,0)
# 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 APython プログラム

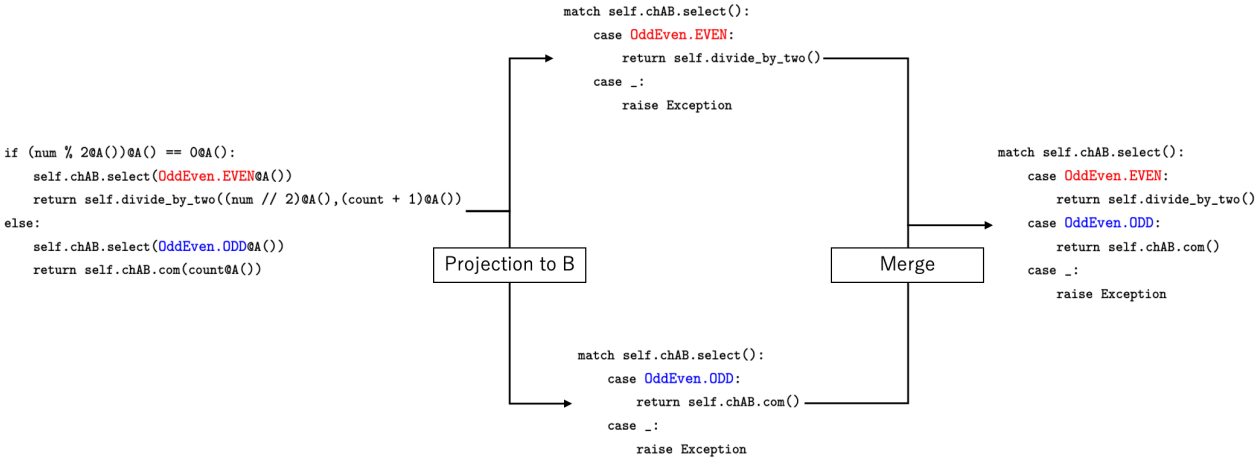
```
1  class Divide2_B():
2      def __init__(self):
3          self.chAB = Channel[object,A,B]('A','B')
4      def divide_by_two(self):
5          match self.chAB.select():
6              case OddEven.EVEN:
7                  return self.divide_by_two()
8              case OddEven.ODD:
9                  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，その設計をもとに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

PyChoral の構文は Python と同一である．図 4.1 は Python の構文定義である．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	$::= \mathbf{@}id$
Type Annotation	τ	$::= Exp : \tau \mid id : \tau$
Literals	lit	$::=$ None $\mid$ True $\mid$ False $\mid$ "a" $\mid \ldots \mid 1 \mid \ldots$
Expression	Exp	$::= lit\mathbf{@}\overline{A()} \mid Exp.id \mid f(\overline{Exp}) \mid Exp.f(\overline{Exp}) \mid C[\overline{id}](\overline{Exp})$ $\mid Exp_1\ BinOp\ Exp_2$
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．

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

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．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，Python のプログラムとして生成される．この例の場合，$\text{rolesOf}(Exp_1) = \{\text{A,B}\}$, $\text{rolesOf}(\overline{Exp_2}) = \{\text{A}\}$, $\text{rolesOf}(Exp_1.f(\overline{Exp_2})) = \{\text{B}\}$ である．

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

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

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 ($\sqcup$) 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]:
    @overload
    def select(self,msg:At[T1,R1]) -> At[T1,R2]:
    @overload
    def select(self,msg:At[T1,R2]) -> At[T1,R1]:
```

PyChoral では Mypy を活用した型検査のプロセスを改造し，型検査を行なった後に各参加者の抽象構文木を再構築する．射影後に再構築される構文木は新しいデータ構造を取り，それに従った Python プログラムが生成される．

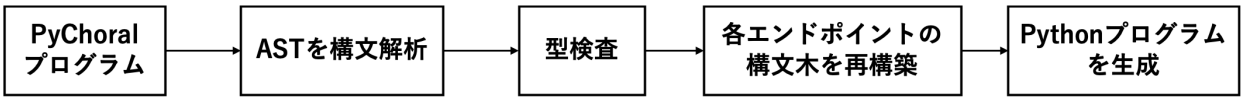


図 4.2　PyChoral プロセス

＜ Mypy の改造＞

PyChoral は Python ベースの言語のため，Java ベースの Choral を模倣するには静的型付け言語として扱え，コンパイル時にエラーの有無を判別したい．そのため，Mypy を活用して静的に型をつける．ただし，Mypy で型注釈をつければ Choral と同じように動くかというと，そうではない．

まず，Choral は Java のパーサーを改良し，独自のコンパイラを使用して Choral プログラムから Java プログラムを自動導出しているが，本研究では既存の Python パーサーをそのまま活用する．Choral はクラス定義，インターフェース，変数，型に @ で参加者の情報を加えていたが，Mypy の型検査ではこれを認識しない．この問題に対して，本研究では主に 2 つの方法で各参加者の情報をとることにした．

1 つ目の方法として，Mypy が型検査する際に参照する標準ライブラリの typeshed に At，Channel といった新たな型クラスを追加し，変数の型注釈に参加者情報を加えられるようにした．At，Channel の引数はジェネリック型をとっており，任意の型を後から取ることができるプレースホルダーとして機能している．例えば PyChoral のプログラム例 (Code 3.1) の 3 行目にある price:At[int,S] は，変数 price の型が int で参加者 S(Staff) に関連する値ということが分かる．

2 つ目の方法として，Mypy が型検査する際に参照される object クラスに @ のためのメソッド

__matmul__ を追加し，role.py で各参加者に対して適用することで **123@A**() のように **@** をつけてプログラミングしても Mypy の型検査が通るようにした．

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

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

Code 4.1　builtins.pyi

```
class Role: # base class
  pass
class A(Role):
  def __matmul__(self, x): # @ を使えるようにする
      pass
```

Code 4.2　role.py

改造した typeshed を用いて PyChoral 言語で記述したファイルは mypytest.py によって一度 AST に変換して保存する (result)．mypycustom は Mypy 標準ファイル main.py を簡易的にしたファイルである．main.py は構築した AST を保存せずに捨てていたが，mypycustom では AST を保存するように変更してある．この保存した AST に対して，木構造に変換し，その要素に対してプロジェクションを行う．

```
filename = sys.argv[1]
pychoralfile = filename + '.py'
result : mypy.build.BuildResult None = mypycustom.main(
    ['--show-traceback', '--custom-typeshed', './typeshed', pychoralfile])

src = result.graph[filename]
typechecker = src.type_checker()

def get_roles(stm_list:list[mypy.nodes.Statement]) -> str:
    ...
roles = get_roles(src.tree.defs)

for r in roles:
    pro_filename = pychoralfile.replace('.', '_' + r + '.')
    f = open(pro_filename, 'w')
```

```
16    f.write('from pychoral' + str(len(roles)) + ' import *\n')
17    g = open(pro_filename, 'a')
18    for stm in projection.projection_all(src.tree.defs,r,typechecker):
19        data.stmt_to_string(stm,0)
20        g.write(data.stmt_to_string(stm,0))
```

Code 4.3　mypytest.py

filename は射影の実行コマンドから受け取る．src には木構造に変換された PyChoral プログラムが代入されており，typechecker は型検査器である．get_roles は PyChoral プログラム中のクラスから，Choreograhy クラスを元に関係する全ての参加者名を取得する関数である．その後，全ての参加者に対して新たなファイルを作成し，射影した結果を各ファイルに印字していく．

＜射影後のデータ構造＞

Import を含む文，ブロック，クラス定義，関数定義は射影後，独自のデータ構造で生成される．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クラスは Mypy に備わっている標準のデータクラスから射影定義に従って必要なパラメータのみ抽出したものである．射影によって構築された新たなデータクラスは関数 stmt_to_string によって文字列に変換される．この際に' '*indent によって印字されるファイルでのインデントのずれなどにも考慮する．

```
1 class Stmt:
2   pass
3
4 class Pass(Stmt):
5   pass
6
7 class Return(Stmt):
8     def __init__(self, exp:str):
9         self.expr = exp
10 ...
11
12 def stmt_to_string(s:Stmt,indent:int) -> str:
13     if isinstance(s,Pass):
14             return ' '*indent + 'pass'
15     elif isinstance(s,Return):
16         return ' '*indent + 'return ' + s.expr
17     ...
```

Code 4.4　data.py

13,14 行目に出てくる関数 isinstance() はオブジェクトが指定されたクラスまたは型に属しているかどうかを判定する Python の標準関数である．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．＜ projection_all ＞

PyChoral プログラムの射影をするときはまず projection_all が呼ばれる．関数 projection_all は Statement のリスト n，射影される参加者名 r，型チェッカー tc を引数にとり，新しいデータ構造を返す関数

である．

```
1 def projection_all(n:list[Statement],r:str,tc:TypeChecker) -> list[Stmt]:
2     result:list[Stmt] = []
3     for node in n:
4         if isinstance(node,Import) or isinstance(node,ImportFrom) or isinstance(node,
              ImportAll):
5             result += [projection_md(node)]
6         elif isinstance(node,ClassDef):
7             result += [projection_class(node,r,tc)]
8         elif isinstance(node,FuncDef):
9             result += [projection_func(node,r,tc)]
10        elif isinstance(node,Block):
11            result += projection_block(node.body,r,tc)
12        else:
13            result += [projection_stm(node,r,tc)]
14    return help_func.normalize_block(result)
```

Code 4.5　projection_all

isinstance() によって文 (node) の型は import，クラス定義，関数定義，ブロック (文のリスト)，文のいずれかとなり，それぞれの定義によって射影された結果を result に加えていく．射影された結果は最後に正規化 (normalize) される．< projection_exp >

式 (Expression) の射影関数 projection_exp は式 n，射影される参加者名 r，型チェッカー tc を引数にとり，文字列を返す関数である．Code 4.6 は 4.3.2 節で紹介したメソッド呼び出しの射影 $(\!|Exp.f(\overline{Exp}) : \tau|\!)^A$ の一部である．メソッド呼び出しの形が $e.f(\bar{e})$ とすると，Expression → CallExpression のダウンキャストよりパラメータ $e.f$ は n.callee となり，パラメータ $\bar{e}$ は n.args となる．パラメータ $e.f$ を $e.f(\bar{e})$ とすると，Expression → MemberExpression のダウンキャストにより e をオブジェクト，f がメソッド名であると判別している．7 行目ではメソッド呼び出し全体の型に射影される参加者名が含まれるか場合分けをしている．射影される参加者が含まれている場合は射影定義に従って引数にあたる $\bar{e}$ に対して射影を再び行う．全てのパラメータが文字列となれば最後に結合させて返す．

```
1 def projection_exp(n:Expression,r:str,tc:TypeChecker) -> str:
2   ...
3   elif isinstance(n, CallExpr):
4     ...
5     elif isinstance(n.callee, Mypy.nodes.MemberExpr):
6       exp_list_i = []
7       if r in help_func.rolesOf(n,tc): # R in e.f(e')
8         for exp_i in n.args:
9           exp_list_i.append(projection_exp(exp_i ,r,tc))
10        exp_var_i = ','.join(exp_list_i)
11        return projection_exp(n.callee.expr,r,tc) + '.' + n.callee.name + '(' +
              exp_var_i + ')'
12      else:
```

```
13        ...
```

Code 4.6　pro_e.py

< projection_stm, projection_block >文 (Statement) の射影関数 projection_stm は文 s，射影される参加者名 r, 型チェッカー tc を引数にとり, Stmt を返す関数である．ブロックの射影関数 projection_block は Statement のリスト s_list，射影される参加者名 r，型チェッカー tc を引数にとり，Stmt のリストを返す関数である．projection_block では，分岐がある Statement と分岐がない Statement で場合分けされる．分岐がある場合は条件 (4 行目,4.3.2 節 select を参照) を満たしたとき Match 文のデータ構造として値が返される (7 行目)．分岐がない場合はリストの先頭の Statement に対して projection_stm を呼び出し，残りは projection_block で再帰的に呼び出す (11 行目)．

```
1 def projection_block(s_list:list[Statement],r:str,tc:TypeChecker)-> list[Stmt]:
2   ...
3   t = s.expr.accept(tc.expr_checker)
4     if isinstance(t,mypy.types.Instance) and 'enum' in t.type.defn.name and r in
          rolesOf(s.expr,tc) and isinstance(s.expr,mypy.nodes.CallExpr) and isinstance(
          s.expr.callee,mypy.nodes.MemberExpr) and s.expr.callee.name == 'select':
5         if len(s.expr.args) == 1:
6             pro_args = projection_exp(s.expr.args[0],r,tc)
7             return [Match(projection_exp(s.expr,r,tc),[pro_args],[Block(
                  projection_block(s_list[1:],r,tc))])]
8         else:
9           ...
10    else:
11    return [projection_stm(s,r,tc)] + projection_block(s_list[1:],r,tc)
12 def projection_stm(s:Statement,r:str,tc:TypeChecker) -> Stmt:
13   ...
```

Code 4.7　pro_s.py

< projection_class >クラス定義 (ClassDef) の射影関数 projection_class は文 n，射影される参加者名 r，型チェッカー tc を引数にとり， 新しく定義した ClassDef のデータ構造 ClassDef を返す関数である．新しい ClassDef クラス mypy 標準の ClassDef クラスのパラメータに，参加者のパラメータ (rolename) が加わった構造をしている．継承されるクラスのリスト (base_type_vars) の先頭は Ch1,Ch2,Ch3 のいずれかとなっており，それぞれ参加者 1,2,3 者の振舞いが関係するクラスであると明示することで射影の際に rolename を取り出すことができる．クラス定義の入れ子になっている部分は defs に該当し，これは Statement のリストである．クラス定義内に関数定義が出てくる場合は defs の先頭に対して projection_func() を呼び出す (7 行目)．その他の式や文である場合は projection_block を呼び出して，各 Statement に対して射影を行なっていく (9 行目)．

```
1 def projection_class(n:ClassDef,r:str,tc:TypeChecker) -> ClassDef:
2   if 'Ch1' in str(n.base_type_exprs[0]) or 'Ch2' in str(n.base_type_exprs[0]) or '
        Ch3' in str(n.base_type_exprs[0]) and r in str(n.base_type_exprs[0]):
3       exprs:list[str] = []
4       for exp in n.base_type_exprs[1:]:
```

```
5           exprs += [(projection_exp(exp,r,tc))]
6       if type(n.defs.body[0]) == FuncDef:
7           return ClassDef(n.name,r,exprs,projection_func(n.defs.body[0],r,tc))
8       else:
9           return ClassDef(n.name,r,exprs,projection_block(n.defs.body,r,tc))
10  ...
```

Code 4.8　pro_class.py

< projection_func >関数定義 (FuncDef) の射影関数 projection_func は文 n，射影される参加者名 r，型チェッカー tc を引数にとり， 新しく定義した FuncDef のデータ構造 FuncDef を返す関数である．projection_func は主に引数の変数に対する処理を行う．変数の型注釈に対して射影される参加者が関係ある，例えば x:At[str,Client] といった引数を Client に対して射影する場合は変数だけを残し，型注釈は消す．変数の型注釈に対して射影される参加者が関係ない場合は，その変数は引数のリストから削除する．関数定義後の式や文は projection_block で射影を行う (8,10 行目)．

```
1 def projection_func(n:FuncDef, r:str, tc:TypeChecker) -> FuncDef:
2   args:list[str] = []
3   if len(n.arguments) = 0:
4       for arg in n.arguments:
5           if arg.type_annotation is not None and r in str(arg.type_annotation):
6               args.append(arg.variable.name)
7           ...
8       return FuncDef(n.name,args,projection_block(n.body.body,r,tc))
9   else:
10      return FuncDef(n.name,[],projection_block(n.body.body,r,tc))
```

Code 4.9　pro_func.py

< projection_md > PyChoral プログラムの import 文は Python プログラムへそのまま文字列として生成する．

```
1 def projection_md(n:mypy.nodes.Statement) -> Stmt:
2     if isinstance(n,mypy.nodes.Import):
3         return Import(n.ids)
4   ...
```

Code 4.10　pro_md.py

< help_function > help_function はここまで紹介してきた projection の補助をする関数の集合である．rolesOf() は式 (Expression) の型情報から参加者名を取り出す関数である．rolesOf_t() は型注釈から参加者名を取り出す関数である．merge(),merge_block() は 2 つの文あるいは文のリストをマージする関数である．normalize(),normalize_block() は文や文のリストを正規化する関数である．

```
1 def rolesOf(n:Expression, typeChecker:TypeChecker) -> list[str]:
2   ...
3 def rolesOf_t(n:Type None, typeChecker:TypeChecker) -> list[str]:
4   ...
```

```
5 def merge_block(s1:list[Stmt],s2:list[Stmt]) -> list[Stmt]:
6     merged_list:list[Stmt] = []
7     for (t1,t2) in zip(s1,s2):
8         merged_list.append(merge(t1,t2))
9     return merged_list
10
11 def merge(s1:Stmt, s2:Stmt) -> Stmt:
12  ...
13
14 def normalize_block(s_list:list[Stmt]) -> list[Stmt]:
15   ...
16 def normalize(s:Stmt) -> Stmt:
17   ...
```

Code 4.11　help_func.py

第 5 章　PyChoral を利用したアプリケーションによる評価

この節では実装を完成させた後，PyChoral 言語を使用したプログラム例とコンパイル時に生成される各参加者のプログラムにより本研究の有用性を示す．(クイックソート，マージソート，分散認証システム，etc)

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 を実装した．PyChoral プログラムは本研究におけるエンドポイント射影の定義によって各参加者の Python プログラムが生成される．エンドポイント射影理論により 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．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保証されている．これにより，マルチスレッド環境におけるプログラムが単一の言語で記述されるプログラムによって実装可能となり，Python でマルチスレッドプログラムをコーディングするプログラマの手助けとなる．

今後の課題としては以下の点がある．

- 実装の完成
  現段階では PyChoral プログラムを mypytest でコンパイルした時に結果としてターミナルに特定の参加者の Python プログラムが文字列として出力されるが，最終的には各参加者の Python プログラムファイルが新たに生成されるようにする．また，記述したコードのバグを修正する．
- 実装したコンパイラによる PyChoral プログラムを制作する (4.5 節)

参考文献

[1] Choral. https://www.choral-lang.org/.

[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] 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　Definition of Projection, Merging, Normalizer

A.1　Projection to Python

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(\!|\textbf{if}\ Exp : Stm_1\ ; \textbf{else} : Stm_2\ ; \overline{Stm}|\!)^A =$$

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

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

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

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

A.2　Merging

Statement

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

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

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

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

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

```
if Exp1 :                  if Exp'1 :                 if Exp1 ⊔ Exp'1 :
    Stm1;                      Stm'1                      Stm1 ⊔ Stm'1
...                        ...                        ...
elif Expn :          ⊔     elif Exp'n :         =     elif Expn ⊔ Exp'n :
    Stmn;                      Stm'n                      Stmn ⊔ Stm'n
else :                     else :                     else :
    Stme;                      Stm'e                      Stme ⊔ Stm'e
Stm                        Stm'                       Stm ⊔ Stm'
```

```
match Exp :                match Exp' :               match Exp ⊔ Exp' :
  case ida : Stm'a;          case ida : Stm''a;         case ida : Stm'a ⊔ Stm''a;
  ...                        ...                        ...
  case idx : Stm'x;    ⊔     case idx : Stm''x;   =     case idx : Stm'x ⊔ Stm''x;
  case idy : Stm'y;                                     case idy : Stm'y;
                             case idz : Stm'z;          case idz : Stm'z;
  case _ : Stm'ex;           case _ : Stm''ex;          case _ : Stm'ex ⊔ Stm''ex;
Stm                        Stm'                       Stm ⊔ Stm'
```

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

Expression

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

A.3　normalizer

Statements

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

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

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

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

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

Expressions

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