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 X 月 XX 日

目次

第 1 章　はじめに

機械学習や IoT の業界で盛んに使用されている Python は GIL がオプション化されるため，並列処理プログラミングの需要が高まると考える．しかし，一般的に並列処理を用いたプログラムは，動作の並行性に起因するデッドロックなどの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，コレオグラフィがある．

コレオグラフィとは，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する．コレオグラフィに従って各エンドポイント (通信の参加者) のプログラムが生成され，それらはデッドロック等の並行性起因のエラーがないことを保証する．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エンドポイント射影により，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型情報を導出する操作である．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り，エンドポイント射影によって並行性起因のエラーなく実行可能である各エンドポイントの Java プログラムが生成される．

本研究では，先行研究である Java を拡張したコレオグラフィックプログラミング言語 Choral の理論を参考に，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 の実装を行い，マルチスレッドプログラムを Python で書くプログラマにとっての有用性を示す．

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 の概要と使用例について述べる．3 章では本研究で開発したコレオグラフィック言語である PyChoral について述べ，詳細は各節に分けて述べる．3.1 節ではプログラム例とともに PyChoral の概要について述べる．3.2 節では PyChoral の構文とコンパイル時に行われるエンドポイント射影の定義を例と共に述べる．3.3 節ではエンドポイント射影の実装について述べる．4 章では PyChoral を用いたアプリケーションを示し，PyChoral の有用性を示す．5 章で結論と今後の課題について述べる．

第 2 章　前提知識・先行研究

2.1　コレオグラフィックプログラミング

本研究で実装する PyChoral はコレオグラフィックプログラミング言語の一つである．この章では本研究に関する前提知識と先行研究について述べる．

2.1.1　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．図 2.1 はコレオグラフィ及びそれに従った各エンドポイントの例である．まず，客が店員に対してお金 money を払う．商品の価格 price に対して払った額が多い場合は，店員から客へ'thanks' を返し，商品の価格 price に対して払った額が少ない場合は，店員から客へ'not enough' を返を返す．

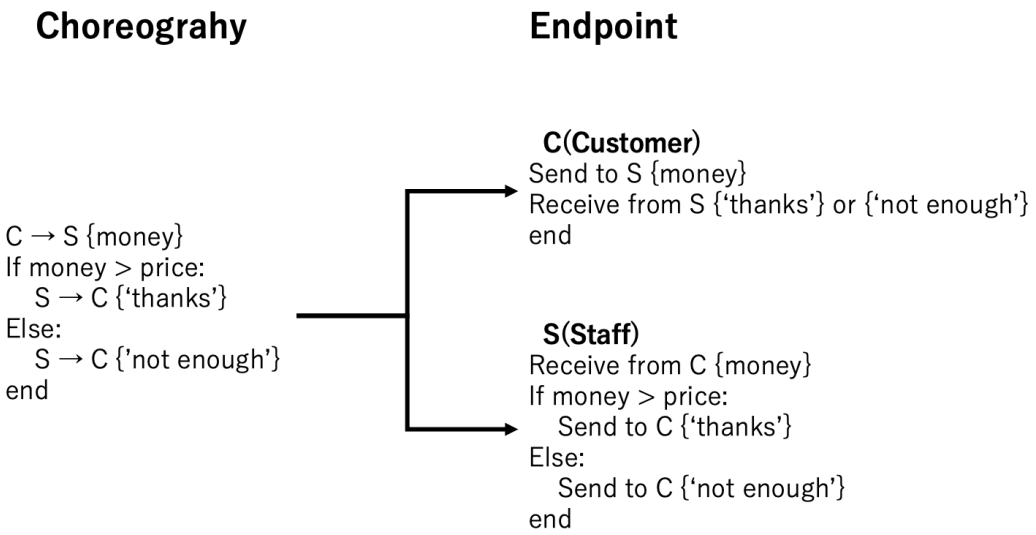

図 2.1　コレオグラフィとエンドポイント (Customer と Staff との会計時の会話)

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分散システム用の実行可能コードに変換する．

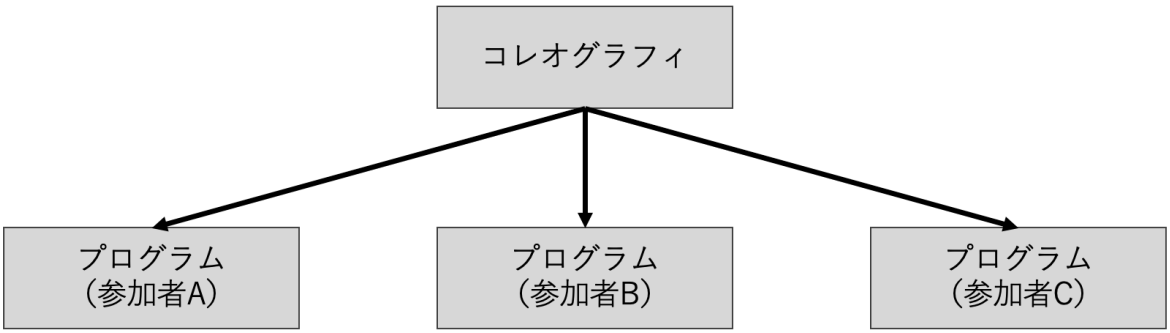

図 2.2　エンドポイント射影 (参加者 3 人)

2.1.2　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．マルチスレッド環境で動作するプログラムでは，データ競合が発生しないようにコーディングする事がプログラマの大きな負担となっていたが，Choral は，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，これら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 $R_1, \cdots, R_n$ が存在する．これにより，Choral プログラムをコンパイルする際に各参加者のプログラムが，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．また，Choral には com メソッドと selection メソッドなるものがある [4].

```
1 interface DiDataChannel@(A,B)<T@C> {
2   <S@D extends T@D> S@B com(S@A m);
3 }
```

code 2.1　データ転送のための基本的な有向チャンネル

code2.1 は com メソッドの最も簡易的に使用しているインターフェースの例である．DiDataChannel は，A と B で抽象化された 2 つの参加者間の有向チャンネルで，型パラメータ T で抽象化された型のデータを A から B に転送するためのインターフェースである．データ転送は com を呼び出すことによって実行される．com は，A に位置する T のサブタイプの任意の値 S@A を取り，S@B を返す.

```
1 interface DiSelectChannel@(A,B) {
2   @SelectionMethod
3   <T@C extends Enum@C<T>> T@B select(T@A m);
4 }
```

code 2.2　selection メソッドの定義

selection メソッドは参加者間で列挙型のインスタンスを送信する際に使用する．selection メソッドは選択を意味し，プロジェクション後は switch 文に変換されて Java プログラムが生成される.

Choral は独自のパーサーにより，Choral プログラムからコンパイラを通して各エンドポイントの Java プログラムが自動導出される．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.

2.2　Mypy

Python は動的型付け言語で実行時にのみエラーが表示されるので，コンパイル時に型に対するエラーは表示されない．Mypy[2] は Python の静的型検査器であり，既存の Python コードに型アノテーションを追加することで，型が誤っていると警告を出すようになる．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．

```
1 def greeting(name):
2   return 'Hello ' + name
3
4 greeting(123)
5 greeting(b'Alice')
```

code 2.3　型注釈のない Python コード

```
1 def greeting(name: str) -> str:
2   return 'Hello ' + name
3
4 greeting('World') # No error
5 greeting(3)
6 # Argument 1 to 'greeting' has incompatible type 'int'; expected 'str'
7 greeting(b'Alice')
8 # Argument 1 to 'greeting' has incompatible type 'bytes'; expected 'str'
```

code 2.4　型注釈のある Python コード

本研究では Mypy を別の Python アプリケーションに統合するために，Python プログラムに Mypy.api をインポートして型検査を行う．図 2.3 は通常の Mypy を使用した際の型検査のプロセスである．

図 2.3　Mypy プロセス

- 標準的に備わっている typeshed の話はいるのか
- トラバース方法について

第 3 章　PyChoral

3.1　PyChoral の概要

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 でマルチパーティなプログラムを記述し，コンパイルすることで型エラーがない場合は，各参加者の Python プログラムコードがコレオグラフィに従った形で自動導出することができる．

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エンドポイントの AST を再構築する．射影後に再構築される AST 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 (詳細は 4.4.2 節).

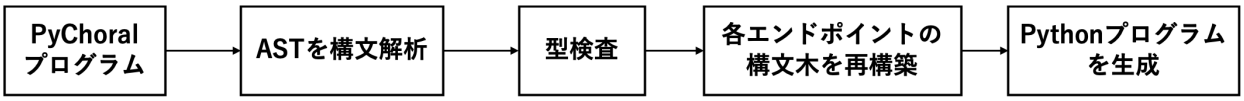

図 3.1　PyChoral プロセス

コード 3.1 は図 2.1 のコレオグラフィを PyChoral で記述したプログラムである．継承クラス Ch2 はクラス内で 2 者 (ここでは C,S) の振舞いが関係することを示す．関数 check の引数 price,money の型注釈 At[object,R] はその型を持つ変数の値の型と関連する参加者の情報を併せ持つ型クラスである．ch_CS,ch_SC の型注釈 Channel[object,R1,R2] は 2 者間の queue を表し，転送される値，送信者，受信者の型を持つ型クラスである．e@R() は参加者 R の値 e であることを示す．com は送信者の値を受信者に転送するメソッドである．

```
1  class Check(Ch2[S,C]):
2      def __init__(self):
3          self.ch_CS : Channel[int,C,S] = Channel[int,C,S]('C','S')
4          self.ch_SC : Channel[str,S,C] = Channel[str,S,C]('S','C')
5
6      def check(self,price:At[int,S],money:At[int,C]) -> At[str,C]:
7            payment = self.ch_CS.com(money)
8            if (payment > price):
9                return self.ch_SC.com('thanks'@S())
10           else:
11               return self.ch_SC.com('not enough'@S())
```

code 3.1　PyChoral

射影後の参加者 C,S のプログラムは射影の定義に従い，それぞれ関係ある式，文が射影されて生成される．生成された Python プログラムは型注釈がない．PyChoral プログラムの型注釈において射影対象でない参加者が関連する値は消える．

```
1  # Customer
2  class Check_C():
3      def __init__(self):
4          self.ch_CS = Channel[int,C,S]('C','S')
5          self.ch_SC = Channel[str,S,C]('S','C')
```

```
6       def check(self,money):
7           Unit.id(self.ch_CS.com(money))
8           return self.ch_SC.com()
9
10  # Staff
11  class Check_S():
12      def __init__(self):
13          self.ch_CS = Channel[int,C,S]('C','S')
14          self.ch_SC = Channel[str,S,C]('S','C')
15      def check(self,price):
16          payment = self.ch_CS.com()
17          if payment>price:
18              return Unit.id(self.ch_SC.com('thanks'))
19          else:
20              return Unit.id(self.ch_SC.com('not enough'))
```

code 3.2　generated Python

3.2　PyChoral の構文と射影定義

この節では本研究で実装した PyChoral プログラムの構文及びコンパイル時のエンドポイント射影の定義を示す．まず，PyChoral の構文を示し，Python との違いについて述べる．次に，PyChoral プログラムから Python プログラムが自動導出されるためのエンドポイント射影の定義を示す．

3.2.1　syntax

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

図 3.2　Syntax of PyChoral

PyChoral のクラス定義は，クラス継承の第一引数に Choreography クラスを継承させる．Choreography クラスは以下のように定義され，どの参加者が関係するか分かるようなクラスとなっている．Ch クラス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．継承される Choreograhy クラスは，子クラスに関わる参加者の数によって決定される．2 者 (A,B) 間の通信の場合はクラス Ch2 を継承させ，実際の PyChoral プログラムの際は class Foo(Ch2[A,B]) と記述する．式はリテラルのみ @ をつけて参加者の情報を得られる事とする．その他の式は基本的に型注釈もしくは Mypy を活用して検査した際に取得できる型情報から参加者の情報を得る．文の構文は Python と変化はない．

```
1 class Ch1(Generic[_R1]):
2     pass
3 class Ch2(Generic[_R1,_R2]):
4     pass
5 class Ch3(Generic[_R1,_R2,_R3]):
6     pass
7   ...
```

code 3.3　Choreography クラス

3.2.2　Projection

この節では PyChoral プログラムから各参加者の Python プログラムを導出するために重要なエンドポイント射影の定義についていくつか示す (定義の全体は付録 A を参照)．とあるマルチスレッドプログラムの参加者 A に対する PyChoral の項のプロジェクションは $(\!|Term|\!)^A$ と記し，これは $Term$ における参加者 A の振舞いを表す Python の項となる．これは図 3.1 の 3 ステップ目にあたる．

クラス定義の射影はクラス継承の第一引数 Ch クラスに存在する参加者名によって Python の項を生成する．

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

(例) 参加者 A, B が関わるクラス Foo の射影

$$\begin{array}{ccc} \text{PyChoral} & \Longrightarrow & \text{Python} \\ (\!|\text{class Foo(Ch2[A,B])}|\!)^A & \Longrightarrow & \text{class Foo\_A( )} \\ (\!|\text{class Foo(Ch2[A,B])}|\!)^B & \Longrightarrow & \text{class Foo\_B( )} \\ (\!|\text{class Foo(Ch2[A,B])}|\!)^C & \Longrightarrow & \end{array}$$

関数定義は引数の型注釈に射影される参加者の情報があればその引数を残す．

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

(例) 参加者 A が関連する引数をもつ関数 f の射影

$$\text{PyChoral} \qquad \Longrightarrow \qquad \text{Python}$$

$$(\!|\text{def f (self}, x : \text{At[int,A]})|\!)^A \implies \text{def f (self}, x)$$
$$(\!|\text{def f (self}, x : \text{At[int,A]})|\!)^B \implies \text{def f (self)}$$

式 Expression は射影する際に文字列を生成する (Expression → String)．リテラルは @ を付けることで関係する参加者を判別可能である．射影される参加者の情報がある場合はリテラルをそのまま生成し，ない場合は Unit 値が生成される．その他の式は型注釈や型推論から参加者の情報を取得し，各定義によって射影される．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 $\overline{Exp_2}$，メソッド呼び出し全体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．ここで，τ はメソッド呼び出し全体の型情報を表す．

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

(例)A と B のチャネルで A から B へメッセージを送るメソッド呼び出し

PyChoral	$\implies$	Python
$(\!|\text{ChAB.com('msg'@A)}|\!)^A$	$\implies$	`Unit.id`(ChAB.com('msg'))
$(\!|\text{ChAB.com('msg'@A)}|\!)^B$	$\implies$	ChAB.com(`Unit.id`)
$(\!|\text{ChAB.com('msg'@A)}|\!)^C$	$\implies$	`Unit.id`(`Unit.id`, `Unit.id`)

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．上記の例の場合，$\text{rolesOf}(Exp_1) = \{\text{A,B}\}$, $\text{rolesOf}(\overline{Exp_2}) = \{\text{A}\}$, $\text{rolesOf}(Exp_1.f(\overline{Exp_2})) = \{\text{B}\}$ である．

文 Statement は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 文の射影は $(\!|\textbf{return } Exp;|\!)^A = \textbf{return } (\!|Exp|\!)^A$ となり，返値 Exp の射影が反映された return 文が新しく Python プログラムとして生成される．しかし，特定の式文 (expression statement) と条件文 (if) は射影の形式が異なる．

式文の式 (Exp) がメソッド呼び出しであり，メソッド名が select の場合を選択文と呼ぶこととする．この選択文は射影すると match 文として Python プログラムが生成される．この時，メソッドの引数は Enum クラスの値であり，これが Match 文における場合分け時の値 (id_2) となる．

$$(\!|Exp; \overline{Stm}|\!)^A = \begin{cases} \textbf{match } (\!|Exp|\!)^A : \\ \quad \textbf{case } id_2 : \overline{(\!|Stm|\!)^A}; \qquad \text{if} \quad Exp = Exp_1.\texttt{select}(id_1.id_2@A()),\ id_1.id_2 : \texttt{Enum} \\ \quad \textbf{case } \_ : \text{assert False}; \\ (\!|Exp|\!)^A; \overline{(\!|Stm|\!)^A} \qquad \text{otherwise} \end{cases}$$

if 文は，条件式が射影対象の参加者が関係する式であれば，そのまま if 文の構造を Python プログラムで生成する．そうでない場合は if 文は消え，then 節と else 節に存在する後続の Statement を正規化し，

マージしたものが式 (Exp) に続いた形で射影される (マージと正規化の詳細は付録 B を参照).

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

マージ演算子 $\sqcup$ は，分岐のプログラムを結合する演算子である．基本的に，2 つの Python の項が与えられると再帰的にマージするには，それらが match 文でない限りは等価であるとする．ここではその match 文のマージについて述べる．

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

上記のように，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．

3.3　Mypy を活用したエンドポイント射影の実装

3.3.1　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 (code3.1) 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,_T2],_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, _:_R) -> At[Self,_R]: ...
```

code 3.4　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 3.5　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')
    f.write('from pychoral' + str(len(roles)) + ' import *\n')
    g = open(pro_filename, 'a')
    for stm in projection.projection_all(src.tree.defs,r,typechecker):
        data.stmt_to_string(stm,0)
```

```
20          g.write(data.stmt_to_string(stm,0))
```

code 3.6　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．

3.3.2　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 3.7　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．

3.3.3　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 3.8　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．

3.3.4　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．code3.9 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 $e.f(\overline{e})$ 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 $\overline{e}$ は n.args となる．パラメータ $e.f$ を $e.f(\overline{e})$ 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 $\overline{e}$ 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 3.9　pro_e.py

3.3.5　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 3.10　pro_s.py

3.3.6　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 3.11　pro_class.py

3.3.7　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 3.12　pro_func.py

3.3.8　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 3.13　pro_md.py

3.3.9　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 3.14　help_func.py

第 4 章　PyChoral を用いたアプリケーション

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 5 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] 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　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

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

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

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

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

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

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

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

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

Expression

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

A.3　normalizer

Statements

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

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

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

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

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

Expressions

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