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

Python は可読性が高く実用的で、高い拡張性を備えたプログラミング言語であり、機械学習や IoT の業界でも盛んに使用されている。一方で、マルチスレッド環境での Python を使用したプログラムは言語の性質上、並列処理の多くが犠牲になるケースが多い。しかし、近い将来 Python は仕様を変更し ([4])、その影響により Python ユーザーがマルチスレッド環境でのプログラミングをすることが増えると考えられる。ただ、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は困難であるため構築が難しい。よって、単一のプログラムとしてマルチパーティなプロトコルを記述し、デッドロック等の並行性起因エラーが起こらないことが保証されているコレオグラフィック言語なるものがあれば、多くの Python プログラマの不安や負担が解消される。

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。

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 [5] 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パラダイムである。これにより、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。コレオグラフィに従った各エンドポイント (通信の参加者) の型情報はエンドポイント射影 [7] という動作により、自動的に抽出される。

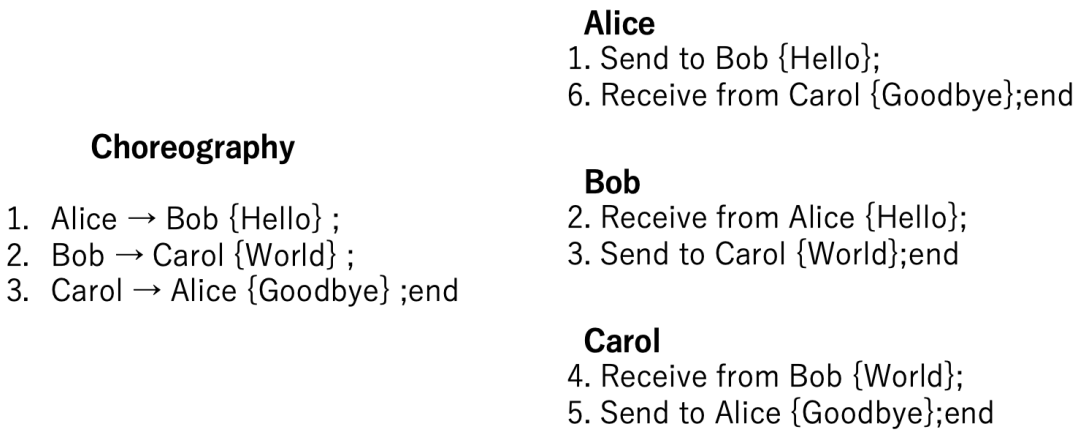

図 2.1　コレオグラフィとエンドポイント (Alice,Bob,Carol の会話)

ただし、通常のコレオグラフィはコンパイル時にエンドポイントの型情報を導出するだけであり、そこからプログラマが手動で導出した型情報を元に各エンドポイントのプログラムを記述しなければならないため、バグが起きた際の処理などがプログラマにとって負担になってしまうという難点がある。

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。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同時分散システム用の実行可能コードに変換する。

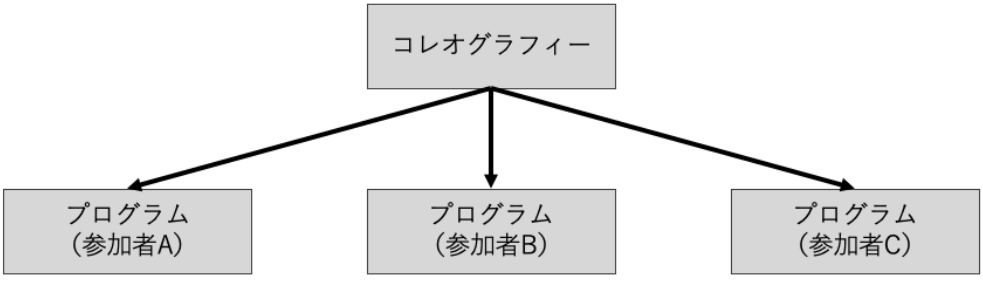

図 2.2　エンドポイント射影 (参加者 3 人)

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。前節で述べたように、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するのがプログラマの責任であり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。

以下は Choral のプログラム例と Choral コンパイラによって自動導出される Java プログラムの一例である [2]。

code 2.1　Choral プログラムの例

Choral プログラムはクラス名や変数、型などに @ で参加者名の情報を加えている。また、Choral には com メソッドと selection メソッドなるものがある [6]。

code 2.2　データ転送のための基本的な有向チャンネル

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

code2.2 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。

code 2.3　selection メソッドの定義

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

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 (code2.4)。

code 2.4　生成された Client の Java プログラム (分散認証)

Client,Service,IP のうち、Client にプロジェクションすると code2.4 が生成される。Java プログラムには@が存在せず、Choral プログラムで Client が関係する式や文のみ射影後に残る。code2.1 の 10〜21 行目の if 文は code2.4 の 14〜17 行目の switch 文に変換されており、case は列挙型の値で場合分けされて、それぞれ if 節と else 節の return 文をもつ。

このように 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トコルを記述する 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[3] 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。

code 2.5　型注釈のない Python コード

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

code 2.6　型注釈のある Python コード

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

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

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 を再構築し、各エンドポイントの Python プログラムを生成する。

図 3.1　PyChoral プロセス

型検査を行う際は `if isinstance()` で型のダウンキャストを行うことで射影するオブジェクトが所属しているクラスの判断をする。これを利用して各ノードの種類を判別している。また、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)。

3.2　PyChoral プログラムと生成される Python プログラムの例

コード 3.1 は参加者 A,B が挨拶をする PyChoral 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 A,B) の振舞いが関係することを示す。関数 sayHello の引数 a,b 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。

code 3.1　PyChoral

射影後の参加者 A,B 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。

code 3.2　generated Python

3.3　PyChoral の仕様

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 との違いについて説明する。次に、PyChoral プログラムをコンパイラによって 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。

3.3.1　syntax

Program	P	$::= P \cdot Class \mid P \cdot FuncDef \mid P \cdot Stm \mid P \cdot Exp \mid P \cdot EOF$
Class	$Class$	$::= \mathsf{class}\ id\ (\mathrm{Ch}(\overline{A}), \overline{Exp}) : \overline{Stm}$
Function	$Func$	$::= \mathsf{def}\ id\ (\overline{id(:TE)}) : \overline{Stm}$
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。

code 3.3　Choreography クラス

```
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   ...
```

クラスに関わる参加者の数によって継承する Choreography クラスが変わってくる。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。

3.3.2　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なる動作の定義についていくつか紹介する (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。

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

(例) 参加者 A に対してクラス Foo(Ch2[A,B]) から射影されたクラスは Foo_A() となる。Ch クラスの参加者名の中に射影される参加者の情報がない場合、そのクラスは Python の項として生成されない。

$$(\!|\mathsf{class}\ \text{Foo}\ (\text{Ch2[A,B]})|\!)^A = \mathsf{class}\ \text{Foo\_A}()$$
$$(\!|\mathsf{class}\ \text{Foo}\ (\text{Ch2[A,B]})|\!)^C = \texttt{absent}$$

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 に対して関数定義 def $f(x : \text{At[int,A]})$ から射影された関数は def $f(x)$ となり、def $f(x : \text{At[int,B]})$ から射影された関数は def $f()$ となる。

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

式 Expression は射影する際に文字列を生成する (Expression → String)。

リテラルは @ で関係する参加者を判別する。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、ない場合はユニット値が生成される。

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

(例) 参加者 A に対して 123@A() を射影すると 123 が文字列として生成される。

$$(\!|123@A()|\!)^A = 123$$
$$(\!|123@A()|\!)^B = \texttt{Unit.id}$$

メソッド呼び出し $Exp.f(\overline{Exp})$ はレシーバオブジェクト Exp、引数 $\overline{Exp}$、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。

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

(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 ChAB.com("msg"@A()) を参加者 A、参加者 B、参加者 C に対してそれぞれ射影すると、以下のようになる。

$$(\!|\text{ChAB.com("msg"@A)}|\!)^A = \texttt{Unit.id}(\text{ChAB.com("msg")})$$
$$(\!|\text{ChAB.com("msg"@A)}|\!)^B = \text{ChAB.com}(\texttt{Unit.id})$$
$$(\!|\text{ChAB.com("msg"@A)}|\!)^C = \texttt{Unit.id}(\texttt{Unit.id}, \texttt{Unit.id})$$

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、$\text{rolesOf}(Exp) = \{\text{A,B}\}$, $\text{rolesOf}(\overline{Exp}) = \{\text{A}\}$, $\text{rolesOf}(Exp.f(\overline{Exp})) = \{\text{B}\}$ 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 $(\!|\textsf{return}\ Exp;|\!)^A = \textsf{return}\ (\!|Exp|\!)^A$ となり、返値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、選択文とする。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。

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

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 をマージしたものが式 (Exp) に続いた形で射影される。

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

マージ演算子 $\sqcup$ は、分岐のプログラムを結合しようとする演算子である。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紹介する。

$\textsf{match}\ Exp :$　　$\textsf{match}\ Exp' :$　　$\textsf{match}\ Exp \sqcup Exp' :$

$$
\begin{array}{lll}
\textsf{case } id_a : Stm'_a; & \textsf{case } id_a : Stm''_a; & \textsf{case } id_a : Stm'_a \sqcup Stm''_a; \\
\cdots & \cdots & \cdots \\
\textsf{case } id_x : Stm'_x; \quad \sqcup & \textsf{case } id_x : Stm''_x; \quad = & \textsf{case } id_x : Stm'_x \sqcup Stm''_x; \\
\textsf{case } id_y : Stm'_y; & & \textsf{case } id_y : Stm'_y; \\
 & \textsf{case } id_z : Stm'_z; & \textsf{case } id_z : Stm'_z; \\
\textsf{case } \_\_ : Stm'_{ex}; & \textsf{case } \_\_ : Stm''_{ex}; & \textsf{case } \_\_ : Stm'_{ex} \sqcup Stm''_{ex};
\end{array}
$$

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。以下の code3.4 は code2.1 の if 文の部分を PyChoral の構文で記述したプログラムである。

code 3.4　射影前のプログラム

```
1 if valid: #@IP
2   ch_Client_IP.select(AuthBranch.OK@IP())
3   ch_Service_IP.select(AuthBranch.OK@IP())
4   ...
5   return AuthResult[Client,Service](ch_Client_IP.com(t), ch_Service_IP.com(t))
6 else:
7   ch_Client_IP.select(AuthBranch.KO@IP())
8   ch_Service_IP.select(AuthBranch.KO@IP())
9   return AuthResult[Client,Service]()
```

if 文の射影の定義により、条件式が射影される参加者と関係ないため code3.5 となり、これらがマージされて code3.6 となる。$(\!|Exp|\!)^A$; $[\![(\!|Stm_1|\!)^A]\!] \sqcup [\![(\!|Stm_2|\!)^A]\!]$; $(\!|\overline{Stm}|\!)^A$ の $[\![(\!|Stm_1|\!)^A]\!]$ 部分にあたるのが code3.5 の OK の場合であり、$[\![(\!|Stm_2|\!)^A]\!]$ 部分にあたるのが code3.5 の KO の場合である。

code 3.5　射影後のプログラム (マージ前)

```
1  # OK
2  match ch_Client_IP.select(Unit.id):
3    case OK:
4      return AuthResult(ch_Client_IP.com(Unit.id),Unit.id)
5    case __:
6      raise Exception
7
8  # KO
9  match ch_Client_IP.select(Unit.id):
10   case OK:
11     return AuthResult()
12   case __:
13     raise Exception
```

code 3.6　射影後のプログラム (マージ後)

```
1 match ch_Client_IP.select(Unit.id):
```

```
2   case OK:
3     return AuthResult(ch_Client_IP.com(Unit.id),Unit.id)
4   case OK:
5     return AuthResult()
6   case __:
7     raise Exception
```

また、マージされる後続の Statement はマージされる前に正規化される (詳細は付録 A.3 に記載)。

3.4　Mypy を活用したエンドポイント射影の実装

3.4.1　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ため Mypy を活用する。ただし、Mypy で型注釈をつけられば 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できるようにした。Choral はクラス定義、インターフェース、変数、型情報に @ で参加者の情報を加えていたが、通常の Python のパーサー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ようにした。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 内にある builtins.pyi ファイルに At クラス、Channel 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 Choral のプログラム例 2.1 の 8 行目にある String@Client salt は、PyChoral では salt:At[str,Client] と記述することで変数 salt の型が str で参加者 Client 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 @ のためのメソッドを追加することで **123@A**() のように @ をつけてプログラミングしても Mypy での検査が通るようにした。

code 3.7　builtins.pyi

```
1 class At(Generic[_T1,_T2],_T1):
2   pass
3 class Channel(Generic[_T1,_R1,_R2],_T1):
4   pass
5
6 class object:
7   ...
8   def __matmul__(self:Self, _:_R) -> At[Self,_R]: ...
```

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。また、main.py は構築した AST を保存せずに捨てていたが、mypycustom では `options.preserve_ast = True` によって AST を保存する。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ことで各参加者の Python プログラムを得る。(詳細は

省く、修論では記載)

code 3.8　mypytest.py

```
result : Mypy.build.BuildResult | None = mypycustom.main([
  "--show-traceback",
  "--custom-typeshed", "./typeshed",
  "ex.py"
  ])

src = result.graph["ex"]
typechecker = src.type_checker()

for stm in projection.projection(src.tree.defs,"A",typechecker):
  print(stm)
```

3.4.2　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。現段階では _repr_ を利用してフォーマットとして射影されたデータ木を文字列として出力するようにしているが、この部分は各参加者のファイルにそれぞれ印字されるように改良する。

code 3.9　projection_all.py

```
class Stmt:
  pass

class Block(Stmt): # list[stm]
    def __init__(self, body:list[Stmt]):
        self.body = body
    def __repr__(self):
        for stm in self.body:
            return f"{stm}\n"

class Pass(Stmt):
    def __repr__(self):
        return f"pass"

class Return(Stmt):
    def __init__(self, exp:str):
        self.expr = exp
    def __repr__(self):
        return f"return {self.expr}"
  ...
```

```
21
22 class ClassDef(Stmt):
23   def __init__(self, name:str, rolename:str, base_type_vars:list[str], defs:Block):
24       self.name = name
25       self.rolename = rolename
26       self.base_type_vars = base_type_vars
27       self.defs = defs
28   def __repr__(self):
29       return f"class {self.name}_{self.rolename}({help_func.list_to_str(self.
             base_type_vars)}): \n {self.defs}"
30
31 class FuncDef(Stmt):
32   def __init__(self, name:str, arguments:list[str] | None, body:Block | None):
33       self.name = name
34       self.arguments = arguments
35       self.body = body
36   def __repr__(self):
37       return f"def {self.name}({list_to_str(self.arguments)}): \n {list_to_str(self
             .body)}\n"
38
39 class Import(Stmt):
40   def __init__(self,ids:list[tuple[str, str | None]]):
41       self.ids = ids
42   def __repr__(self):
43       return f"Import {self.ids[[0]]}"
44   ...
```

3.4.3　projection_all

PyChoral プログラムの射影は projection_all.py に一任している。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。

code 3.10　projection_all.py

```
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 += [pro_md.projection_md(node)]
6          elif isinstance(node,ClassDef):
7              result += [pro_class.projection_class(node,r,tc)]
8          elif isinstance(node,FuncDef):
9              result += [pro_func.projection_func(node,r,tc)]
10         elif isinstance(node,Block):
11             result += [pro_s.projection_block(node.body,r,tc)]
12         else:
```

```
13              result += [pro_s.projection_stm(node,r,tc)]
14      return result
```

ダウンキャストされた文 (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。この result は生成される Python プログラムが収納されていく。

3.4.4　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。code3.11 は 4.3.2 節で紹介したメソッド呼び出しの射影 $\langle\!| Exp.f(\overline{Exp}) : \tau |\!\rangle^A$ 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。

code 3.11　pro_e.py

```
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         ...
```

3.4.5　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)。

code 3.12　pro_s.py

```
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   ...
```

3.4.6　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)。

code 3.13　pro_class.py

```
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  ...
```

3.4.7　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)。

code 3.14　pro_func.py

```
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))
```

3.4.8　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。

code 3.15　pro_md.py

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

3.4.9　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なるファイルである。ここには式 (Expression) の型情報から参加者名を取り出す関数 rolesOf()、型注釈から参加者名を取り出す関数 rolesOf_t()、@ が付いた値から参加者の情報を取り除いた値だけを取得する関数 NameExpr() などが存在する。これらの呼び出しは上記で記した関数内で現れる。

code 3.16　help_func.py

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

```
5 def nameExpr(e:Expression) -> str:
6   ...
```

3.5　PyChoral プログラム

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 4 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] Choral example. distributed-authentication. https://github.com/choral-lang/examples/tree/master/choral/distributed-authentication.

[3] mypy. https://mypy.readthedocs.io/en/stable/.

[4] python. https://peps.python.org/pep-0703/.

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

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

[7] 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A.2　Merging

Statement

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

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

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

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

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

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

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

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

Expression

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

A.3　normalizer

Statements

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

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

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

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

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

Expressions

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