中間発表 恩田晴登 (2023 年 7 月 25 日)

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

1 研究背景

コレオグラフィー [1] は，プログラマーが通信のロールごとに振舞いを個別に定義するのではなく，ロール間でどのように通信が起こるのかといった全体像を示すマルチパーティプロトコルである．コレオグラフィーで各ロールの動作を相互的に定義しているため，デッドロックや競合状態などの並行性に起因するエラーが起こらないことは保証されており，各ロールの具体的な実装はコレオグラフィーに従って記述することで安全性が保たれている．しかし，分散システムにおける各ロールのプログラミングをする際は，相互作用する複数のプログラムを正しく調整しながら記述する必要があり，手動で制御 (channel を介して送信・受信) するのはプログラマーの大きな負担となる．

先行研究で開発されたコレオグラフィックプログラミング言語である **Choral**[2] 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，プログラマーの負担を軽減する．Choral は Java に新しい型 $T@(R_1, \cdots, R_n)$ を加えて拡張した言語であり，これによって相互的な通信が各ロール間でどのように行われているかが分かるようになる．さらに，Choral で記述したコレオグラフィーをコンパイルすると，ロールごとの Java ライブラリに変換されて自動生成されて，各プログラムは全体のプロトコルに正しく従う．

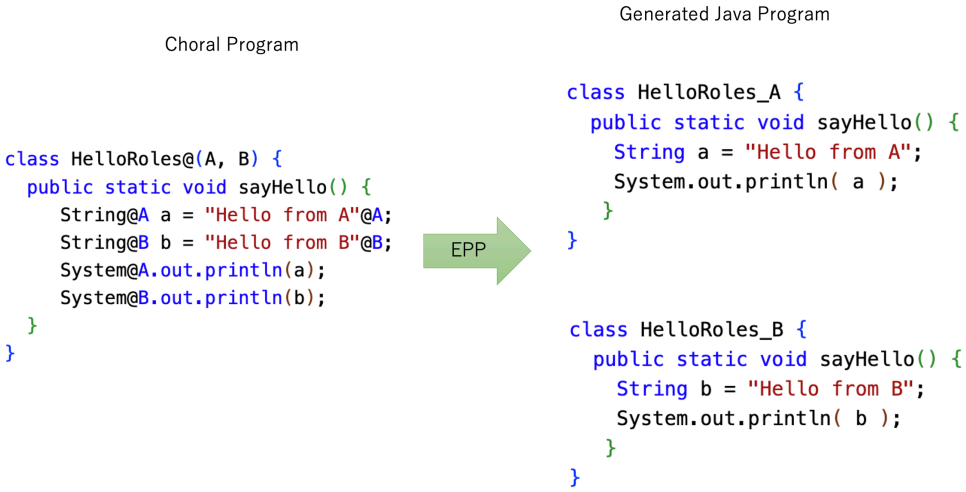

図 1: Choral の例

コンパイル時にコレオグラフィーからエンドポイントのプログラムを自動生成する研究は Java 以外の言語を拡張した場合も実装できるのではないかと考え，本研究では Java を拡張した Choral を参考に，Python を拡張したコレオグラフィックプログラミング言語である PyChoral の開発を行う．

2　研究概要

本研究はコレオグラフィーを用いて分散システムを記述するプログラマーの負担を軽減することを目的として，Python を拡張した Pychoral の開発をする．ここで，Choral の拡張元である Java が静的型付け言語であることに対して Python は動的型付け言語であるため，コンパイル時に型情報から発生するエラーは生じない．この問題を解決するために，Python に標準的に備わっている静的型検査器である **Mypy**[3] を使用してプログラムの実行前にエラーの有無を確認できるようにする．Mypy は Python プログラムに型注釈をつけて，その情報をもとにコンパイル時に型検査を行う．

しかし，Mypy を用いて Python を Java と同じように拡張すればいいかと言うと，問題がある．Choral の型検査では @ の後ろに型として記述されているロールの型情報をもとに各ロールのプログラムに射影していたのだが，Mypy には @ でロールの情報をとる機能はあらかじめ備わっていない．そこで，本研究では Python の object クラスに @ のためのメソッドを追加する．例えば，123@A と記述した場合 **At**[**int**,**A()**] という型情報が得られる．PyChoral では構文解析時にこの **At** 型が現れたらリストの第二引数からロールの情報を取得できる．

PyChoral は Mypy の内部機能によって一度構文木を構築し，visitor パターンによって射影するロールの情報を持ちながら構文木をトラバースし，射影の定義に従って各ロールの Python プログラムを文字列として生成する．

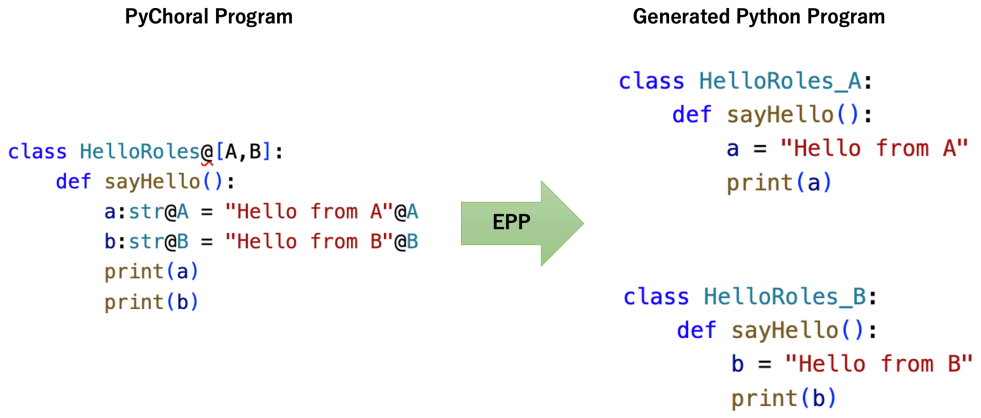

図 2: PyChoral(完成後) の例

3　進捗

Python のプログラムのうち式 (Expression) に対して構文・射影の定義，実装をした（ただし，PyChoral のプログラムを射影した結果を出力するのみ）．

例として，メソッド呼び出し `ch`$_{\texttt{AtoB}}$`.comm(1@A):int@B` の射影後のプログラムの実装を紹介する．`ch`$_{\texttt{AtoB}}$ はロール `A,B` 間のチャネルを表し，`comm` はあるロールからもう一方のロールに引数の値を渡すメソッドである．例の場合，ロール A がロール B に対して `int` 型の値をチャネルを介して送信していることになる．これをロール A，ロール B，この通信に関係ないロール C に対して，本研究で定義した方式で射影した結果を以下に示す．ただし，$(\!|Term|\!)^A$ はロール A に対する項の

射影を表すとする．

$$(\!|\texttt{ch}_{\texttt{AtoB}}\texttt{.comm(1@A):int@B}|\!)^{A} = \texttt{Unit.id(ch}_{\texttt{AtoB}}\texttt{.comm(1))}$$
$$(\!|\texttt{ch}_{\texttt{AtoB}}\texttt{.comm(1@A):int@B}|\!)^{B} = \texttt{ch}_{\texttt{AtoB}}\texttt{.comm(Unit.id)}$$
$$(\!|\texttt{ch}_{\texttt{AtoB}}\texttt{.comm(1@A):int@B}|\!)^{C} = \texttt{Unit.id}$$

ロール A の場合，チャネルと comm の引数に関わっているが，メソッド呼び出し自体の戻り値としては関与していないため，呼び出しだけを残した形でロール A の Python プログラムに射影される．ロール B の場合，戻り値の型情報で出てきているため式は残るが，comm の引数の型情報としてロール B の情報は取れないため，引数は Unit 値となる．ロール C の場合，通信を行うチャネルに関与してないため，ロール C のプログラムにはこのメソッド呼び出しは記述されない．Unit.id と射影された後，その値をプログラム中から消すかどうかは Normalizer で処理するため，現在考察中である．

このように，構文や射影方式の定義を基にして式だけでなく文やクラス定義などを実装することで，Choral と同様に，コレオグラフィーから振り付け規則を遵守したエンドポイントの Python プログラムを生成する言語の開発を進める．

4 今後の課題

- 引き続き PyChoral の構文定義と実装を行う
- Merging,Normalizer の理解を深める
- 現段階では PyChoral プログラムから射影された Python プログラムを文字列として出力しているだけであるため，これらを新たなファイルとして生成するように変更する．
- PyChoral を用いて分散システムのプログラム例をいくつか示す．
  （例：分散認証システム，クイックソート，マージソート）

参考文献

[1] Saverio Giallorenzo, Fabrizio Montesi and Marco Peressotti. Object-Oriented Choreographic Programming, 2022.

[2] Choral. https://www.choral-lang.org/index.html

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