A-032

# 分散オブジェクトからπ計算プロセスへの変換系

A Translation System of Distributed Objects into Pi-Processes

甲斐 貴史† 加藤 暢‡ 樋口 昌宏†
Takafumi Kai Toru Kato Masahiro Higuchi

## 1. はじめに

分散オブジェクト技術を利用した分散システムでは, オブジェクトの参照を変更することで通信相手を動的に切り替えることができる. 一方, π計算 [1] にはプロセス間でチャネル (通信に使用する名前) そのものを受け渡す機能がある. オブジェクトの参照渡しとπ計算のチャネル送信には, 通信相手の動的な切り替えという類似点が見られる. 我々はこの点に着目し, Java プログラムからπ計算への変換方法を提案し, さらに変換系を作成した [2]. Java プログラムからπ計算プロセスへと変換することで, PiET[3] を用いた自動的な等価性検証が可能となる. しかし, その変換系では動的なオブジェクト生成, 制御構造に対応できていなかった. 本報告では, コンストラクタ, 制御構造に対応した変換方法を提案する.

## 2. π計算

π計算はプロセス代数の一種であり, 通信に使用するチャネル名そのものも値として扱うことができるという特徴を持つ. 遷移動作の例を示す.

$$a(x).P \mid \bar{a}b.Q \quad \xrightarrow{\tau} \quad P\{b/x\} \mid Q \tag{1}$$

式 (1) は, 出力 $\bar{a}b$ と入力 $a(x)$ が同期的に実行され, $P$ 中の全ての名前 $x$ が $b$ に置き換えられることを表している. これにより, $P$ は $b$ を介してそれまで知らなかった相手と通信できるようになる. π計算には, PREFIX(連接)“.”, SUM(選択)“+”, PAR(並行)“|”, MATCH(等価)“=”, MISMATCH(非等価)“≠”, RES(制限)“$\nu$”などの演算子がある. その操作的意味は文献 [1] に準ずる.

## 3. 変換して得られるプロセスの構造

Java ソースコードを本変換系で変換すると, それに含まれるクラスごとにその機能を表すπ計算プロセス式が生成される. 得られるプロセス式には, クラス, コンストラクタ, メインメソッド, インスタンス, インスタンスメソッドそれぞれに相当するπ計算プロセスが定義されている. 変換後のプロセス式の構造を図 1 に示す. また, その具体例を図 2 (4 節) に示す.

**クラスプロセス:** Java ソースコードのクラス定義に相当するプロセスである. クラスプロセスにはコンストラクタとメインメソッドのプロセスが含まれる.

**コンストラクタプロセス:** コンストラクタ定義に相当するプロセスである. コンストラクタプロセスにはインスタンスプロセスが含まれており, アクセスされるとインスタンスプロセスが一つアクティブになる.

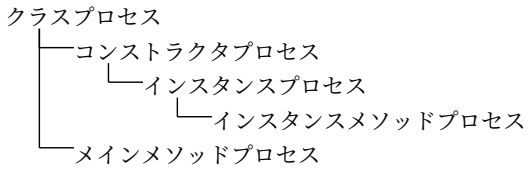


図 1 変換後のプロセスの構造

**インスタンスプロセス:** 生成されたインスタンスを表すプロセスである. インスタンスプロセスにはインスタンスメソッドプロセスが含まれる. インスタンスプロセスがアクティブになると, そのプロセス固有のチャネルが割り当てられる. このチャネルが Java プログラムでのオブジェクト参照に相当する.

**インスタンスメソッドプロセス:** インスタンスメソッドの定義に相当するプロセスである. インスタンス固有のチャネル (Java ソースコードのオブジェクト参照に相当) を使用してアクセスされる.

**メインメソッドプロセス:** メインメソッドプロセスには, Java のメインメソッドと同じ順序で処理が実行されるように, コンストラクタやインスタンスメソッドのプロセスにアクセスするための操作が記述される.

## 4. Java からπ計算への変換

ここでは簡単な Java ソースコードを用いて具体的な変換方法について説明する. なお, 本変換系で対象とする Java ソースコードでは, オブジェクト間の相互作用をメソッド呼び出しによるもののみに限定する. つまり, インスタンスへのフィールド参照を禁じる. また, static なメソッドはメインメソッドのみとする. 本変換系では変換対象に Java RMI を利用した分散アプリケーションを想定しており, オブジェクトのローカル/リモートの差異を無くすためにこれらの制限を設ける.

図 2 は図上部の Java ソースコード (Foo クラス) から図下部のπ計算プロセス式へと変換した例である. Foo クラスは Remote インタフェースを実装しているため, インスタンスは Java RMI のリモートオブジェクトとなる. 変換後のプロセス式には 3 節で示した各プロセスが含まれている. 変換後のプロセス式について説明する.

`List L`: PiET では, プロセス式中で使用されるチャネル名は予め宣言されている必要がある. `L` は宣言済みチャネル名のリストである. 各プロセスの定義に (`L`) を記述することで, `L` 内のチャネルの使用を宣言する.

`AGENT Cls_Foo`: Java ソースコード `Foo` クラスから得られる

† 近畿大学大学院総合理工学研究科
‡ 近畿大学理工学部情報学科

```
public class Foo implements Remote {
  public Foo() {}
  public void bar1() {}
  public void bar2(Object baz) {}
  public static void main(String[] args) {
    Foo foo = new Foo();
    foo.bar1();}}
```

↓

```
List L := clsFoo, conFoo, bar1, bar2;
AGENT Cls_Foo(L) := !clsFoo(ch).ch(con).
  Con_Foo(L, ch) | !clsFoo().Main_Foo(L);
AGENT Con_Foo(L, ch) := ^obj ch<obj>.
  Ins_Foo(L, obj);
AGENT Ins_Foo(L, obj) := obj(ch).ch(mtd).
  ([mtd=bar1]Mtd_bar1(L, ch, obj) +
   [mtd=bar1]Mtd_bar2(L, ch, obj));
AGENT Mtd_bar1(L, ch, obj) := ch<>.
  Ins_Foo(L, obj);
AGENT Mtd_bar2(L, ch, obj) := ch(baz).
  ch<>.Ins_Foo(L, obj);
AGENT Main_Foo(L) := ^ch clsFoo<ch>.
  ch<conFoo>.ch(obj).^ch foo<src>.
  ch<bar1>.ch());
```

図 2　Java プログラムからπ計算式への変換例

クラスプロセスである．コンストラクタとメインメソッドのプロセスが PAR 演算子によって並行に動作している．また，“!”の記号は両プロセスを何度でもアクティブにできることを表している．

clsFoo を通じて ch を受け取るとコンストラクタプロセスがアクティブになる．ch は，コンストラクタへの引数を受け取るため，そして，インスタンスプロセス固有のチャネル (後述の obj) を呼び出し元へ送り返すために使用される．また，clsFoo から受け取るチャネルが無ければ，メインメソッドプロセスがアクティブになる．

AGENT Con_Foo: コンストラクタプロセスである． 先頭の ^obj は，チャネル obj を新規に生成する操作である． この obj は Java ソースコードでのオブジェクト参照に相当する． 続く ch<obj> で呼び出し元に obj を渡し，最後の Ins_Foo(L, obj) でチャネル obj を持つインスタンスプロセスをアクティブにする．

AGENT Ins_Foo: インスタンスプロセスである． 先頭の obj(ch) は呼び出し元を特定するためのチャネルを受け取る操作であり，メソッドが呼ばれたことを表す．このあとの ch(bar1) でメソッド名を受け取ると，あとの条件式にマッチするインスタンスメソッドプロセスがアクティブになる．

AGENT Mtd_bar1 と AGENT Mtd_bar2: インスタンスメソッドプロセスである．Java ソースコードの bar2 メソッドは引数を持つため，Mtd_bar2 には最初に ch を使用して引数 obj を受け取る操作がある (Mtd_bar1 では不要である)．処理内容がある場合はこの後に続く．両プロセスの ch<>は呼び出し元へメソッド処理の終了を伝える操作である．戻り値がある場合は ch<**戻り値**>と記述する．最後の Ins_Foo(L, obj) はインスタンスが元の状態に戻ったことを表す．

AGENT Main_Foo: メインメソッドプロセスである．Java ソースコードの処理順序に従って，コンストラクタを呼び出したあとで foo のメソッドを呼び出すという操作が記述されている．^ch は，コンストラクタやメソッドの呼び出し元を特定するためのチャネルを生成する操作である．これ以降の foo への操作は全て obj に対して実行され，結果が ch を介して呼び出し元へ返される．

## 5． 制御構造

制御構造の条件判定では，オブジェクト参照の同一性を評価する if 文と while 文に限定する．Java ソースコード中に if 文が記述されていた場合，以下のような変換が実行される．

```
if(obj1.equals(obj2)) { P }
else { Q }
```

↓

```
[obj1 = obj2]P + [obj1 # obj2]Q
```

図 3　if 文の変換例

図 3 の上部は，obj1 と obj2 が同じオブジェクトの参照であるかを評価する Java ソースコードである．このソースコードを変換すると下部のようなプロセス式が生成される．この例では評価結果が真の場合に P，偽の場合に Q を動作させる．なお，ここでは条件分岐後の処理を P, Q と省略した．

もう一つの制御構文 while は if 文と同様のプロセス式に変換される．異なる点は，評価結果が真になった場合にはプロセスの再帰呼び出しが実行される点である．

## 6． 結論

本研究では RMI を利用した Java 分散プログラムからπ計算プロセス式への変換系を提案した．本変換系を利用することで，Java プログラムからπ計算プロセス式を生成できる．プロセス式には，Java のクラスやメソッドの定義を表現するプロセスが含まれる．Java プログラムからπ計算に変換することで Java プログラムの形式的な扱いが可能となり，PiET を使用して各プロセスの等価性を自動的に検証できるようになる．

## 参考文献

[1] J.Parrow, :An Introduction to the π-Calculus, in J.A.Bergstra, A.Ponse, and S.A.smolka, eds., HANDBOOK OF PROCESS ALGEBRA, NORTH-HOLLAND (2001).

[2] 山口 将志, 他, 動的な接続関係を持つ Java プログラムの一記述法とπ計算への変換, FIT2005

[3] Matteo Mio, PiET:Pi Calculus Equivalences Tester, (http://piet.sourceforge.net/)