Aπ 計算の Coq による形式化

安武 祥平　渡部 卓雄

アクターモデルは並行計算のモデルのひとつであり，アクターと呼ばれる計算主体がそれぞれ並列に動作し互いに非同期にメッセージを送りあって計算を進める．アクターモデルは，アクターの名前は一意であるという性質，他のアクターの名前あるいはメッセージとして送られてきた名前ではアクターを作れないという性質，アクターにはいつでもメッセージを送ることができるという性質，の 3 つの性質を持っている．これらの性質を満たすように π 計算に型を付け，アクターモデルとしての振る舞いを強制させたものに Aπ 計算があるが，Aπ 計算が確かにアクターモデルとしての振る舞いを示すかどうかについて形式的な証明はまだ与えられていない．そこで，本研究では，Aπ 計算およびその型システムの定義を，等価だが証明を進めやすい定義に変更するなどの工夫を行い，Aπ 計算の型システムがアクターモデルとしての振る舞いを強制することについて，定理証明支援系 Coq を用いて形式的な証明を与えた．

英語アブスト

1　はじめに

π 計算やアクターモデルは並行計算のモデルとして発展してきた．アクターモデルはアクターと呼ばれる計算主体 (computing entity) が並列に動作し，互いに非同期にメッセージを送り合うことによって計算を行う．アクターモデルの持つべき性質として，アクターの名前は一意であるという性質，メッセージとして送られてきた名前でアクターを作ることはできないという性質，アクターにはいつでもメッセージを送ることができるという性質，を持っている．

このようなアクターモデルの性質を満たすように設計された，π 計算を用いた形式化として，Aπ 計算が提案されている [3] ．Aπ 計算は，π 計算に型を付けることによってアクターモデルの性質を表現することを目的としており，May Testing と呼ばれる，二つのアクターの等価性についての理論に使われている．

このような基礎的な理論には形式的な証明があることが望ましい．しかし，Aπ 計算の性質に対して形式的な証明を与えているものはまだない．

1.1　目的

本研究では，Aπ 計算の型付けにおける健全性に対して，形式的な証明を与えることを目的とする．形式的な証明を与えるためのツールとして定理証明支援系 Coq を用いる．また，Coq で証明する際に，Aπ 計算およびその型システムについての定義をより具体的なものに定義しなおすことによって，より簡単に証明を行えるようにしている．

2　背景知識

2.1　アクターモデル

アクターモデルは非同期メッセージ通信に基づいた並行計算のモデルである [4]．アクターと呼ばれる計算主体があり，それらは名前 (またはアドレス) と内部状態をもっている．各アクターは並列に動作し，非同期にメッセージを送り合うことでコミュニケーションをとる．

アクターは，メッセージの内容に応じて一定の動作を行う．これを振る舞い (behavior) という．振る舞

A formalization of Aπ-calculus using Coq
Shohei Yasutake, 東京工業大学大学院情報理工学研究科, Dept. of Computer Science, Tokyo Institute of Technology.

いには以下の様なものがある．

- 他のアクターにメッセージを送信する．
- 新しいアクターを作る．
- 自らの振る舞いを変える．

2.1.1 配置

アクターモデルにおける配置とは，アクターモデルの世界におけるその時点での状態を切り取ったものを表すための概念である．配置はアクターとその振る舞い，まだ受け取られていないメッセージの集合から表される．

配置内の，配置外からのメッセージを受け取ることのできるアクターを，窓口 (receptionist) という．アクターは，送り先の名前 (アドレス) を知ることで，その名前のアクターにメッセージを送れるようになる，よって窓口は外部のアクターに自身の名前を知られているアクターと言い換えることができる．また，窓口の集合のことを窓口集合 (receptionist set) という．

2.1.2 アクターモデルの性質

アクターモデルが満たすべき性質として，以下がある．

一意性 (uniqueness property)
アクターの名前は一意である．

新鮮性 (freshness property)
アクターが作られるとき，作られるアクターの名前はまだどのアクターの名前でもない．アクターは，メッセージとして送られてきた名前でアクターを作ることはできない．

永続性 (persistence property)
アクターは消えない．一旦アクターが作られると，いつでもそのアクターにメッセージを送ることができる．

2.2 π 計算

π 計算は，Milner らによって提唱された並行計算のモデルである [6]．プロセスと呼ばれるオブジェクト同士が通信しデータを受け渡していくことで計算を進める．また，アクターモデルとは異なり，メッセージを送受信する際にはプロセス間で同期をとる必要がある．

π 計算の抽象構文は 図 1 のようになっている．ここで，$P, P_1, P_2, ...$ はプロセスの式，$x, y, ...$ は名前である．

$P ::= 0$	何もしない
$\mid x(y).P$	x から値を受け取り,P 中の y に束縛する
$\mid \bar{x}\langle y\rangle.P$	x へ y を送る
$\mid P_1 \mid P_2$	P_1, P_2を並列に実行させる
$\mid (\nu x)P$	P 中に現れる x を束縛する
$\mid\ !P$	P を繰り返す

図 1: π 計算の文法

2.3 定理証明支援系 Coq

Coq はフランス国立情報学自動制御研究所で開発されている定理証明支援系である [2]．Coq を用いることで，プログラムがある仕様を満たすということや，数学的な定理などに対して形式的かつ厳密な証明を与えることができる．Coq によって証明されたものとしては，四色定理が有名である [5].

2.3.1 カリー・ハワード同型対応

Coq における定理証明は，カリー・ハワード同型対応という理論に基づいている．カリー・ハワード同型とは，論理における命題と証明という構造と，コンピュータプログラムにおける型と項という構造が直接的に対応しているという理論である．これにより，ある命題を証明することが，命題に対応する型と，その型を持つ項を作ることに帰着される．

カリー・ハワード同型対応では，表 1 に示すような対応関係がある．

3 Aπ 計算

本章では，アクターモデルの振る舞いを強制するために π 計算に型を付けた Aπ 計算について説明する．まず準備として記号の定義をし．Aπ 計算の配置の定義をする．次に型システムを導入するが，その際に仮名関数という関数を導入し，型付け規則を定義する．最後にこの型システムにおける健全性について述べる．また，Aπ 計算の定義および使われている記号

表 1: カリー・ハワード同型対応

論理	コンピュータプログラム
命題 P	型 `P`
命題 P の証明	型 `P` を持つ項
命題 $P \Rightarrow Q$	型 `P -> Q` (`P` から `Q` への関数)
命題 $P \wedge Q$	`P, Q` の直和型
命題 $P \vee Q$	`P, Q` の直積型

は，文献 [3] で用いられているものを用いる．

3.1 配置

配置 (configuration) は，アクターシステムのある時点での状態を表す．配置の文法は，図 2 のようになっている．

$$
\begin{aligned}
P ::= {} & 0 \mid x(y).P \mid \bar{x}y \mid (\nu x)P \mid P_1 | P_2 \\
& \mid \texttt{case}\ x\ \texttt{of}\ (y_1 : P_1, ..., y_n : P_n) \\
& \mid ((\tilde{u}, \tilde{v})u_1(z).P)(\tilde{x}, \tilde{y})
\end{aligned}
$$

図 2: Aπ 計算の配置の文法

通常の π 計算と異なる点は，他のアクター (プロセス) にメッセージを送ったあとにさらに何かを実行できないこと，名前の場合分けが書けること，振る舞いの雛形からアクターを作るための文法があること，の 3 つである．

3.2 型システムの導入

Aπ 計算の文法のみでは，アクターモデルの性質は満たさないような配置も書けてしまう．例えば，$x(u).P|x(v).Q$ は Aπ 計算の文法で書けるが，x という名前のアクターが 2 つ作られるので，一意性を満たしていない．また，$x(u).(u(v).P|x(v).Q)$ も Aπ 計算の文法で書けるが，メッセージとして送られてきた名前でアクターが作られているので，新鮮性に反する．

また，アクターモデルでは，1 回の通信で複数の名前を送信することができるが，Aπ 計算では 1 回の通信で一つの名前しか送ることはできない．複数の名前を送信するには，その数だけ連続して通信する必要がある．しかし，アクターモデルではこのような複数の名前を含むメッセージの受け渡しはアトミックに行なわれ，即座に次のメッセージを受け取れる状態にならなければいけない．そこで，いつでもメッセージを送信できるという永続性を弱め，アクターはメッセージの送信先のアクターの処理が終わるまで待てるようにする (いつかは必ず送れる)．そしてアトミックに行うことのできない処理の最中はメッセージを送れないということを，正規の名前を隠して一時的な名前を使って処理を行うことで実現させる．ここで，仮の名前をとった後，元の名前とは異なる名前になってしまうと，そのアクターにメッセージを送ろうとしているアクターはメッセージを送れなくなってしまい，永続性に反する．仮の名前をとったときには必ず元の名前に戻るという制約が必要である．

以上のような性質を満たすようにするため，型システムを導入し，Aπ 計算に制約を与える．

3.2.1 仮名関数

仮の名前は必ず正規の名前に戻るという制約を型システムを用いて与えるためには，仮の名前をとったときに正規の名前を覚えておく必要がある．そこで仮の名前から元の正規の名前にマッピングする関数を導入する．これを 仮名関数 (temporary name mapping function) と呼ぶ．

定義 3.1 (仮名関数)**.** 仮名関数 f は，集合 X から集合 X^* への写像である．ただし，$\perp, * \notin \mathcal{N}, X \subset \mathcal{N}, X^* = X \cup \{\perp, *\}$ である．

仮名関数の入力と返り値の関係は，以下のような意味を持つ．

$f(x) = \perp$	x が正規の名前のとき
$f(x) = *$	x が環境から隠されている名前であるとき
$f(x) = y \notin \{\perp, *\}$	アクター y の仮の名前が x であるとき

3.2.2 仮名関数の構築

仮名関数は以下に示す ch 関数，仮名関数の合成，仮名関数の制限のいずれかで作ることができる．

定義 3.2 (ch)．ch は名前の組をとり，仮名関数を返す関数である．組の長さは任意である．組を $\langle x_1, x_2, ..., x_n \rangle$ としたとき，$ch(\langle x_1, x_2, ..., x_n \rangle)$ は以下のような仮名関数となる．

$$ch(\langle x_1, x_2, ..., x_n \rangle)(x_i) = \begin{cases} x_{i+1} & 1 \leq i < n \text{ のとき} \\ \bot & i = n \text{ のとき} \end{cases}$$

定義 3.3 (仮名関数の合成)．$f_1 : \rho_1 \to \rho_1^*$，$f_2 : \rho_2 \to \rho_2^*$ とする．仮名関数の合成を表す二項演算子 $f_1 \oplus f_2 : \rho_1 \cup \rho_2 \to (\rho_1 \cup \rho_2)^*$ を以下のように定義する．

$$(f_1 \oplus f_2)(x) = \begin{cases} f_1(x) & x \in \rho_1 \text{かつ } f_1(x) \neq \bot \text{，または} \\ & x \in \rho_1 \text{かつ } x \notin \rho_2 \text{のとき} \\ f_2(x) & \text{その他} \end{cases}$$

定義 3.4 (仮名関数の定義域の制限)．$f : \rho_1 \to \rho_1^*, \rho \subset \rho_1$ である f, ρ に対して定義域の制限を表す二項演算子 $f|\rho : \rho \to \rho^*$ を以下のように定義する．

$$(f|\rho)(x) = \begin{cases} f(x) & f(x) \in \rho \text{のとき} \\ * & \text{その他} \end{cases}$$

3.2.3 仮名関数が満たすべき性質

仮名関数 f はその定義から，以下を満たすべきである．

- $f(x) \neq x$
- $f(x) = f(y) \notin \{\bot, *\} \Rightarrow x = y$
- $f^*(f(x)) = \bot$

$f(x) \neq x$ は，仮の名前から正規の名前を得たとき，それは仮の名前と同名ではないという性質である．$f(x) = f(y) \notin \{\bot, *\} \Rightarrow x = y$ は，正規のアクターは 2 個以上の仮の名前をとれないという性質である．$f^*(f(x)) = \bot$ は，仮の名前のアクターはさらに仮の名前を持つことはできないという性質である．

また，互換性という性質を定義する．

定義 3.5 (互換性 (compatible))．二つの仮名関数 f_1, f_2 が以下の性質を満たすとき，f_1, f_2 は互換性を持つという．また，この性質を満たすという述語を $compatible(f_1, f_2)$ と書く．

- $f_1 \oplus f_2 = f_2 \oplus f_1$
- $f_1 \oplus f_2$ が仮名関数の性質を満たす

3.3 型付け規則

Aπ 計算が先に述べた性質を満たすようにするための型付け規則は 図 3 のようになっている．配置 P，P の窓口集合 ρ，P 中の仮の名前から正規の名前への写像 $f : \rho \to \rho^*$ で，$\rho; f \vdash P$ が成り立つとき，P は正しく型付けされている．

それぞれの規則についての説明を以下に示す．

3.4 型システムの健全性

Aπ 計算の型システムにおける健全性は，この型システムを満たした Aπ 計算の配置はアクターモデルとしての振る舞いを示すということを表した性質である．単純型付き λ 計算における型システムの健全性のような，正しく型付けされるならば stuck 状態にならないという性質とは少し異なることに注意されたい．

Aπ 計算の型システムの健全性は以下のように定義されている．

定義 3.6 (Aπ 計算の健全性)．配置 P について，$\rho; f \vdash P$ で型付けできたとき．以下の *3* つの補題を満たすならば，この型システムは健全である．

補題 3.1 (窓口集合の健全性)．ρ は P 中に現れる自由名の部分集合

補題 3.2 (仮名の取り方の健全性)．以下の *3* つを満たす．

1. $f(x) \neq x$
2. $f^*(f(x)) = \bot$
3. $f(x) = f(y) \neq \{\bot, *\} \Rightarrow x = y$

補題 3.3 (型付けの一意性)．$\rho'; f' \vdash P$ ならば，$\rho = \rho'$ かつ $f = f'$

それぞれの補題は，以下の意味を持っている．

窓口集合の健全性

窓口集合に属するアクターはメッセージとして送られてきた名前で作られたものではなく，かつ環境から隠されたものでもない．

$$\frac{}{\emptyset; \{\} \vdash 0} \quad \text{(Nil)}$$

$$\frac{}{\emptyset; \{\} \vdash \bar{x}y} \quad \text{(Msg)}$$

$$\frac{\rho; f \vdash P \qquad \rho - \{x\} = \hat{z} \qquad y \notin \rho \qquad f = \begin{cases} ch(\langle x \rangle \tilde{+} \langle \hat{z} \rangle) & x \in \rho \text{のとき} \\ ch(\langle \hat{z} \rangle) & \text{その他} \end{cases}}{\{x\} \cup \hat{z}; ch(\langle x \rangle \tilde{+} \langle \hat{z} \rangle) \vdash x(y).P} \quad \text{(Act)}$$

$$\frac{\forall 1 \leq i, j \leq n, i \neq j,\ \rho_i; f_i \vdash P_i \qquad compatible(f_i, f_j)}{(\cup_i \rho_i); (f_i \oplus f_2 \oplus ... \oplus f_n) \vdash \texttt{case}\ x\ \texttt{of}\ (y_1 : P_1, ..., y_n : P_n)} \quad \text{(Case)}$$

$$\frac{\rho_1; f_1 \vdash P_1 \qquad \rho_2; f_2 \vdash P_2 \qquad \rho_1 \cap \rho_2 = \phi}{\rho_1 \cup \rho_2; f_1 \oplus f_2 \vdash P_1 | P_2} \quad \text{(Comp)}$$

$$\frac{\rho; f \vdash P}{\rho - \{x\}; f | (\rho - \{x\}) \vdash (\nu x)P} \quad \text{(Res)}$$

$$\frac{\{\tilde{u}\}; ch(\tilde{u}) \vdash u_1(z).P \qquad len(\tilde{x}) = 2 \ \Rightarrow\ x_1 \neq x_2}{\{\tilde{x}\}; ch(\tilde{x}) \vdash ((\tilde{u}, \tilde{v})u_1(z).P)(\tilde{x}, \tilde{y})} \quad \text{(Inst)}$$

図 3: 型付け規則

仮名の取り方の健全性

アクターが仮の名前をとったとき，それは同名のものではなく，さらに仮の名前を持つこともない．また，アクターが複数の仮の名前を持つことはない．

型付けの一意性

配置に型が付いたとき，その型付けは一意である．

Aπ 計算は以下の定理が成り立つ．

定理 3.1. *Aπ 計算の型システムは健全である．*

4 Coq による形式化

本節では，3 節で述べた Aπ 計算についてどのように Coq で記述していくかについて説明する．まず名前，配置，配置中の自由名，仮名関数およびその性質についての Coq での表現を述べる．次に型付け規則を Coq で使いやすくための規則の具体化を説明し，最後に型システムの健全性の Coq での証明方針を述べる．本研究では，アクターの振る舞いに関するこの 3 つの補題について，Coq を使って形式的な証明を与える．

4.1 仮名関数

次に，仮につけた名前から正規の名前を得る 仮名関数の Coq での定義について述べる．

仮名関数 f の型は $X \to X^*$ であり，定義域と値域の集合は $\perp$ と $*$ を除くと等しくなっている．Coq では簡単にするためにその情報を捨て，図 **??** のように任意の `name` 型の値から任意の `option star` 型への関数とした．定義域の範囲に入っていない入力は `None` を返すことを想定している，

この変更により，以下の述語が必要となる．$f : X \to X^*$ ということを，$in\_range\_in\_domain(f)$ を満たすことで表している．

定義 4.1 (*in_domain*)**.** 名前 x ，仮名関数 f につい

て，$x \in domain(f)$ を $in\_domain(f,x)$ で表す．ただし $domain(f)$ は f の定義域である．

定義 4.2 (in_range)．名前 x ，仮名関数 f について，$x \in range(f)$ を $in\_range(f,x)$ で表す．ただし $domain(f)$ は f の値域である．

定義 4.3 ($in\_range\_in\_domain$)．ある 仮名関数 f が，任意の名前 x について $in\_range(f,x) \Rightarrow in\_domain(f,x)$ を満たすということを $in\_range\_in\_domain(f)$ で表す．

また，2 つの 仮名関数の定義域が重なっていないことを表す述語 $fun\_exclusive$ を定義する．

定義 4.4 ($fun\_exclusive$)．2つの 仮名関数 $f : X \to X^*, g : Y \to Y^*$ について，$X \cap Y = \emptyset$ であるという述語を $fun\_exclusive$ と定義する．

4.1.1 仮名関数の構築

仮名関数は ch 関数，仮名関数の合成，仮名関数の定義域の制限，のいずれかで作ることができる．ここで，ch 関数が使われている型付け規則は Act のみであるが，この中で使われている ch 関数は入力の要素数が 0 個，1 個，2 個のものしかない．よって，ch 関数はこの 3 パターンに限定でき，それぞれ ch_0, ch_1, ch_2 と書くこととする．また，仮名関数の定義域の制限を行う二項演算子 (|) の第二引数は要素が 1 個の場合しかなく，要素を 1 個に限定して定義すると，証明が簡単になる．よって，仮名関数の定義域の制限は，以下の定義を用いる．

定義 4.5 (仮名関数の定義域の制限)．$f : \rho \to \rho^*, x \in \rho$ である f, x に対して定義域の制限を表す二項演算子 $f|x : \rho - x \to (\rho - x)^*$ を以下のように定義する．

$$(f|x)(y) = \begin{cases} * & f(y) = x \text{ のとき} \\ f(y) & \text{その他} \end{cases}$$

また，以上の 5 つについて仮名関数の性質を満たすことを証明した．

4.2 型付け規則

本節では，Aπ 計算の型付け規則を Coq で定義する．

Aπ 計算の型付け規則には，Coq 上で定義しにくい，または証明を進めにくいものがある．実際に Coq で型付け規則を定義する前に，一般的すぎる型付け規則は具体化しておく．

4.2.1 Act 規則の分割

型付け規則 Act はアクターが作られる際に考えられるパターンを複数まとめたものであり，このままの定義を用いると証明を進めることが難しくなってしまう．そこで，具体化のため，ρ の要素数によって場合分けを行い 4 つに分割する．

ここで，ch 関数の入力としてとりうる組は要素数が 0，1，2 のものしかないことに注意されたい．

4.2.2 Case 規則の分割

型付け規則 Case は，前提の数がパターンに依存して変動するので，Coq で定義しにくい．よって 図 5 のように Case をパターンが 0 個の場合と $i+1$ に分割し，帰納的に定義する．これで，Case 規則の前提の数はパターンの数によらず一定となる．

また，この定義によって作られる型付け $(...((\rho_1 \cup \rho_2) \cup \rho_3)... \cup \rho_n) \cup \rho; (...((f_1 \oplus f_2) \oplus f_3)... \oplus f_n) \oplus f \vdash \texttt{case}\ x\ \texttt{of}\ (y : P, y_1 : P_1, ..., y_n : P_n)$ において，元の定義では $f, f_1, f_2, ..., f_n$ が相互に互換性を持つことを条件としているが，$f, f_1, ..., f_n$ が相互に互換性を持つことは自明ではない．よって以下の補題が必要となる．

補題 4.1. 仮名関数 f, f_1, f_2 について，f と $f_1 \oplus f_2$ が互換性を持ち，f_1 と f_2 もまた互換性を持つとき，f, f_1, f_2 は相互に互換性を持つ．

この補題によって，$f, f_1, ..., f_n$ が相互に互換性を持つことが帰納的に保証される．

4.2.3 Res 規則

仮名関数の定義域の制限を行う二項演算子の定義を変更したことに合わせて，型付け規則 Res は 図 6 のようになる．

4.2.4 Inst 規則の分割

Coq による定義では振る舞いの雛形からアクターを作る配置を二つに分割している．よって型付け規則も二つに分割する必要がある．図 7 のようになる．

$$\frac{\emptyset; ch_0 \vdash P}{\{x\}; ch_1(x) \vdash x(y).P} \quad \text{(Act-empty)}$$

$$\frac{\{x\}; ch_1(x) \vdash P \qquad x \neq y}{\{x\}; ch_1(x) \vdash x(y).P} \quad \text{(Act-x)}$$

$$\frac{\{z\}; ch_1(z) \vdash P \qquad x \neq z \qquad y \neq z}{\{x, z\}; ch_2(x, z) \vdash x(y).P} \quad \text{(Act-z)}$$

$$\frac{\{x, z\}; ch_2(x, z) \vdash P \qquad x \neq z \qquad x \neq y \qquad y \neq z}{\{x, z\}; ch_2(x, z) \vdash x(y).P} \quad \text{(Act-xz)}$$

図 4: 型付け規則 Act の分割

$$\frac{}{\emptyset; \{\} \vdash \texttt{case}\ x\ \texttt{of}\ ()} \quad \text{(Case-nil)}$$

$$\frac{\rho; f \vdash \texttt{case}\ x\ \texttt{of}\ (y_1 : P_1, ..., y_i : P_i) \qquad \rho'; f' \vdash P \qquad \mathit{compatible}(f, f')}{\rho \cup \rho'; f \oplus f' \vdash \texttt{case}\ x\ \texttt{of}\ (y : P, y_1 : P_1, ..., y_i : P_i)} \quad \text{(Case-cons)}$$

図 5: 型付け規則 Case の分割

$$\frac{\rho; f \vdash P}{\rho - \{x\}; f|x \vdash (\nu x)P} \quad \text{(Res)}$$

図 6: 型付け規則 Res

$$\frac{\{u\}; ch_1(u) \vdash u(z).P}{\{x\}; ch_1(x) \vdash ((\langle u \rangle, \tilde{v})u(z).P)(\langle x \rangle, \tilde{y})} \quad \text{(Inst-1)}$$

$$\frac{\{u_1, u_2\}; ch_2(u_1, u_2) \vdash u_1(z).P \qquad x_1 \neq x_2}{\{x_1, x_2\}; ch_2(x_1, x_2) \vdash ((\langle u_1, u_2 \rangle, \tilde{y})u_1(z).P)(\langle x_1, x_2 \rangle, \tilde{y})} \quad \text{(Inst-2)}$$

図 7: 型付け規則 Inst の分割

4.3 健全性の証明

Coq で Aπ 計算の健全性を証明する．まず健全性の証明に用いる公理を述べ，健全性の証明に使われるいくつかの補題を証明する．最後に健全性の証明の方針について述べる．

4.3.1 公理

関数の同値性を示す必要がある場合には関数の外延的同値という公理を用いる．これは以下のような公理である．

公理 4.1 (関数の外延的同値)**.** 関数 f, g が任意の入力に対して等しい結果を返すならば f, g は等しい．

この公理は Coq の標準ライブラリで提供されているものを使った．

4.3.2 補題

健全性の証明にはさらに以下の補題が必要となる．

補題 4.2 (定義域と値域の一致)．$\rho; f \vdash P$ ならば，$in\_range\_in\_domain(f)$ が成り立つ．

補題 4.3 (仮名関数の定義域と窓口集合の一致)．$\rho; f \vdash P$ ならば，任意の名前 x について $in\_domain(f, x)$ であるとき，かつそのときに限り，$x \in \rho$ が成り立つ．

補題 4.4 ($fun\_exclusive$ における集合の一致)．$\rho_1; f_1 \vdash P_1$ かつ $\rho_2; f_2 \vdash P_2$ であり $\rho_1 \cap \rho_2 \neq \emptyset$ ならば，$fun\_exclusive(f_1, f_2)$ を満たす．

4.3.3 証明

健全性は，以上の補題を使って型付け規則の構造に関する帰納法で証明できる．Coq による定義と証明の全文は GitHub リポジトリ [1] で閲覧可能である．

5 結論

Aπ 計算における型システムの定義を Coq で証明を行いやすいように再定義し，Aπ 計算の型付けの健全性を形式的に証明した．これによって Aπ 計算の型システムが確かにアクターとしての振る舞いを強制することを示した．

謝辞 しゃじ

参 考 文 献

[1] : A formalization of Aπ, `https://github.com/amutake/a-pi`.

[2] : The Coq Proof Assistant, `http://coq.inria.fr/`.

[3] Agha, G. and Thati, P.: An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language, *From Object-Orientation to Formal Methods*, Lecture Notes in Computer Science, Vol. 2635, Springer-Verlag, 2004, pp. 26–57.

[4] Agha, G. A.: *ACTORS - A Model of Concurrent Computation in Distributed Systems*, MIT Press series in artificial intelligence, MIT Press, 1990.

[5] Gonthier, G.: A computer-checked proof of the Four-Colour Theorem, 2004.

[6] Milner, R.: *Communicating and Mobile Systems: The π-calculus*, Cambridge University Press, New York, NY, USA, 1999.