代数付きπ計算によるデータベースの記述と検証

清野貴博 †, 竹内泉 †, 宮本賢治 †⋆
Takahiro Seino, Izumi Takeuchi, Kenji Miyamoto
seino-takahiro@aist.go.jp, takeuti@ni.aist.go.jp, miyamoto-kenji@aist.go.jp

† 産業技術総合研究所 (AIST), システム検証研究センター (CVS)
⋆ 京都大学大学院情報学研究科

本研究では，代数付きπ計算を使って，データベースの振る舞いの検証を行なった．代数付きπ計算は状態遷移とデータの流れとを統合的に記述する計算体系である．この代数付きπ計算を使うことによって，データベースの状態遷移とデータの流れとの双方に言及した検証項目の簡潔な記述ができるようになった．検証には証明支援系 Agda を用いた．

1 はじめに

本研究では，竹内によって提案された代数付きπ計算 [5] に基づき，データを表す代数仕様の項と状態遷移を統合的に扱い，計算機の上で形式的な仕様検証を行った．

従来の研究では，データを表す項と状態遷移は統合的には扱われておらず，データ項の検証と状態遷移の検証 [2] が別々に行われてきた．結果，従来の検証技法では,「プロセスの遷移によってデータを蓄積した結果が仕様を満たしている」「プロセスによってデータを取り出した結果が仕様を満たしている」等の検証項目を簡潔に記述することはできなかった．

例 1.1 (一階述語論理でのデータベース仕様記述)．プロセスによってデータベースへの書き込みが行われる様子を一階の述語論理で記述すると，次のようになる．

- trans(write(key, dat), db(storage), db(add(storage, key, dat)))

これは，プロセス write(key,dat) と db(storage) が平行して実行された結果，storage にデータが追加されて，プロセス db(add(storage, key, dat)) に遷移することを述語 trans によって記述している．この遷移が仕様を満たすことを検証しようとすると，例えば体系に算術を加えてデータ型と遷移の長さに関する多重帰納法によるところとなり，煩雑な記述が要求される．

例 1.2 (π計算でのデータベース仕様記述)．通常のπ計算で上記の仕様記述を行う場合，π計算が名前のみをデータとして扱うことが問題を複雑にする．解決法としては，データをプロセスによってエンコードする手法 [8] が知られているが，記述は複雑にならざるをえない．

本研究のような，データと状態遷移が関連したシステムの仕様記述に対しては，データを表す代数の項を持つ代数付きπ計算が有効である．

例 1.3 (代数付きπ計算での仕様記述)．代数付きπ計算で上記の仕様記述を行うと，次のようになる．

- $write(key,\ dat)*db(storage) \xrightarrow{\tau} db(add(key,\ dat,\ storage))$

この遷移が仕様を満たすことを検証するために必要なのは，二回の遷移関係の記述のみであり，一階述語論理で記述した場合や，データをプロセスでエンコードしたπ計算の場合と比較すると簡潔である．

よって，代数付きπ計算はデータベース検証と相性の良い計算体系と考えられるので，証明支援系 Agda を用いて実装を行った．

2 データベースの仕様

データベースはデータの書き込みと読み出しの，二つの操作を受け付ける仕様になっている．

読み書きの際には，データに関連付けられた鍵を与えることになっており，次の図 1 のように書き込む際には鍵とデータ本体を渡し，読み出す際には鍵を渡してデータ本体を受け取る．同じ鍵を用いて別のデータを書き込んだ場合，後で書き込んだデータのみを読み出すことができる．

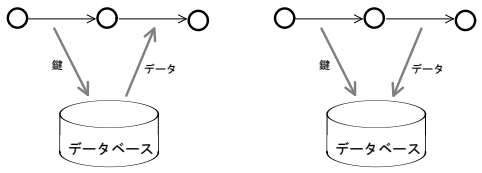

図 1: データベースからの読み出しと書き込み

図 1 の小さな白丸はプロセスを，黒い矢印はプロセス遷移を，太い灰色の矢印はデータの流れを表している．左図の場合，一回目の遷移でデータベースに鍵を与え，二回目の遷移でデータベースからデータを受け取る様子を示している．右図では，二回目の遷移でデータベースにデータを書き込んでいる．代数付き π 計算では，このときのデータの流れだけでなく，データベースの内容が変更されることをも同時に記述し，検証することができる．

3 代数付き π 計算

代数付き π 計算は，π 計算に代数仕様の項を付け加えたものである．代数仕様の項は π 計算のプロセスでエンコードできるので，代数付き π 計算は π 計算の計算論的な拡張とはなりえないが，具体的な項を簡潔に記述する用途においては実用性がある．

定義 3.1 (名前対称代数)．名前対称代数とは，項の等式を一階述語論理で導くことのできる代数で，任意の置換 σ に対して，項 t, t' について $t = t'$ と $t\sigma = t'\sigma$ が同値であり *(*対称性*)*，かつ等式が真であるか偽であるかのどちらかである *(*決定性*)* ものを指す．

この代数の典型例として，完備な項書換え系が挙げられる．また，対称性と決定性は Nepi[7] に従い，決定性を満たすために公理は無限であるとする．

定義 3.2 (代数付き π 計算)．名前対象代数 A があるとき，π 計算に A の項を付け加えた代数付き π 計算を以下のように定義する
変数 x：代数 A の変数
名前 c：代数 A の名前
値項 t：代数 A の項
プロセス変数 X：*0* 以上の各引数の個数に対して，十分多くある
プロセス項 $P ::= 1 \mid P * P \mid (c)P \mid Xt_1 \dots t_n \mid G$
ガード項 $G ::= \mathsf{out}\ tt'P \mid \mathsf{in}\ txP \mid \tau P$
$\mid G + G \mid \mathsf{if}\ tG$
$\mid (\lambda x_1 \dots x_n.\rho X.G)t_1 \dots t_n$

次に，項の構造等価の概念を導入する．構造等価の関係は，プロセス項やガード項上の同値関係である．

定義 3.3 (構造等価)．構造等価とは，以下の規則から生成される同値関係である．

- $P * Q \sim Q * P$, $(P * Q) * R \sim P * (Q * R)$, $P \sim P * 1$,

- $P \sim P + P$, $P + Q \sim Q + P$,
- $(P + Q) + R \sim P + (Q + R)$
- $(\lambda\vec{x}.\rho X.P)\vec{t} \sim P[\vec{t}/\vec{x}, (\lambda\vec{x}.\rho X.P)/X]$
- $(c)P \sim P$, ただし c は P に現れない.
- $(c)P[Q] \sim P[(c)Q]$, ただし P 中で c は $P[X]$ に現れず，また $P[X]$ 中で X は ρ の有効範囲の中に現れない.

例 3.4 (構造等価). *$(\lambda x.\rho X.in\ c\ x\ X)t \sim in\ c\ t\ (\lambda x.\rho X.in\ c\ x\ X)$*
$P[Q[(c)R]] \sim (c)[P[Q[R]]]$ ただし，*c* は *$P[Q[X]]$* 中に現れず，また *$P[Q[X]]$* 中で *X* は ρ の有効範囲の中にない.

π 計算の遷移は，遷移要素と遷移規則によって定められる.

定義 3.5 (遷移要素). 遷移要素とは，次の三種の形をしたものをいう.

- 出力　　$\mathsf{out}\ cv$
- 入力　　$\mathsf{in}\ cv$
- 内部　　τ

$\mathsf{out}\ cv$, $\mathsf{in}\ cv$ は，それぞれ文献 [8] の $\bar{x}y$，xy に相当する.

定義 3.6 (遷移規則). プロセス項 P から Q への遷移要素 α による遷移を $P \xrightarrow{\alpha} Q$ と書く. 遷移は以下の規則によって定義される.

$$\textit{Tau:}\ \frac{}{\tau P \xrightarrow{\tau} P} \qquad \textit{Input:}\ \frac{}{\mathsf{in}\ cxP \xrightarrow{\mathsf{in}\ cv} P[v/x]} \qquad \textit{Output:}\ \frac{}{\mathsf{out}\ cvP \xrightarrow{\mathsf{out}\ cv} P}$$

$$\textit{Com:}\ \frac{P \xrightarrow{\mathsf{in}\ cv} Q \quad P' \xrightarrow{\mathsf{out}\ cv} Q'}{P * P' \xrightarrow{\tau} Q * Q'} \qquad \textit{Res:}\ \frac{P \xrightarrow{\alpha} Q}{(c)P \xrightarrow{\alpha} (c)Q}\ \text{但し } c \text{ は } \alpha \text{ に現れない.}$$

$$\textit{Par:}\ \frac{P \xrightarrow{\alpha} Q}{P * R \xrightarrow{\alpha} Q * R} \qquad \textit{Non-Det:}\ \frac{P \xrightarrow{\alpha} Q}{P + R \xrightarrow{\alpha} Q} \qquad \textit{If:}\ \frac{P \xrightarrow{\alpha} Q}{\mathsf{if}\ tP \xrightarrow{\alpha} Q}\ \text{但し } A \vdash t = \mathsf{true}$$

$$\textit{Calc-I:}\ \frac{\mathsf{in}\ cxP \xrightarrow{\alpha} Q}{\mathsf{in}\ txP \xrightarrow{\alpha} Q}\ \text{但し } A \vdash t = c \qquad \textit{Calc-O:}\ \frac{\mathsf{out}\ cvP \xrightarrow{\alpha} Q}{\mathsf{out}\ tt'P \xrightarrow{\alpha} Q}\ \text{但し } A \vdash t = c \text{ かつ } A \vdash t' = v$$

$$\textit{Structural:}\ \frac{P \xrightarrow{\alpha} Q}{P' \xrightarrow{\alpha} Q'}\ \text{但し } P \sim P' \text{ かつ } Q \sim Q'$$

例 3.7 (遷移の導出). 次の例は，平行して走る 2 つのプロセス間で，データの受け渡しが行われる遷移を記述したものである.

$$\frac{\dfrac{\mathsf{in}\ cxXx \xrightarrow{\mathsf{in}\ ct} Xt \quad \mathsf{out}\ ct1 \xrightarrow{\mathsf{out}\ ct} 1}{(\mathsf{in}\ cxXx) * (\mathsf{out}\ ct1) \xrightarrow{\tau} Xt * 1}}{(\mathsf{in}\ cxXx) * (\mathsf{out}\ ct1) \xrightarrow{\tau} Xt}$$

4 代数付きπ計算でのデータベースの記述

代数付きπ計算で，本論文で扱うデータベースの仕様が，どのように記述されるのかを述べる．

まず，データベースに鍵を与えて，データを取り出すためのプロセスの仕様を記述する．

$read\ x\ X = \mathsf{in}\ rch\ key\ (\mathsf{out}\ rch\ (lookup\ key\ x)\ (X\ x))$

ここで，rch は読み出し用のチャネル名，key は鍵の名前，$lookup$ はデータベースから該当するデータを取り出す関数を指す．

次に，データベースに鍵とデータを与えて，データを書き込むためのプロセスの仕様を記述する．

$write\ x\ X = \mathsf{in}\ wch\ key\ (\mathsf{in}\ wch\ data\ (X\ (append\ key\ data\ x)))$

ここで，wch は書き込み用のチャネル名，$append$ はデータベースにデータを書き込む関数を指す．

これらを共に備えたデータベースシステムは，次のように定義した．

$server\_db = \lambda x.\rho X.(read\ x\ X + write\ x\ X)$

例 4.1 (データベースからのデータ読み出し)．データベース DB から，鍵 key に関連付けられたデータ dat を読み出すことは，次のプロセス遷移によって記述される．

$$server\_db\ DB \overset{\mathsf{in}\ rch\ key}{\rightarrow} \mathsf{out}\ rch\ (lookupkey\ DB)\ (server\_db\ DB) \overset{\mathsf{out}\ rch\ dat}{\rightarrow} server\_db\ DB$$

遷移の導出は，次の通りである．マクロは適宜展開している．

$$\frac{\dfrac{}{read\ DB\ server\_db \overset{\mathsf{in}\ rch\ key}{\rightarrow} \mathsf{out}\ rch\ (lookup\ key\ DB)\ (server\_db\ DB)}}{\dfrac{read\ DB\ server\_db + write\ DB\ server\_db \overset{\mathsf{in}\ rch\ key}{\rightarrow} \mathsf{out}\ rch\ (lookup\ key\ DB)\ (server\_db\ DB)}{\lambda x.\rho X(read\ x\ X\ + write\ x\ X)\ DB \overset{\mathsf{in}\ rch\ key}{\rightarrow} \mathsf{out}\ rch\ (lookup\ key\ DB)\ (server\_db\ DB)}}$$

$$\frac{}{\mathsf{out}\ rch\ (lookup\ key\ DB)\ (server\_db\ DB) \overset{\mathsf{out}\ rch\ dat}{\rightarrow} server\_db\ DB}$$

5 Agda の概要

Agda[1] は Martin-Löf の型理論 [6] の実装である．我々は，Martin-Löf の型理論の流儀で証明を記述し，Agda によって証明が正しいかどうかを検査することができる．証明の検査は，項が正しく型付けされているかどうかを検査することにより実行される．Agda のコード例を以下に記述する．

```
Prop :: Type = Set

example1 (A,B::Prop) :: A->(B->A)
    = \(h::A) -> \(h'::B) -> h
example2 (A,B,C::Prop) :: (A->(B->C))->((A->B)->(A->C))
    = \(h1::A -> B -> C) -> \(h2::A -> B) -> \(h3::A) -> h1 h3 (h2 h3)
```

このコードは，それぞれ論理式

1. $A \supset (B \supset A)$
2. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$

の証明となっている．構文は Haskell に似ていて，Agda での意味を簡単に説明すると，

(識別子と引数宣言) :: (型) = (証明)

という式で，(識別子) に (型) が付くことを表わす．

上記のコードは Agda の型検査を通るが，(証明) の部分に誤りがあればエラーが検出され，型検査は通らない．また，Agda は対話型証明検証系として，ユーザが Agda にヒントを与えながら，証明を構築していくことが可能である．

6 Agda での代数付きπ計算の実装

本節では，代数付きπ計算の Agda での実装について述べる．

6.1 プロセス項とガード項

π計算のプロセス項 `Process` とガード項 `Guarded` は，Agda で次のように記述される．

図 2: `Process` と `Guarded` の定義

```
data Process' (X::Set) =
                end
              | (*) (P::Process X) (Q:Process X)
              | new (CP::Chennel -> Process X)
              | g2p (G::X)

data Guarded =
                tau (P::Process Guarded)
              | inp (T::Term) (T'::Term -> Process Guarded)
              | out (T::Term) (T:Term) (P::Process Guarded)
              | (+) (G::Guarded) (H::Guarded)
              | if_ (T::Term) (G::Guarded)
              | rec (TPG::Term -> (Term -> Process Guarded) -> Guarded)
                     (T::Term)

Process :: Set = Process' Guarded
```

ここで，`(*)` などは中置記法の演算子をあらわし，`*`は *Par* をあらわす．他の記法は，一般的なπ計算と同じである．

6.2 構造等価

構造等価をあらわす述語 `Equiv` は，帰納的データ型を用いて次のように記述される．`Equiv` *p1 p2* で，*p1* と *p2* が構造等価であることをあらわす．

図 3: Equiv の定義 (抜粋)

```
idata Equiv :: Process -> Process -> Set where
        Ref (P::Process) :: Equiv P P
        ComPar (P, Q::Process) :: Equiv (P*Q) (Q*P)
        IdmSum (G::Guarded) :: Equiv (g2p (G+G)) (g2p G)
        Rec (TPG::Term -> (Term -> Process) -> Guarded) (T::Term)
          :: Equiv (g2p (rec TPG T)) (g2p (TPG T (\(x::Term) -> g2p rec (TPG x)))
```

6.3 遷移

遷移要素は次のように記述される.

図 4: 遷移要素 TransElt

```
data TransElt = eTau
              | eInp (C::Channel) (V::Value)
              | eOut (C::Channel) (V::Value)
```

上記の遷移要素を用いて，遷移関係 Evolve(抜粋) とその連続的関係 Evolves は次のように定義される.

図 5: 遷移関係 Evolve

```
idata Evolve :: TransElt -> Process -> Process -> Set where
       Tau (Q::Process)
                :: Evolve eTau (g2p (tau Q)) Q
       Inp (c::Channel) (v::Value) (TQ::Term -> Process)
                :: Evolve (eInp c v) (g2p (inp (c2t c)) TQ) (TQ (v2t v))
       Out (c::Channel) (v::Value) (Q::Process)
                :: Evolve (eOut c v) (g2t (out (c2t c)) (v2t v) Q) Q
       Sum (e::TransElt) (P, R::Guarded) (Q::Process)
           (E::Evolve e (g2p P) Q)
                :: Evolve e (g2p (P + R)) Q
       Eqv (e::TransElt) (P, Q::Process) (P', Q'::Process)
           (E::Evolve e P' Q') (Eq1::Equiv P P') (Eq2::Equiv Q Q')
                :: Evolve e P Q
```

図 6: 遷移関係の連続的関係 Evolves

```
idata Evolves :: (List TransElt) -> Process -> Process -> Prop where
   Final (P::Process) (GOAL::Process) (Eqv::Equv P GOAL)
     :: Evolves Nil P GOAL
   Succs (e:TransElt) (P::Process) (Q::Process) (GOAL::Process)
         (E::Evolve e P Q) (elts::List TransElt)
         (EVS::Evolves elts Q GOAL)
     :: Evolves (e : elts) P GOAL
```

7 データベースの仕様検証

7.1 Agda における仕様記述

先に挙げたπ計算によるデータベースの仕様記述は，Agda においては以下のように記述される．

図 7: データベースの仕様記述

```
read_db :: AssocList EqString Value -> (Term -> Process) -> Guarded
  = \(DB::AssocList EqString Value) -> \(TPG::Term -> Process) ->
    ginp (c2t rch) (\(key::Term) -> pout (c2t rch)
      (lookupWrapper key DB) (TPG (v2t (db DB))))

write_db :: AssocList EqString Value -> (Term -> Process) -> Guarded
  = \(DB::AssocList EqString Value) -> \(TPG::Term -> Process) ->
    ginp (c2t wch) (\(key::Term) -> pinp (c2t wch) (\(dat::Term) ->
      TPG (v2t (db (append (v2s (reduce key)) (reduce dat) DB)))))

server_db :: Term -> Process
  = \(DB::Term) ->
    g (rec (\(T::Term) -> \(TP::Term -> Process) ->
      read_db (v2d (reduce T)) TP + write_db (v2d (reduce T)) TP)
    (DB))
```

コードを代数付きπの項として理解するために必要な型と関数のうち，主なものの説明は以下の通りである．

1. `AssocList EqString Value`：データベースが扱うデータである，文字列の連想リストの型
2. `Term`：代数付きπ計算の項の型
3. `Process`：代数付きπのプロセスの型
4. `Guarded`：代数付きπのガード項の型
5. `ginp`：入力を行うガード項
6. `pinp`：入力を行うプロセス項

7. pout：出力を行うプロセス項
8. g：ガード項をプロセス項に変換する関数
9. rec：再帰を記述するための項
10. append, lookupWrapper：データベースの操作を行う関数

7.2 仕様検証

本研究で仕様検証した検証項目は以下の通りである．

1. server_db にデータを入力する遷移を行わせたとき，データベースに正しくデータが書き込まれる
2. server_db とデータを出力するプロセスが平行して走っていたとき，データが正しくデータベースに書き込まれる
3. server_db にデータを出力する遷移を行わせたとき，データベースから正しくデータを取りだすことができる
4. server_db とデータを入力するプロセスが平行して走っていたとき，データが正しくデータベースから取りだされる

また，上記の仕様を Agda で記述すると図 8 のようになる．

証明の記述量は，1. 52 行, 2. 67 行, 3. 30 行, 4. 49 行, となった．

8 関連研究

本節では，データベースやπ計算の検証に関する研究と本研究の関連について述べる．

本研究では単純なシステムを検証の対象としたが，実用的なデータベースであるアクティブデータベースを対象とした研究として，Choi らの研究 [3] が挙げられる．アクティブデータベースとは，データベースへのクエリーなどを中心とする外部からのイベントだけでなく，あらかじめ記述されたルールに従って，内部で起こるイベントに対しても自動的にデータ処理を行うことができるシステムのことである．Choi らはアクティブデータベースを検証の対象とし，与えられたルールに対するデータベースの動作の停止性を，モデル検査によって検証する手法を提案した．代数付きπ計算の枠組みでは，ルールによるデータベースの動作は，プロセスの遷移として記述することが可能である．かつ代数付きπ計算では，代数項によってデータの内容を詳細に記述することが可能であるので，検証項目がより詳細な記述になっても対応する記述力を持つ．しかし，代数付きπ計算の項をモデル検査できる検査器が存在しないため，Choi らの検証項目を含め，モデル検査の技法を用いて行われているような検証を行うことはできていない．代数付きπ計算でもモデル検査ができるようなモデル検査器の開発が期待される．

π計算を用いて情報流を解析する計算体系として，Kobayashi による型を用いた情報流解析の研究 [4] がある．Kobayashi の体系では，デッドロックが発生しないことや，通信や同期が適正であることを，静的な型付けによって保証することができる．さらに型推論のアルゴリズムを与えて，型推論の健全性と完全性を示した．代数付きπ計算では Kobayashi の体系とは異なり，代数項によってデータの詳細な記述が可能であるが，情報流解析に関しては研究の余地がある．

9 今後の課題

本研究はデーターベースを検証の対象として扱ったが，今回対象としたシステムは仕様が単純なもので，機能も限られていた．代数付きπ計算はより大きなシステムの記述も可能であるので，実務で用いられているデータベースを検証することを今後の目標としている．また，代数付きπ計算との親和性が高く，かつ

図 8: 検証項目の仕様記述

1.

```
proof_when_write :: Evolves
  [eInp wch (str "N0006"), eInp wch (str "Gyorgy")]
    (server_db (v2t (db DB1)))
    (server_db (v2t (db (append "N0006" (str "Gyorgy") DB1))))
```

2.

```
proof_when_write_par :: Evolves
  [eTau, eTau]
    (server_db (v2t (db DB1)) * write_test)
    (server_db (v2t (db (append "N0005" (str "Feldman") DB1))))

write_test :: Process = pout (c2t wch) (v2t (str "N0005"))
  (pout (c2t wch) (v2t (str "Feldman")) end)
```

3.

```
proof_when_read :: Evolves
  [ eInp rch (str "N0001"), eOut rch (str "Akiyama") ]
    (server_db (v2t (db DB1)))
    (server_db (v2t (db DB1)))
```

4.

```
proof_when_read_par :: Evolves
  [ eTau , eTau ]
    (server_db (v2t (db DB1)) * read_test)
    (server_db (v2t (db DB1)))

read_test :: Process = pout (c2t rch) (v2t (str "N0001"))
  (pinp (c2t rch) (\(h::Term) -> end))
```

検証の対象となりうるものの具体例として，ワークフロー図のようにプロセスがデータを送受信する対象を記述したもの等を見付けた．よって，実務で用いられているワークフロー図が仕様を満たしているかを検証することも，将来の課題としては適当であると位置付けている．

本研究で用いた代数付きπ計算は，記述力が既存のπ計算や一階述語論理と比較すると強力になっているので，紙の上では証明の記述が簡潔になる．しかし実際に計算機上で証明を実装したところでは，予想ほど簡潔な証明記述にはならず，むしろ煩雑で労力のかかる作業となった．今回の実装では証明支援系として Agda のみを用いたが，他の検証系によっては少ない労力で記述ができる可能性もありうる．証明の形式的検証において，記述にかかる労力は大きな要素なので，Coq や Isabelle など他のシステムで実装することによって証明記述の簡潔さがどのように変わってくるのかを考えたい．

また，代数付きπ計算で情報流解析を行う手法の一つとして，型理論を用いることは有効な手段であると考えられる．現時点では代数付きπ計算には型が付いていないので，体系に型を導入することも今後の課題である．

参考文献

[1] `http://www.cs.chalmers.se/~catarina/agda/`

[2] 崔銀惠, 河本貴則, 渡邊宏, 画面遷移仕様のモデル検査, コンピュータソフトウェア*, vol. 22, no. 3*, 岩波書店, pp.146-153, July 2005.

[3] Eun-Hye Choi, Tatsuhiro Tsuchiya, Tohru Kikuno.: Model Checking Active Database Rules, *Programming Science Technical Report AIST-PS-2006-001*, AIST CVS, 2006.

[4] Kobayashi, N.: Type-Based Information Flow Analysis for the Pi-Calculus, *Acta Informatica, Vol. 42, No. 4–5*, pp. 291–347, 2005.

[5] 竹内泉, パイ計算による仕様を検証する論理体系, 情報処理学会論文誌：プログラミング*, Vol. 46 No.SIG 11 (PRO 26)*, 情報処理学会, 2005, pp. 57–65.

[6] Martin-Löf, P.: *Intuitionistic Type Theory*, Bibliopolis, 1984.

[7] Mano, K. and Kawabe, Y.: Nepi: syntax, semantics and imprementation, *5th World Multi Conference on Systemics, Cybernetics and Informations* (*SCI '01*), 2001.

[8] Sangiogi, D. and Walker, D.: *The π-calculus, a therory of mobile processes*, Cambridge University Press (2001).