ソフトウェア基礎論　27－9
プログラミング言語　19－9
（1988. 12. 9）

プロセス代数型言語「あ」について
An introduction of a process algebraic language $\mathcal{A}$

戸村 哲、石川 裕、二木 厚吉
SATORU TOMURA, YUTAKA ISHIKAWA, KOKICHI FUTATSUGI
電子技術総合研究所
Electrotechnical Laboratory

あらまし　プロセス代数型言語「あ」（英語名：$\mathcal{A}$）は、Milnerの交信系計算理論（CCS:Calculus for Comminucating Systems）に基づいた並行プロセス記述プログラミング言語である。本論文では、交信系計算理論の並行計算モデルに基づいて「あ」の形式的意味を与える事象関係の証明系を定義し、「あ」の実行系と証明系の関係について考察をおこなう。

Abstract　A process algebraic language $\mathcal{A}$ is introduced. $\mathcal{A}$ is based on Milner's CCS (Calcullus for Comminucating Systems) and is designed for describing concurrent processes. In this paper the formal semantics of $\mathcal{A}$ is given in terms of the inference system of the event relations. The relations between the executing system and the formal semantics system are also discussed.

1　はじめに

プロセス代数型言語「あ」（英語名：$\mathcal{A}$）は、並行プロセス系の記述、実現、検証を目的としたプログラミング言語である。「あ」の並行計算モデルは交信系計算理論（CCS: Calculus for Comminucating Systems[1]）の並行計算モデルに基づいている。交信系計算理論の並行計算モデルは形式的意味体系を持ち、プログラムの同値関係、合同関係について理論的研究 [2,4,5] の進んだ体系である。従って交信系計算理論はこれらの研究成果に基づくプログラム変換法などの並行プログラムの理論的扱いに最も適した理論の一つである。「あ」は交信系計算理論に基づいてプログラミング言語としての拡張を行なったものである。

本論文では、交信系計算理論の並行計算モデルに基づいて「あ」の形式的意味を与える事象関係の証明系を定義する。そして「あ」の実行系と証明系の関係について考察をおこなう。

2　並行計算モデル

「あ」の並行計算モデルである交信系計算理論の並行計算モデルについて簡単に説明する。このモデルでは交信系全体は**プロセス**の集まりである。各プロセスはそれぞれの状態 Bを持ち、状態を次々と変化させる。状態が変化した時には状態変化に対応する**事象** μが観測される。プロ

セス間の協調動作は次のようにモデル化する。2つのプロセスが状態 B_1, B_2からそれぞれ事象 μ_1, μ_2を起こして状態 B_1', B_2'に遷移する場合で、この事象の組 $\langle \mu_1 \mu_2 \rangle$が特別な関係：**共役関係**にある時には、2つの事象は融合してそれぞれの事象の代わりに特別な事象 tauが観測されて状態遷移が起こる。このようにして交信系全体の状態が遷移するモデルが交信系計算理論での並行計算モデルである。

この並行計算モデルでプロセスの状態を表現する式を**挙動式** (behaivor expression) と呼ぶ。挙動式はそのプロセスが今後起こす可能性のある事象とそれらの事象を起こした後のプロセスの状態を表現するものである。状態 B_1のプロセスが状態 B_2に遷移した時事象 μが観測されるという関係を $B_1 \xrightarrow{\mu} B_2$と表現する。関係 $\xrightarrow{\mu}$ ($\subseteq$ 状態 $\times$ 事象 $\times$ 状態) を**事象関係**と呼ぶ。

挙動式は事象または挙動式を演算子で結合したもので表現される。交信系計算理論で提案された基本的演算子には、連結演算子、選択演算子、合成演算子、制限演算子、置換演算子がある。これらの演算子の意味は、演算子を含む挙動式に関する事象関係を推論する推論規則として与えられる。挙動式の意味はこれらの推論規則から構成される証明系によって定められる。

3 プロセス代数型言語「あ」

「あ」は前節で述べた並行計算モデルに基づく並行プログラミング言語である。記述能力を向上させるために挙動式の基本演算子に同時事象演算子を追加した。また「あ」のプログラムが扱うデータである値集合に関する条件判定機能を一般的な枠組で導入するために条件事象を事象として追加した。さらに挙動式での引数の記述法をできるだけ一般的なものとして表現力を向上させている。次に「あ」の構文規則を示す。

＜プログラム＞ ::= ＜挙動定義＞ ...＜挙動式＞.
＜挙動定義＞ ::= ＜名前＞＜引数＞:-＜挙動式＞.
＜挙動式＞ ::= ＜挙動式 $_1$＞ | ＜挙動式 $_1$＞; ＜挙動式＞
＜挙動式 $_1$＞ ::= ＜挙動式 $_2$＞ | ＜挙動式 $_2$＞&＜挙動式 $_1$＞
＜挙動式 $_2$＞ ::= ＜挙動式 $_3$＞ | ＜挙動式 $_3$＞,＜挙動式 $_2$＞
＜挙動式 $_3$＞ ::= ＜挙動式 $_4$＞ | ＜挙動式 $_3$＞\＜集合＞ | ＜挙動式 $_3$＞/＜置換＞
＜挙動式 $_4$＞ ::= (＜挙動式＞) | ＜事象＞ | ＜挙動呼出し＞
＜挙動呼出し＞ ::= ＜名前＞＜引数＞
＜事象＞ ::= ！＜事象名＞＜引数＞ | ？＜事象名＞＜引数＞ | ＜事象＞ ^＜事象＞
| tau | ＜条件事象＞
＜項＞ ::= ＜数＞ | ＜文字＞ | ＜文字列＞ | ＜変数＞ | ＜構成子名＞＜引数＞
＜引数＞ ::= ＜空＞ | (＜項＞ ,...,＜項＞)
＜置換＞ ::= {＜置換対＞,...,＜置換対＞}
＜置換対＞ ::= ＜事象名＞ /＜事象名＞
＜集合＞ ::= {＜事象名＞ ,...,＜事象名＞ }
＜空＞ ::=
＜条件事象＞ ::= 項に関する条件式
＜事象名＞ ::= 小文字で始まる識別子
＜名前＞ ::= 小文字で始まる識別子
＜構成子名＞ ::= 小文字で始まる識別子
＜変数＞ ::= 大文字で始まる識別子

＜数＞ ::= 整数
<文字> ::= 文字定数
<文字列> ::= 文字列

「あ」の＜プログラム＞は＜挙動定義＞の並びと＜挙動式＞とから構成される。＜挙動定義＞は挙動に名前を付け、挙動式の中から呼び出せるようにする宣言である。実行系は＜プログラム＞の＜挙動定義＞の宣言に基づいて最後の＜挙動式＞を実行する。

＜項＞は「あ」のプログラムが扱うデータである値集合であり、変数集合 ＜変数＞ 上の項である。変数 X ∈ ＜変数＞ は値として ＜項＞ の要素である項を取るものである。

「あ」の形式的意味は事象関係に関する証明系として定義される。次節では「あ」の演算子の説明をすると同時に証明系の推論規則を示す。以下の説明ではプログラムが変数を含む場合と含まない場合とを分けて考える。両者を区別するために、変数を含まない挙動式を**基底挙動式**と、変数を含まない項を**基底項**と呼び、基底でない挙動式を**一般挙動式**と、基底でない項を**一般項**と呼ぶことにする。

3.1 「あ」の証明系

「あ」のプログラムの意味は事象関係で与えられる。プログラムの意味を 2段階に分けて与える。まず基底挙動式に関する事象関係を定義し、次に基底挙動式の定義を用いて一般挙動式に関する事象関係を定義する。

推論規則は、

$$\frac{\text{前提条件}}{\text{結論}}$$

の形式で与え、推論規則での記法は次の通りとする。

$$\begin{aligned} B, B_i, B_i' &: \text{挙動式} \\ p &: \text{挙動名} \\ e, e_i &: \text{事象名} \\ V &: \text{引数} \\ \mu &: \text{事象} \\ \lambda &: \text{正事象または負事象} \\ c &: \text{条件事象} \\ S &: \text{事象名の集合} \\ M &: \text{事象名の置換 : 事象名} \mapsto \text{事象名} \\ \sigma &: \text{変数から基底項への置換 : 変数} \mapsto \text{基底項} \end{aligned}$$

証明系の推論規則での事象関係は先に定義した挙動式を拡張した拡張挙動式間で定義される。この拡張挙動式は＜挙動式 $_4$＞の定義を

＜挙動式 $_4$＞ ::= (＜挙動式＞) | ＜事象＞ | ＜挙動呼出し＞ | ϵ

に変更したものである。この ϵ はプロセスの実行が終了した状態に対応するものである。ϵ の意味は推論規則の形式ではなく、次に に示す ϵ を含む拡張挙動式の等式の形で定義する。

$$\epsilon\ , B \equiv B$$
$$\epsilon\ \&B \equiv B \quad B\&\epsilon\ \equiv B$$
$$\epsilon\ ;\ B \equiv \epsilon \quad B;\ \epsilon\ \equiv \epsilon$$
$$\epsilon\ \backslash S \equiv \epsilon \quad \epsilon\ /M \equiv \epsilon$$

3.1.1 基底挙動式の意味

挙動式の構成要素について説明をおこない、基底挙動式に関する事象関係の推論規則を示す。

事象 事象には外部事象、内部事象、条件事象がある。外部事象は事象のモード、事象名、引数から構成される。事象のモードには"!" と"?" とがあり、モードが"!" のものを正事象、モードが"?" のものを負事象という。

内部事象は外部からは事象が起こったことだけが観測でき、事象の詳細が観測できない事象を表現する。例えばプロセスが自立的に状態を遷移させた場合がこの内部事象に対応する。内部事象は事象名が"tau" の事象である。内部事象にはモードはない。

推論規則を簡潔に記述するために事象に関する関数：共役関数 $\overline{\ }$、事象名関数 name、事象引数関数 args を次のように定義する。

$$\overline{x} = \begin{cases} !eV & \text{if } x = ?eV \\ ?eV & \text{if } x = !eV \\ \texttt{tau} & \text{if } x = \texttt{tau} \end{cases}$$

$$\mathrm{name}(x) = \begin{cases} e & \text{if } x = ?eV \\ e & \text{if } x = !eV \\ \texttt{tau} & \text{if } x = \texttt{tau} \end{cases}$$

$$\mathrm{args}(x) = \begin{cases} V & \text{if } x = ?eV \\ V & \text{if } x = !eV \end{cases}$$

条件事象 条件事象とは値に関する条件成立を表現する事象である。例えば条件事象 evenp(X)は Xが偶数である時、そしてその時に限って内部事象を発生させる事象である。条件事象として記述できる値に関する条件は「あ」の外で別に定めるものとする。

「あ」の値集合を拡張する方法として<項> を利用者が定義できる代数的抽象データ型に置き換えることは「あ」の他の部分と独立に行なうことができる。その場合は条件事象の定義を代数的抽象データ型定義で行なうことができる。

$$\frac{\text{true}}{\mu \xrightarrow{\mu} \epsilon} \qquad \frac{c\text{が成立する。}}{c \xrightarrow{\texttt{tau}} \epsilon}$$

連結演算子　連結演算子 ”,” は挙動式と挙動式から挙動式を合成する演算子である。プロセスの逐次実行を表現するものである。挙動式 B_1, B_2 は挙動式 B_1 の動作が終了した後、挙動式 B_2 と同じ状態になる挙動式である。

$$\frac{B_1 \xrightarrow{\lambda} B_1'}{B_1, B_2 \xrightarrow{\lambda} B_1', B_2}$$

選択演算子　選択演算子 ”; ” は限定式と限定式とから挙動式を合成する演算子である。限定式 B とは B を左辺に持つ事象関係 $B \xrightarrow{e} B'$ が唯一つしか存在しない挙動式のことである。挙動式 $B_1;\ B_2$ は限定式 B_1 または 限定式 B_2 のいづれかの挙動をする挙動式である。

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1;\ B_2 \xrightarrow{\mu} B_1'} \qquad \frac{B_2 \xrightarrow{\mu} B_2'}{B_1;\ B_2 \xrightarrow{\mu} B_2'}$$

合成演算子　合成演算子 ”&” は挙動式と挙動式とから挙動式を合成する演算子である。プロセスの並列実行を表現するものである。挙動式 $B_1 \& B_2$ は挙動式 B_1 で観測される挙動と挙動式 B_2 で観測される挙動とがインターリーブして観測される挙動式である。並行計算モデルの説明で述べた通り、合成演算子で結合されたプロセスが互いに共役な事象を起こして状態遷移する時はそれらの事象が融合して２つの事象の代わりに１つの内部事象が観測されることがある。これはプロセス間の同期を表現するものである。

$$\frac{B_1 \xrightarrow{\lambda} B_1' \quad B_2 \xrightarrow{\overline{\lambda}} B_2'}{B_1 \& B_2 \xrightarrow{\mathtt{tau}} B_1' \& B_2'}$$

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1 \& B_2 \xrightarrow{\mu} B_1' \& B_2} \qquad \frac{B_2 \xrightarrow{\mu} B_2'}{B_1 \& B_2 \xrightarrow{\mu} B_1 \& B_2'}$$

同時演算子　同時事象演算子 ”^” は事象と事象から事象を合成する演算子である。事象 $\lambda_1 \hat{} \lambda_2$ は事象 λ_1 と事象 λ_2 とが同時に起こる事象を表す。

$$\frac{B_1 \xrightarrow{\lambda_1 \hat{} \lambda_2} B_1' \quad B_2 \xrightarrow{\overline{\lambda_1}} B_2' \quad B_3 \xrightarrow{\overline{\lambda_2}} B_3'}{B_1 \& B_2 \& B_3 \xrightarrow{\mathtt{tau}} B_1' \& B_2' \& B_3'}$$

制約演算子　制約演算子 ”\” は挙動式と事象名集合から挙動式を合成する演算子である。挙動式 $B \backslash S$ は挙動式 B で観測される事象について、その名前が集合 S に含まれていない場合に限ってその事象が観測される挙動式を表す。但し事象名集合は tau を含まないものとする。

$$\frac{B \xrightarrow{\mu} B',\ \ \mathrm{name}(\mu) \notin S}{B \backslash S \xrightarrow{\mu} B' \backslash S}$$

置換演算子　置換演算子 ”/” は挙動式と置換とから挙動式を合成する演算子である。挙動式 B/M は挙動式 B で観測される事象の事象名を置換 M で置換したものが観測される挙動式を表す。

$$\frac{B \xrightarrow{\mathtt{tau}} B'}{B/M \xrightarrow{\mathtt{tau}} B'/M} \qquad \frac{B \xrightarrow{\lambda} B'}{B/M \xrightarrow{\lambda M} B'/M}$$

ただし λM は

$$\lambda M = \begin{cases} !e_1 V & \text{if } \lambda = !e_2 V \text{ and } e_2/e_1 \in M \\ ?e_1 V & \text{if } \lambda = ?e_2 V \text{ and } e_2/e_1 \in M \\ \lambda & \text{otherwise} \end{cases}$$

である。

挙動定義と挙動呼出し　挙動定義は挙動に名前を付け、繰り返しや再帰的呼び出しなどを可能とするものである。プログラムが挙動定義 pV :- B を含む場合、挙動式 pV は挙動式 B と同じ挙動をする。

$$\frac{pV :\text{-} B \quad B \xrightarrow{\mu} B'}{pV \xrightarrow{\mu} B'}$$

3.1.2　一般挙動式の意味

基底挙動式の意味を用いて一般挙動式の意味を定義する。一般挙動式に出現する変数を基底項に一斉置き換えすることによって得られる基底挙動式の挙動が一般挙動式の挙動である。一般挙動式の事象関係を定義する推論規則は次のようになる。ただし、$B\sigma$ は挙動式 B に置換 σ を適用して得られる挙動式を、$\mu\sigma$ は事象 μ に置換 σ を適用して得られる事象をそれぞれ表すものとする。

$$\frac{\exists\sigma \ B\sigma \xrightarrow{\mu\sigma} B'\sigma}{B \xrightarrow{\mu} B'}$$

3.2　「あ」の実行系

「あ」の実行系は「あ」のプログラムを実行するシステムである。実行系の動作を表現するために実行関係 $\rightsquigarrow$ を次のように定義する。

定義　1 (実行関係) 実行系が挙動式 B_1 を実行した結果、 B_2 に変化したとする。この時実行関係 $B_1 \rightsquigarrow B_2$ が成り立つという。

実行系と証明系の間の関係を明確にするために健全性、完全性、忠実性の 3 つの概念を定義する。

定義　2 (健全性) 実行系が健全であるとは、

$$\forall B_1, \forall B_2 (B_1 \rightsquigarrow B_2 \Rightarrow B_1 \xrightarrow{\texttt{tau}} B_2)$$

が成り立つことである。

定義　3 (完全性) 実行系が完全であるとは、

$$\forall B_1, \forall B_2 (B_1 \xrightarrow{\texttt{tau}} B_2 \Rightarrow B_1 \rightsquigarrow B_2)$$

が成り立つことである。

定義 4 (忠実性) 挙動式族 $\{B_i\}$を Bから tau-遷移できるすべての挙動式とする。つまり、

$$\forall B'(B \xrightarrow{\texttt{tau}} B' \Rightarrow B' \in \{B_i\})$$

とする。このとき、実行系が忠実であるとは、

$$\{B_i\} \neq \phi \Rightarrow \exists i_0(B \rightsquigarrow B_{i_0})$$

が成り立つことである。

健全性の条件は実行系の動作が証明系の定義に基づいていることを保証するものである。健全であることは実行系として最低限の条件である。完全性の条件は実行系が証明系の全ての証明を実行することを保証するものである。しかし、完全性の条件は実行系を実現する立場からは並行プロセスのスケジューリングの公平さを保証することと同じ条件であり、その実現は困難である。また、健全な実行系にはすべてのプログラムに対して何の実行もしない実行系が含まれる。

忠実な実行系とは証明系で tau-遷移が１つ以上存在する場合には実行系は必ず１つは実行を行なうものである。先に述べたすべてのプログラムに対して何の実行もしない実行系は忠実な実行系ではない。

「あ」の言語設計目標の一つは健全で忠実な実行系を効率的に実現可能なことである。実行系の動作環境としてはマルチプロセッサ環境での並行実行を想定している。この想定の下で効率的に実現するために、実行のオーバーヘッドとなる論理変数と後戻り制御を実現せずに実行系を実現したい。そこで 1. 並行動作環境での変数共有の禁止、2. 挙動呼出しの値渡しへの制限、の２つの制約を加えた。これらの制約はいづれも構文的な制約であり静的に制約を満たしていることの判定ができる特徴を持つ。

3.2.1 並行動作環境での変数共有の禁止

「あ」のプログラムの変数の有効範囲は、挙動定義では定義全体であり、実行する挙動式では挙動式全体である。この定義のままで同じ変数名の変数を同一の変数とすると、並行動作環境で変数を共有しなければならない場合がある。例えば p(X) &q(X)の２つの Xを同じ変数とした場合である。そこでこのような変数を別々の変数として扱えるように変数の有効範囲に制約をつける。まず変数の有効範囲の制約を定義するために有効経路という概念を導入する。

有効経路とは変数の有効範囲に対応する挙動式の部分式である。

定義 5 (有効経路) 挙動式 Bに対して、その部分式 B_1 &B_2 を B_1または B_2に置き換えるという手続きを繰り返し行なうことで合成演算子を含まなくなった挙動式 B'を得る。この挙動式 B'を挙動式 Bの有効経路という。

例えば、p(X,Y) :-!p1(X,Z,Z) &!p2(Y,Z,Z) .の有効経路は
p(X,Y) :-!p1(X,Z,Z) . と
p(X,Y):-!p2(Y,Z,Z) . の二つである。

有効経路を用いて変数の定義的出現と参照的出現を定義する。

定義 6 (変数の定義的出現) 挙動式 Bにおける変数 Xの出現が定義的出現であるとは、挙動式 Bの有効経路でその変数出現を含む B'を左から右に順に見て行くと、その変数出現が同じ名前の変数 Xが最初の出現であることである。

定義 7 (変数の参照的出現) 変数の出現が参照的出現であるとは、定義的出現でない変数出現をいう。

次に変数の有効範囲を定義する。

定義 8 (変数の同一性) 挙動式 B の変数 Xの参照的出現は、その参照的出現を含む有効経路 B'の変数 Xの定義的出現と同一の変数である。

例題 p(X,Y) :-!p1(X,Z,Z) &!p2(Y,Z,Z) .の変数で定義的出現に下線をつけると
p(X,Y) :-!p1(X,Z,Z) &!p2(Y,Z,Z) .となる。従って p1の Zと p2の Zとは別の変数であり、pの Xと p1の Xは同じ変数である。この挙動定義に出現する変数名を同じ変数だけが同じ変数名を持つように変更すると、 p(A,B) :-!p1(A,C,C) &!p2(B,D,D) .となる。

この定義ではある変数の参照的出現は複数の定義的出現を持つことがあり、変数共有の原因となる。これを避けるために次の制限を設ける。

制限 1 変数の参照的出現はただ一つの定義的出現を持つ。

このように変数の有効範囲に制約を加えることによって単純な並行動作環境での変数共有を避けることができる。

3.2.2 挙動呼出しの値呼出しへの制約

変数の有効範囲の制約だけでは、次の例の場合にやはり並行動作環境での変数共有が起こる。

例題プログラム p(X) :-tau.
p(X) ,(p1(X) &p2(X))の場合を考える。 この場合、挙動呼出し p(X)の後でも Xは値が定まらず変数のままである。そのため p1(X)と p2(X)とで並行動作環境での変数共有が発生する。こうした事態を避けるため挙動呼出しについて次の制約を設ける。

制限 2 挙動呼出しはすべて値呼出しとする。

ただし値呼出しは次のように定義する。

定義 9 (値呼出し) 挙動呼出しが値呼出しであるとは、挙動呼出しの引数に変数の定義的出現を含まないことである。

変数の有効範囲の定義と挙動呼出しを値呼出しに制限することによって、実行系は論理変数を実現せずにまた後戻りすることなく検算で忠実に実現することができる。この２つの制約条件は [1] での引数形式に関する制約を一般化したものになっている。

4 まとめ

「あ」は交信系計算理論の並行計算モデルに基づく並行プログラミング言語である。「あ」ではプログラム記述能力を向上させるために挙動式の基本演算子に同時事象演算子を追加した。また「あ」のプログラムが扱うデータである値集合に関する条件判定機能を一般的な枠組で導入するために条件事象を事象として追加した。さらに挙動式での引数の記述法をできるだけ一般的なものとして表現力を向上させた。　「あ」の形式的意味を事象関係に関する証明系として定義した。「あ」の証明系と実行系の動作との関係を明確にするために実行系の健全性、完全性、忠実性という概念を導入した。「あ」の実行系を後戻り制御をせずに健全で忠実な実行系とするため制約として、1. 並行動作環境での変数共有の禁止、2. 挙動呼出しを値呼出しに制限する、という制約を付加した。これらの制約は構文上で判断することができる静的な条件であり、[1] で与えられていた引数形式に対する制約を一般化したものである。

参考文献

[1] R. Milner, "A Calculus of Communicating Systems," Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[2] R. Milner, "A Complete Inference System for a Class of Regular Behaviours," Journal of Computer and System Science 28, pp. 439-466, 1984.

[3] R. Milner, "Process Constructors and Interpretations," Information Processing 86, IFIP, 1986.

[4] R. De Nicola, M.C.B. Hennessy, "Testing Equivalences for Processes," Theoretical Computer Science 34, North-Holland, pp. 83 - 133, 1984.

[5] Iain Phillips, "Refusal Testing." Theoretical Computer Science 50, North-Holland, pp. 241 - 284, 1987.