（1988. 5. 26）

μ算法によるＮｅＯの操作的意味について

Operational Semantics of the NeO by μ-calculus

猪股　俊光　　西村　義行

Toshimitsu INOMATA　Yoshiyuki NISHIMURA

豊橋技術科学大学　工学部

Faculty of Engineering, Toyohashi University of Technology

あらまし　：　筆者らが開発中のＮｅＯはメッセージパッシング型計算モデルに基づいた並列処理システムのためのプログラム図式であり、プログラムはラベル付き有向グラフとして記述される。本文では、ＮｅＯに形式的意味を与えるためのＮｅＯから Predicate/Transition-netsへの変換法および Predicate/transition-nets からμ算法への変換法について論じる。これらの変換を行うことにより、ＮｅＯのプログラムの意味を μ算法における事象間の因果関係として与えることが可能となった。

Abstract　：　The NeO is a program schema , which has been developed by the authors, for a variety of parallel processing systems on the basis of a message-passing computation model . NeOs are described by labeled directed graphs . We propose in this paper a method of translating a NeO program into Predicate/transition-nets and that of translating the Predicate/transition-nets into objects of μ-calculus , to describe the formal semantics of the NeO . As a result of this research , a semantics of executions of the NeO program can be given by the relation of causality between events which are the sequence of objects in μ-calculus .

1．は　じ　め　に

コンピュータ技術の発達により、産業、経済、学問などあらゆる分野で計算機が盛んに利用されるようになっている。ＯＡやＦＡシステムなどに代表されるさまざまな情報システムでは、システムの大規模化、複雑化、分散化、ネットワーク化が進められている。多くの情報システムは、いくつかの処理実体（プロセッサ）が通信を行いながら並列に独自の処理を行っている並列処理システムとしてとらえることができる。このような多数のプロセッサからなる並列処理システムでは、これらをうまく制御するための新しいソフトウェア技術を開発していくことが重要であり、新しいプログラミング方法論や並列型計算モデル、並列プログラミング言語に関する研究が行われている[1]。

筆者らは、並列処理システムの解析・設計を体系的に行うための手法の確立を目指しており、並列処理システムの解析・設計を「実行可能仕様＋変換」というプログラミング・パラダイム[2] に基づいて行うための計算モデル、記述言語、変換技法の開発を進めている。このプログラミング・パラダイムは、「実行可能仕様」に対して「変換」を施しながら最終的なプログラムを作り出すという方式であり、これに基づきこれまでに、計算モデルとしてオブジェクト指向プログラミングの基礎となっているメッセージパッシング[3] 型の計算モデルを考え、このモデルに基づいた記述言語ＮｅＯ[4] および処理系（エディタ・デバッガ・シミュレータ）[5,6] の開発を行っている。ＮｅＯはオブジェクト指向プログラミングのためのプログラム図式でもあり、プログラムはラベル付き有向グラフ（ネット）によって記述される。また、計算の状態は特定のノードに対するマーキングとして表現される。そして、このＮｅＯを実行可能仕様とみなし、ＮｅＯに対しプログラム変換を施しながら解析・設計に必要な性質を求めるための変換技法に関する研究も進めている [4]。プログラム変換は、プログラムの意味を保ちながら行

わなければならず、プログラムの意味を明確に定めておく必要がある。このため、筆者らは、マーキングがどのように変化するかということによりＮｅＯのプログラムの意味を定め、変換の前後におけるマーキングの変化の等価性により変換規則の正当性を保証することを試みた[4]。

今回は、より形式的なＮｅＯの意味を定めるために、メッセージパッシング型計算モデルに関連のあるμ算法[7]を用いてＮｅＯに操作的意味を与えることを試みる。これは、ＮｅＯのプログラムをμ算法に基づいて書かれたプログラムに変換することにより、ＮｅＯの意味をμ算法に基づいた抽象機械（μ算法インタプリタ）の動作によって記述しようとするものである。これを実現するため、まず、ＮｅＯからグラフモデルの一つである Predicate/transitionネット[8]（以下 Pr/Tネットと記す）への変換を行い、次に Pr/T ネットからμ式への変換を行うことによりＮｅＯからμ算法へのプログラム変換を試みる（図１）。

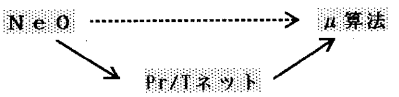

図１　ＮｅＯとμ算法の関係の抽出

２．で対象とする並列処理システムのための計算モデルについて述べ、３．で記述言語ＮｅＯについて述べる。４．でＮｅＯからPr/Tネットへの変換規則を述べ、５．でμ算法およびPr/Tネットからμ算法への変換手順を述べる。そして、６．でＮｅＯとμ算法の関係について論じ、７．を結びとする。

2.　メッセージパッシング型計算モデル

2.1　計算モデル

計算モデルとして、自分だけが参照／更新できる内部メモリと独自のメソッド（処理手続き）をもつオブジェクトが互いにメッセージパッシングを行いながら並列に動作するモデルを考える。メッセージパッシングは、メッセージキュー（以下キューと記す）とよばれるバッファを通じて行われ、オブジェクト同士はキューで結合される（図２）。

各オブジェクトには少なくとも一つのキューが対応しており、オブジェクトがメッセージをキューから取り出すことをメッセージ受理、メッセージをキューへ入れることをメッセージ送信とよぶ。メッセージ送信は、送信先のオブジェクトにとってメッセージの到着に相当する。あるオブジェクトにメッセージが到着すると、メッセージが受理可能かどうか

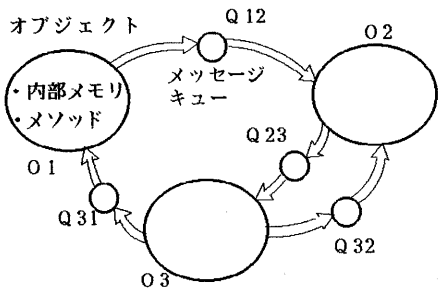

図２　メッセージパッシング型計算モデル

のチェックが行われ、受理条件を満足したときそのメッセージは受理される。メッセージが受理されると、メッセージの内容に応じたメソッドが起動され、

・内部メモリの参照／更新や逐次計算
・メッセージ送信

という動作列の実行が開始される。メッセージの送信後、受理可能なメッセージが到着していればその処理を続けることになる。

2.2　計算モデルの形式的定義

対象とする計算モデルは、各オブジェクトがメッセージパッシングを行いながら並列動作しているものであり、このような構造を次のように定義する。

【定義 1】 メッセージパッシング型計算モデル Mは次の３項組で定義される。

$M=\langle \mathcal{O}, \mathcal{Q}; A \rangle$

$\mathcal{O}$：オブジェクトの集合

$\mathcal{Q}$：メッセージキューの集合

A：$A \subseteq (\mathcal{O}\times\mathcal{Q}) \cup (\mathcal{Q}\times\mathcal{O})$

ただし、$\mathcal{O}\cup\mathcal{Q}\neq\emptyset$、$\mathcal{O}\cap\mathcal{Q}=\emptyset$。Aはオブジェクト間の通信路を要素とする有向枝の集合である。$q_1, q_2 \in \mathcal{Q}$かつ$\mathcal{O}_1 \in \mathcal{O}$で、$q_1\times\mathcal{O}_1 \in A$のとき q_1を$\mathcal{O}_1$の入力キュー、$\mathcal{O}_1\times q_2 \in A$のとき q_2を$\mathcal{O}_1$の出力キューとよぶ。 ■

なお、オブジェクトの構造は 2.3 に示す。

2.3　オブジェクトの振舞い

現在の内部状態（内部メモリの値の組）とメッセージの内容に応じてメッセージが受理されたのち、状態の更新、メッセージの送信が行われる（図３）。

図３　オブジェクトの振舞い

このようなオブジェクトの振舞いは、メッセージを入出力信号とした順序機械としてとらえることができ（図４）、次のように定義することができる。

【定義 2】 オブジェクト$\mathcal{O} \in \mathbb{O}$の動作は、次の５つ組で定義される。

$\mathcal{O} = <X, R, S, \sigma, \omega>$

X：オブジェクトの内部状態の集合

R：入力メッセージの集合

S：出力メッセージの集合

σ：状態遷移関数　$X \times R \rightarrow X$

ω：メッセージ送信関数　$X \rightarrow S$　■

すなわち、オブジェクトは、現在の状態が State $\in X$のときメッセージ Mrec$\in R$を受理したのちは、NewState$\in X$ の状態に遷移し、

σ(State,Mrec) = NewState

その結果に応じメッセージ Msend$\in S$を送信する。

ω(NewState) = Msend

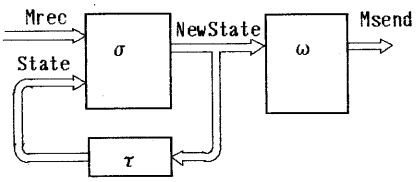

図４　順序機械としてのオブジェクト

2.4 メッセージの管理法

キューにおけるメッセージの管理は次のようにして行われる（図５）。メッセージは到着順に管理されており、先頭のものから順に処理されていく。このため、受理可能なメッセージが複数存在する場合には、先着のものが優先的に処理される。ただし、メッセージの受理はメソッドごとに独立に行われるため、メッセージパターンが異なるもの同士は並列に処理可能であり、オブジェクト内では複数のメソッドが起動される。

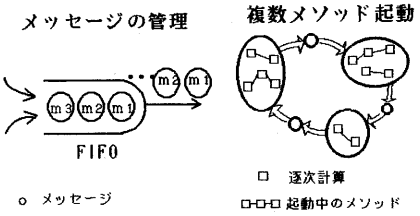

図５　メッセージの管理法

3．計算モデルのネットによる表現

3.1 基本的な考え方

2.2 で与えられたメッセージパッシング型計算モデルをネットとして表現することを考える。ネットの形式的定義は 4.2 に示す。

ネットは、内部メモリ◎、メッセージ受理▮、メッセージ送信▯、キュー○のノードとさらにラベル付きアーク→から構成される。これらとオブジェクト$\mathcal{O}$の構造$<X, R, S, \sigma, \omega>$およびキュー$\mathbb{Q}$、アーク$\mathbb{A}$との対応は次のようになる。

内部状態の集合 X　➡　◎内の変数の値

入出力メッセージの集合　R，S

➡　○内の変数の値

状態遷移関数　σ　➡　▮の集合

メッセージ送信関数ω➡　▯の集合

メッセージキュー$\mathbb{Q}$　➡　○の集合

アーク　$\mathbb{A}$　➡　→の集合

ここで、◎および○内の変数とは計算の状態を表示するために用いるトークン（後述）のことをいう。

図４のオブジェクトをこれらのノードとアークを用いて表現すると図６のようになる。◎→▮と○→▮はメッセージの受理条件、▮→◎ はメッセージを受理したときの動作の実行を表しており、σに対応している。ここで、◎→▮は内部状態（State）のフィードバックループに相当しており、これによりオブジェクトの内部メモリの参照を行っている。また、◎→▯および▯→○は、メッセージの送信条件および送信（Msend）を表しており、ωに対応する。

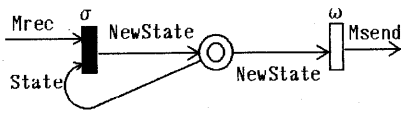

図６　ネットによるオブジェクトの記述

オブジェクト同士がメッセージパッシングを行うようす（図２）を表したのが図７である。

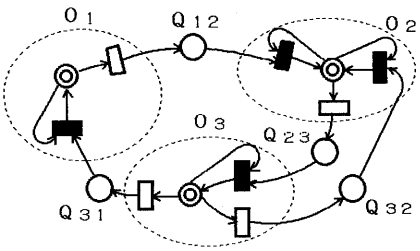

図７　オブジェクトの結合

いくつかのオブジェクトの集まりとしてサブシステムを形成し、サブシステムの集まりとしてシステムを表すことも可能である。

3.2　プログラム図式ＮｅＯ

3.1 でメッセージパッシング型計算モデルＭをネットとして表現する基本的な考え方を示したが、次にネット（ＮｅＯとよぶ）の形式的定義を与える。

提案するＮｅＯは、計算モデルをラベル付き有向グラフとして定義するためのネットである。ＮｅＯでは、内部メモリおよびキューが集合𝕍、メッセージの受理・受信が集合𝔼のノードとして表され、動作列がラベル付きアークとして表される。さらに、ＮｅＯの実行の効果（後述）を表すためにノードへマーキングが与えられる。内部メモリでのマーキングは内部変数の値を、メッセージキューでのマーキングはメッセージを表している。

【定義 3】　ＮｅＯは次の要素からなる（図８）。

(1)　ネット　$N=(\mathbb{V}, \mathbb{E}; A)$

$\mathbb{V}\cup\mathbb{E}\neq\emptyset$、$\mathbb{V}\cap\mathbb{E}=\emptyset$

$A\subseteq(\mathbb{V}\times\mathbb{E})\cup(\mathbb{E}\times\mathbb{V})$

ここで、$\mathbb{V}=\mathbb{Q}\cup\mathbb{C}$。$\mathbb{Q}$はキューを表すＯの集合、$\mathbb{C}$は内部メモリを表す◎の集合で、$\mathbb{Q}\cup\mathbb{C}\neq\emptyset$、$\mathbb{Q}\cap\mathbb{C}=\emptyset$。$\mathbb{E}$はメッセージの受理・受信を表す集合で、$\mathbb{E}=\mathbb{S}\cup\mathbb{R}$、$\mathbb{S}\cup\mathbb{R}\neq\emptyset$、$\mathbb{S}\cap\mathbb{R}=\emptyset$。ただし、$\mathbb{S}$はメッセージ送信を表す□の集合であり、$\mathbb{R}$はメッセージ受理を表す▌の集合である。Ａは有向枝の集合である。

(2)　構造　Σ

記号列（数を含む）の集合・演算の集合・関係の集合からなる構造。変数名、演算名を英小文字で、文字定数を英大文字で表すものとする。

(3)　有向枝に対するラベル付け

Ａの各要素に対し、Σの要素によりラベル付けされる（図９）。

(4)　𝕍へのマーキング

内部メモリへは、内部変数が変数名と値の組のトークンとしてマーキングされる。また、キューへはメッセージがメソッド名と引数の組のトークンとしてマーキングされる。　■

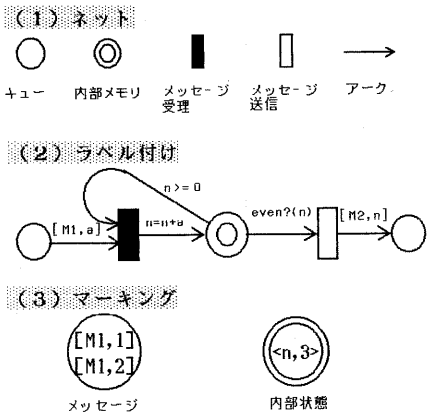

図８　ＮｅＯの構成要素

アークのラベルの種類を図９に示す。Ｏ-▌のアークには、メッセージの受理条件として、メッセージパターン、パラメータの条件、メッセージの個数を記述することができる。◎-□のアークには、メッセージの送信条件として、内部メモリ変数を引数とする述語を記述することができる。内部状態の条件を表す◎-▌のアークにも同様の述語を記述できる。▌-◎のアークには、処理手続きが記述され、代入文、条件文あるいはこれらの逐次処理が記述される。□－Ｏのアークには、送信メッセージパターンを個数とともに記述できる。

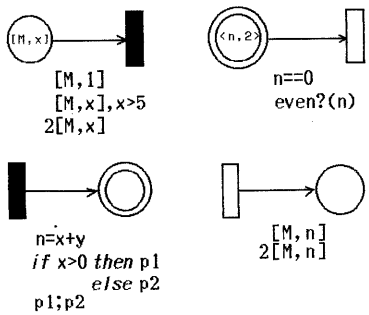

図９　ラベルの種類

ＮｅＯの解釈実行とは、次に示す遷移則に従いマーキングを変更することである。そして、ＮｅＯの実行の効果を、ＮｅＯを実行し停止するまでにどのようにマーキングが変化するかにより表す[4]。

【定義 4】　ＮｅＯの遷移則

ＮｅＯの実行は、次の遷移則に従って行われる。

メッセージ受理：▮のすべての入力アークのラベルを満足するトークンが○（入力キュー）および◎に存在するとき、○のトークン（メッセージ）は受理可能であるという。受理が実行されるとトークンが取り除かれ、▮の出力アークにラベル付けされているメモリの更新が行われる。

メッセージ送信：メッセージ受理後、▯の入力アークのラベルを満足する◎のマーキングが存在するならば、オブジェクトはメッセージ送信可能であるという。送信が実行されると▯の出力アークのラベルに従うトークンがアークの指す○（出力キュー）に置かれる。 ■

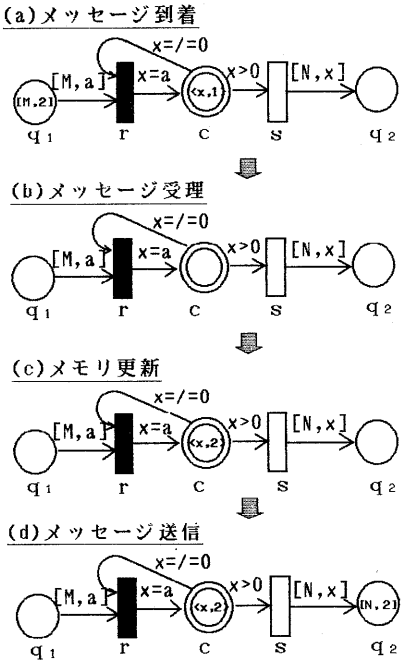

図１０　ＮｅＯの遷移の例

図10に遷移の例を示す。(a)：q_1にメッセージ[M,2] の到着。(b)：メッセージ[M,2] はラベル[M,a]を満足し、さらに、ｃの変数<x,1> がラベルx=/=0を満足するため、メッセージは受理可能。(c)：メッセージ受理後、各ラベルに割り当てられたトークンが取り除かれ、ラベル x=aの実行によりメモリは更新される。(d)：変数 x の値がｓの入力アークのラベルの論理式 x>0 を満足するため送信可能。(e)：送信が行われるとラベル[N,x] に従ったメッセージ[N,2]がq_2に入れられる。

3.3　ＮｅＯの表現能力

ＮｅＯの構造とオブジェクトの動作の対応関係を図11を用いて説明する。

図11-(a),(b)はメッセージ受理に関する動作を表している。まず、(a) においてメッセージパターンが同じ（ $m_1=m_2$）ときは、内部状態に応じた受理メッセージの選択が行われる。メッセージパターンおよび内部状態に対する条件が同時に成立することがない場合には、各々独立に、すなわち並列処理可能であることを表している。メッセージパターンは異なるが、同じ内部状態で受理可能となる場合は、非決定的な選択が行われることを表しており、このとき、競合状態になっている場合もある。(b)の場合

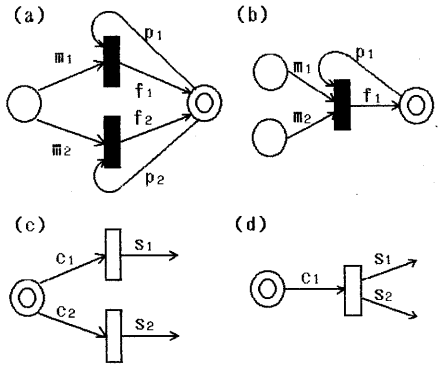

図１１　ＮｅＯの表現能力

は、m_1と m_2 が到着して初めて受理が実行される同期を表している。

メッセージ送信に関しては、(c)のような場合、c_1、c_2ともに満足されれば同時にメッセージが送信されるが、そうでなければ満足されたもののみが送信される。(d)においては、c_1が満足されれば s_1、s_2が同時に送信される。

4．ＮｅＯから Pr/Tネットへの変換

4.1　Pr/Tネット

有向グラフを用いて並列処理システムを記述する

ためのモデルとして、ペトリネットおよびその拡張したモデルである Pr/Tネット[8]が提案されている。Pr/Tネットはペトリネットに比べてその記述能力は強力であり、データベースの並行アクセス制御およびネットワークのプロトル制御などの記述に使用されている[9]。図12に Pr/Tネットの例を示す。

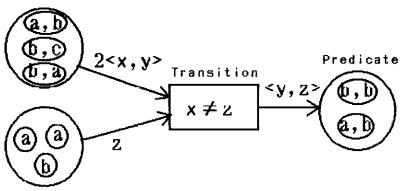

図１２　Pr/Tネットの例

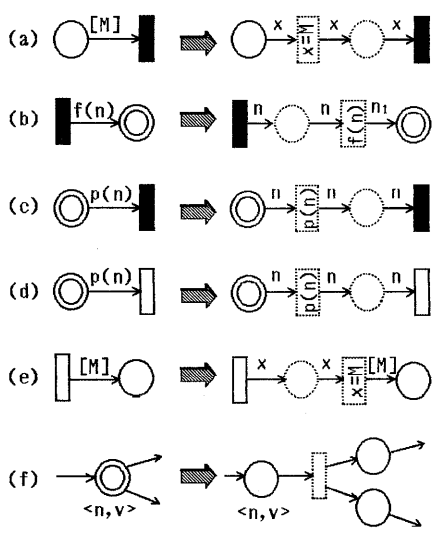

図１３　ＮｅＯから Pr/Tネットへの変換規則

以下では、ＮｅＯとPr/Tネットとの関係を考察し、ＮｅＯから Pr/T ネットへの変換を考える。なお、Pr/Tネットの定義と図式は[8]に従う。

4.2 変換規則

図10のＮｅＯと図12 の Pr/Tネットを比べてみると、ＮｅＯの○は Pr/T ネットの Predicate（以下 Pr）に、▮および ▯は Pr/T ネットの Transition（以下 Tr ）に変換すればよいことがわかる。さらに、ＮｅＯの◎およびアークのラベルに対する変換が必要であり、変換規則を図13 に示す。ここで、[M] はメッセージパターン、ｆは１変数に対する演算でありn_1 はその実行結果 、ｐは１引数の述語、<n,v> は内部メモリ変数とその値の組である。ＮｅＯのラベル上の演算記号（ｆ、ｐなど）はそのままの形で Pr/Tネットにおいても実行可能であるものとする。破線のノードは、ＮｅＯのラベルの処理のために新たに挿入されたものである。図13-(f)に示すように◎を変換するときには、次の規則に従い入出力アークに対応する Prを新たに設ける。

- ◆ 入力アーク（▮→◎）に対応する Prを唯一つ設ける。
- ◆ 出力アークのうち▮と結合しているアーク（◎→▮）に対応する Prを唯一つ設ける。
  ▯と結合しているアーク（◎→▯）については各々に対応した Prを設ける。

なお、メッセージパターンがｍ項組の場合、あるいはｆやｐがｍ変数の場合は、破線のノードの入出力アークのラベルは、ｍ項組となる。

受理メッセージ数をカウントし、偶数番目のときにメッセージ送信を行うＮｅＯプログラムのPr/Tネットへの変換例を図14に示す。ＮｅＯプログラム(a)に図13の変換規則を適用した結果が(b)である（簡単化のためラベルは省略している）。(c)は(b)を等価変換した結果得られたものである。

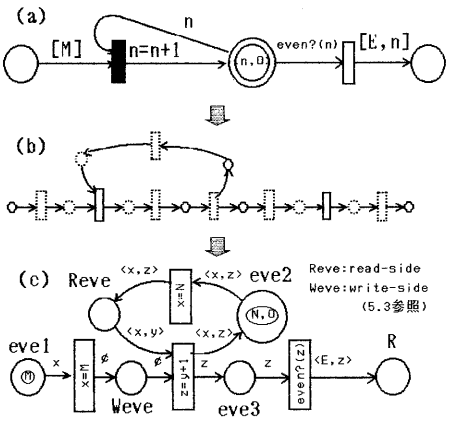

図１４　変換例

5. μ 算法による Pr/Tネットの記述

5.1 μ 算法について

μ算法[7] は、特定のシンタックスに従う文字列を特定の規則に従ってリダクションを行うという体系であり、λ算法の拡張と考えられるものである。λ算法が関数型言語に果たす役割と同様に、μ算法がメッセージパッシングを含む言語に対して理論的

な基礎付けを与える道具の役割を果たすことが期待される[10]。

5.1.1 純μ算法

μ算法に現れる領域は、対象、事象および事象間の”因果関係（→）”である。

【定義 5】 対象とは、

i) あらかじめ決められた定数の集合C（例えば自然数）の要素

ii) 変数の集合Vの要素（英小文字で表記）

iii) 原始演算（例えば加算、比較）の有限集合Pの要素

iv) $(\mu x_1 x_2 \cdots x_n . E_1 E_2 \cdots E_m)$ の形をしたμ式 (x_iは変数、E_jは事象、$n \geqq 1$、$m \geqq 0$)

のいずれかである。 ■

【定義 6】

事象とは対象の列 $(A_1 A_2 \cdots A_n)$ である(n>1)。 ■

事象Eが直接E´を引き起こすならば $E \rightarrow E'$ と表す。→は半順序である。ここで、μ式の $x_1 x_2 \cdots x_n$ を束縛変数、$E_1 E_2 \cdots E_m$を本体と呼ぶ。

【公理 1】 《μリダクション》

事象Eが $((\mu x_1 x_2 \cdots x_n . E_1 E_2 \cdots E_m)\ A_1 A_2 \cdots A_n)$ のとき、すべてのE_jに対して $E \rightarrow E'_j$ が成立する。ただし、E'_jはE_jの中の $x_1 x_2 \cdots x_n$の自由な出現に対して $A_1 A_2 \cdots A_n$ をそれぞれ代入して得られた形 $S[A_1, A_2, \cdots, A_n; x_1, x_2, \cdots, x_n; E_j]$ である。 ■

【公理 2】 《原始演算》

事象Eが (p $c_1 c_2 \cdots c_n$ A) のとき、$E \rightarrow (A\ \tilde{p}[c_1; c_2; \cdots; c_n])$が成立する。ただし、$\tilde{p}$は pが表すn変数の関数である。 ■

事象（対象の列）$A_1 A_2 \cdots A_n$は、$A_2 \cdots A_n$という対象のn-1個の列の、対象 A_1に対する伝送と解釈される。公理１は、μ式$(\mu x_1 x_2 \cdots x_n . E_1 E_2 \cdots E_m)$がメッセージ $A_1 A_2 \cdots A_n$を受け取ると、m個の新しい事象が引き起こされることを表している。また、公理２は原始関数への適用結果$\tilde{p}[c_1; c_2; \cdots; c_n]$が A（継続）に送られることを表している。

5.1.2 μ算法

純μ算法では、一つの事象から複数の事象が並列的に引き起こされることを許すが、逆にいくつかの独立した事象によって一つの事象が引き起こされるような現象を許しておらず、並列に走るプロセス間の通信、同期という形が実現されていなかった。そのため、純μ算法に ”conduit”という概念を導入して、μ算法が構成されている。

【定義 7】

conduitとは、次のような集合の要素をいう。

$K = \{ < r_x, w_x > \mid X$はμ算法の対象$\}$ ■

r_x、w_xは新しいクラスの対象であり、それぞれread-side、write-sideと呼ぶ。次の公理はconduitを特徴づけるものである。ここで導入される⇒は、→が直接的な因果関係であるのに対し、遠隔的な因果関係といえるものである。

【公理 3】 《conduit》

(1) $E \rightarrow E'$ ならば $E \Rightarrow E'$

(2) $E \Rightarrow E'$ および $E' \rightarrow E$ ならば $E \Rightarrow E$

(3) $E \Rightarrow (r_x A)$ および $E \Rightarrow (w_x B)$ ただし、$< r_x, w_x > \in K$ならば、$E \Rightarrow (A B)$

(4) $E \Rightarrow E'$ は(1)～(3)の規則の有限回の適用だけから得られるものである。 ■

【公理 4】 《conduitの創生》

創生作用素ξは、次のような性質をもつ。

$\xi C \rightarrow C\, r_c\, w_c$ ただし、Cは対象。 ■

Cは作用素ξの働きによって新しく作り出されるconduit を受け取る継続の役割をもつと同時に、その読み側と書き側の対応関係を指定する名前の役目も果たす。

公理３を拡張して書き側のｎ要素を読み側に送れるようにしたのが次の公理である。

【公理 5】 《公理３の拡張》

$r_x A$と$w_x B_1 B_2 \cdots B_n$の対は、事象

$A B_1 B_2 \cdots B_n$ を引き起こす。

ただし、Xは対象である。 ■

5.2 μ算法インタプリタによる実行例

現在のインタプリタは、[10]の簡約系をCに移植したものである。簡約系の概要およびμ算法の並行システム記述への応用例などは、[10]を参照されたい。なお、構文要素はすべてＳ式として記述される。

【Ｓ式による表現例】

```
• > 4 3 c  ---> ( > 4 3 c )
• μxc.(+3xc) --->  (mu (x c) (+ 3 x c))
• (μxc.(+x7c)(-x7c)) 5 R  --->
    ((mu (x c) ((+ x 7 c)(- x 7 c))) 5 R)
```

インタプリタに原始演算として用意されている主

なものをあげる。リストのcar部が演算名、cdr部が引数に相当する。numは自然数を contは継続を表す。

(succ num cont)　：算術演算で後者関数。
(+ num num cont)　：-,*,/ も同様な算術演算。
(even? num cont)　：述語演算。
(eq? var var cont)：述語演算。
(> num num cont)　：>=,<,<=,=も同じ述語演算。

述語演算の場合、真なら μabc.caを、偽なら μabc.cbを継続へ送る。

μ式に名前nameをつけるときは次の演算を用いる。

(defMu name bind events)

bindはμ式の束縛変数、eventsは本体。

conduitを生成するときには、

(create name)　name は対象名

を用いると Rname および Wname が作られる。

インタプリタによるリダクション例を示す。

【リダクション例】　Rを継続とする。

```
Mu >> (succ 8 R)
 --> (R 9)
Mu >> ((mu (c) ((+ 3 3 c) (c 4))) R)
 --> (R 4)
 --> (+ 3 3 R)
Mu >> (> 5 2 (mu (x) (x 1 0 R)))
 --> ((mu (x) (x 1 0 R)) (mu (_a _b _c) (_c _a
)))
 --> ((mu (_a _b _c) (_c _a)) 1 0 R)
 --> (R 1)
Mu >> (defMu add1 (a c) (succ a c))
add1
Mu >> (add1 2 R)
 --> (succ 2 R)
 --> (R 3)
Mu >> ((mu (x c) ()) 7 R)
nil
```

最後の例のように本体がないμ式(m=0)の計算は停止する。本体のないμ式を以下では halt とする。

5.3 Pr/Tネットの表現

Pr/Tネットをμ算法の対象を用いて表現するための変換手順を以下に示す。変換手順の概要は、Pr/Tネットを図15に示す基本的な要素ネットの組合せと考え、要素ネットをμ式で表し、 Pr/T ネットの構成を要素ネットを組み合わせた式で表すものである。

【 変換手順 】

(1) 各 Pr に一意に定まる名前をつける。ただし、Trが２つの入力Prをもつ場合、一方を read-side、他方をwrite-sideとし、それぞれに名前をつける。

(2) 次に示す変換規則に従いながら Trごとにμ式 ($\mu x_1 x_2 \cdots x_n . E_1 E_2 \cdots E_m$)を作成する。入力アークのラベルがｎ項組なら束縛変数を $x_1 x_2 \cdots x_n$とする。そして、求められたμ式に (defMu を用いて) 入力Pr名を名前としてつける。なお、簡単化のため以下の説明および図15では n=1としている。以下に示すμ式の束縛変数名は任意に選んでよい。

(2-1)　共通の入力Prをもたない Trに対して：

① １入力で出力 Prが read-side および write-sideではない場合

図15-(a)：

◆ｆが算術演算で、f(a)=c のとき、

$\mu a.(f\ a\ \mu x.(D_1\ x)\cdots(D_k\ x))$

なお、図15-(b)のように k=1 のときは、

$\mu a.(f\ a\ D)$　でもよい。

◆ｆが述語演算のときは、Trが発火可能であるときのみ演算結果を送る。すなわち、述語が真ならば D_j へ a を送る。

```
μa.(f a μw.(w μx.(D1 x)···(Dk x))
              μy.(halt y)
              μz.(z a)))
```

② １入力 Prで、出力 Pr が write-side の場合

図15-(c)：①の継続D_jを write-side名とする。

③ １入力 Prで、出力 Pr が read-side の場合

図15-(d)：①の継続（波線部）を

$(D\ \mu h.(C\ x\ h))$　とする。

ただし、D は read-side名、C は次に示すように、図15-(e)から求められるμ式、あるいはそのμ式に付けられた名前である。

◆ｇが算術演算ならば、

$\mu ab.(g\ a\ b\ \mu v.(D_1\ v)\cdots(D_k\ v))$

◆ｇが述語ならば、

```
μab.(g a b μw.(w μx.(D1 x)···(Dk x))
                 μy.(halt y)
                 μz.(z a)))
```

ここで、ｇは２変数の演算とする。

(2-2) 共通の入力 Pr をもつ Tr について

ＮｅＯを変換して得られる Pr/T ネットに関しては、f_i がすべて述語演算の場合であるため、f_iを順番に実行しながら最初に真となった Tr の継続へ値を送るとよい。

図15-(f)： i=2 の場合は次のようになる。

```
μa.(f1 a
    μw.(w μx.(D x)
          μy.(f2 a μu.(u μv.(E v)
                       μv.(halt v)
                       μw.(w a))
          μz.(z a)))
```

なお、すべて算術演算である場合、あるいは述語と算術演算が組み合わさせている場合は、非決定的にTrが選ばれ演算が行われるように実現する方法が考えられる。[11]ではそのための

公理が導入されている。

(3) Trの入力Pr上にトークンがある場合には、Trに相当するμ式とトークンに相当する対象との組による事象を作る。

図15-(g)： (μa.(+ a 2 D)) 1

図15 に現れないようなネットの場合は、補助 Prを挿入しながら、要素ネットの組み合わせとなるように変形していけばよい[11]。

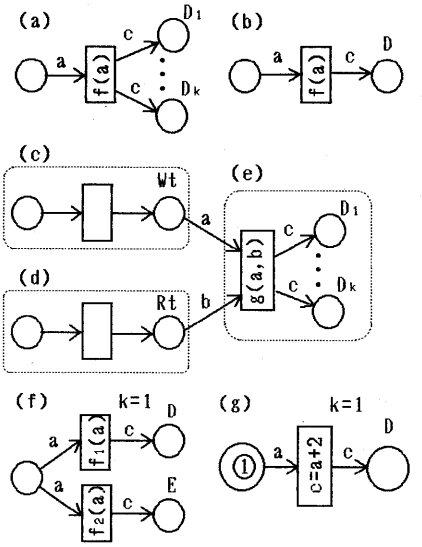

図１５　要素ネット

図14-(c)をμ式を用いて表現した例を示す。

```
(defMu eve1 (x)          ; 手順(2-1)②
   (eq? x M (mu (w) (w (mu (a) (Weve a))
                       (mu (b) (halt b))
                       (mu (c) (c x))))))

(defMu eve0 (n num)     ; ReveとWeveが結合された
   (+ num 1 (mu (new)   ; とき実行されるμ式
                ((eve2 n new) (eve3 new)))))

(defMu eve2 (x num)       ; 手順(2-2)③
    (eq? x N (mu (w)
          (w (mu (a1 a2)
                (Reve (mu (h) (eve0 a1 a2))))
             (mu (b) (halt b))
             (mu (c) (c x num))))))

(defMu eve3 (num)          ; 手順(2-2)①
    (even? num (mu (w) (w (mu (a) (R E a))
                          (mu (b) (halt b))
                          (mu (c) (c num))))))
```

プログラム－１

5.4　Pr/Tネットのトークンゲーム

Pr/T ネットの初期マーキングからのトークンゲームを、μ算法での事象のリダクション過程としてとらえるとき、次に示す問題点が生じる。

(i)　公理３の変更

公理３を実現するには、conduit に送られてきた対象すべてを記憶しておく必要がある。これでは、read-sideとwrite-sideの Prに送られたトークンはTr発火後いつまでも残っていることになる。そこで発火後は、トークンが Pr から取り除かれるようにするため、公理３の (4)を次のように変更する。

【公理３′】　《conduit》

(4)　E⇒E′ は(1)、(2)の規則の有限回の適用と(3)だけから得られるものである。　■

(ii)　トークンの保存

通常の Prの場合、(i)とは対象的に、発火不可能なのにトークンが取り除かれることが生じる。それは、Trが述語演算の場合、発火可能であるかかどうかは述語演算のリダクションを行いはじめてわかるため、真偽にかかわらずトークンは取り除かれてしまう。このため、発火不可能なときにトークンを残しておくためには、再びPrへトークンを送るための処理が必要である。ただし、このような状況がトークンゲームに何ら影響を与えない場合は、発火不可能なトークンが取り除かれたままでもさしつかえない（プログラム－１がこれにあたる）。

プログラム－１の実行は次のように行われる。まず、create を用いて conduit を生成する。そして、eve1と eve2の Prに初期トークンをおくことによりリダクションが開始される。

```
Mu >> (create eve)
eve
Mu >> ((mu (a b c) ((eve1 a) (eve2 b c))
Mu >>       M N 0 )
 --> (eve2 N 0)
 --> (eve1 M)
 --> (eq? N N (mu (w) (w (mu (a1 a2) (Reve (mu
 (h) (eve0 a1 a2)))) (mu (b) (halt b)) (mu (c)
 (c N 0)))))
 --> (eq? M M (mu (w) (w (mu (a) (Weve a)) (mu
 (b) (halt b)) (mu (c) (c M)))))
      :
      :
```

6.　μ 算法によるＮｅＯの表現

6.1　ＮｅＯ、 Pr/Tネット、μ算法の関係

２段階の変換過程を通じてμ算法を用いてＮｅＯを表現することが可能となった。

ＮｅＯから Pr/Tネットへの変換過程においては、ＮｅＯのラベルの処理に相当する Tr を新たに挿入することにより変換が行われる。これにより、Ｎｅ

Oの構造は図15のような要素ネットの組合せとして表現されることになる。要素ネット（例えば図 15-(b)）の Prをキュー、Trをオブジェクトとみなした場合、ＮｅＯから Pr/T ネットへの変換は、ＮｅＯで定義されたオブジェクトのより基本的なオブジェクトへの分解を行っていることになる。また、変換後は、メッセージを表すトークンと内部メモリ変数を表すトークンの区別がなくなり、単にｎ項組のトークンとして表され、このトークンがメッセージの役目をはたす。

さらに、Pr/Tネットからμ算法への変換過程においては、Tr ごとに出力 Prを継続とするμ式を定義していくことにより変換が行われる。このとき、入力 Pr に置かれたトークンは対象として表現され、Tr に相当する μ式と組み合わされてリダクションが行われる。その結果は継続である出力 Pr へ送られる。 Pr/Tネットから μ算法へ変換することにより、Pr/Tネットのトークンゲームはμ算法のリダクションとして実現される。

以上の考察から次のことがいえる。これまではＮｅＯを実行するとき、アークを並列処理の最小単位としていたが[5]、ＮｅＯを μ算法の対象によって表現した場合、アークをさらにいくつかに分割した単位で並列処理が行えるようになる。すなわち、並列処理の粒度をより細かくすることができる。このことは、オブジェクト同士でしか行われていなかったメッセージパッシングがオブジェクト内のメソッド処理においても行われることを表している。

6.2 プログラムの意味

μ算法において、計算の過程（実行の効果）は事象間の因果関係→としてとらえられる。ＮｅＯプログラムを μ 算法で表現することにより、ＮｅＯの計算過程（実行の効果）を因果関係→としてとらえることが可能となる。μ算法での事象のリダクションが、ＮｅＯにおけるラベルの実行に相当しており、事象間の因果関係は、ラベルの実行系列に対応している。図16に示すようにＮｅＯでは、この因果関係をノードの結合関係として直接的に表しており、ＮｅＯプログラムの意味をアークの実行系列としてとらえることができる。

7. おわりに

ＮｅＯから Pr/Tネットへの変換、さらには Pr/Tネットからμ算法への変換を通じてＮｅＯのプログラムをμ算法のプログラムに変換する規則が与えられ、μ算法を用いたＮｅＯの形式的意味記述が可能となった。これで、ＮｅＯのプログラムの意味をマーキングの変化系列として与える方法とラベルの実行系列として与える方法の２つの考え方が得られたわけであるが両者の関係を明らかにすることが今後の課題である。また、ＮｅＯをアーク単位で並列処理していたがより細かな処理単位で並列処理を行う処理系の開発も課題である。

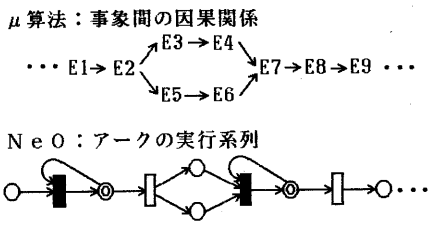

図１６　事象の因果関係とアークの実行系列

参考文献

[１] 電子総合研究所：新形態プログラミング：現状と展望、電子技術総合研究所調査報告第210号 (1984).

[２] 二木厚吉：実行可能仕様に基づく変換プログラミング、情報処理、Vol.28,No.7,pp.906-912 (1987).

[３] Hewitt,C.：Viewing Control Structures as Patterns of Passing Messages, Artificial Intelligence, Vol.8, pp.323-364 (1977).

[４] 猪股、西村：プログラム図式ＮｅＯによるオブジェクト指向型言語プログラム変換、第36回情報処理学会全国大会、2H-4 (1988).

[５] 片野田、猪股、西村：オブジェクト指向に基づく並行システムのためのモデリング・シミュレーション支援システム：Ｃｏｓｍｏｓ、第35回情報処理学会全国大会、1R-4 (1988).

[６] 片野田、田中、猪股、西村：並行システム・シミュレーション支援システムＣｏｓｍｏｓ－シミュレーション過程表示－、第36回情報処理学会全国大会、2K-4 (1988).

[７] Ward,S.A.,Halstead,Jr.R.H. : A Syntactic Theory of Message Passing, J. of ACM, Vol.27, No.2, pp.365-383 (1980).

[８] Genrich,H.J.,Lautenbach,K.: System Modelling with High Level Petri Nets, Theoretical Computer Science, Vol.13 , pp.109-136 (1981).

[９] 斉藤：ＯＳの理論、コンピュータから生まれた新しい数学、日本評論社、pp.127-140 (1986).

[１０] 猪股、橋爪、西村：μ式簡約系の試作と並行システム記述への応用、ソフトウエア基礎論研究会、87-SF-21 (1987).

[１１] 橋爪、神保、猪股、西村：μ式によるペトリネットの表現、第34回情報処理学会全国大会、6U-2 (1987).