ICOT Technical Report: TR-500

TR-500

ペトリネットに基づく
並行プログラミング言語
-時相命題論理による検証—

内平　直志、
本位田　真一（東芝システムソフトウェア研究所）

September, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

ペトリネットに基づく並行プログラミング言語

Petri Net-based Concurrent Programming Language on Distributed Processing Systems

--- 時相命題論理による検証 ---

Verification using Temporal Logic

内平直志　　本位田真一

Naoshi Uchihira　　Shinichi Honiden

株式会社　東芝　　システムソフトウェア技術研究所

TOSHIBA Corporation　Systems & Software Engineering Lab.

概要

ペトリネットに基づき，モジュール化機構とＣＣＳ的同期通信機構を導入した並行プログラミング言語ＭＥＮＤＥＬ／８９を提案する．また，時相命題論理によるＭＥＮＤＥＬプログラムの検証手法を示す．

目次

１．はじめに

原子力発電所の制御システム，あるいは複数の協調するロボットの制御システムなど分散協調システムは世の中に多数存在する．このような分散協調システムのソフトウェアの開発は一般に難しい．その理由は，いくつかのプロセスが並行に動く場合，起こりうる状況は組合せ的に多くなり，設計者は全ての状況を把握しきれなくなるからである．そこで，高レベル並行プログラミング言語とそのプログラミング支援環境は必要不可欠であり，多くの研究開発が成されてきた．本論文では，分散協調システムの概念設計レベルの支援を目的としたプロトタイピング言語とその検証手法について述べる．

高レベル並列プログラミング言語として，世の中にはAda，CSP，Occam，GHC等が既にあるが，本論文では概念設計レベルの支援を目的として，視覚的理解性に優れているペトリネットをベースとした並行プログラミング言語MENDEL/89（以下MENDELと呼ぶ）を提案する．MENDELの特徴は，ペトリネット

にモジュール化機構とＣＣＳ[1]風のハンドシェイク型同期通信機構を導入し，またそれを視覚的に表現したMENDELネットを提供した点である．

並行プログラミング支援環境としては，様々な実行の可視化ツール，デバッガ，検証系などが報告されている．本論文では時相論理を用いた並行プログラムの正当性検証手法について述べる．ペトリネットの解析としては，安全性，有界性，保存性，デッドロック可能性などの判定が可能であった[2]．ここでは，さらにトランジションの発火の順序関係に関する制約条件を時相命題論理で与え，それが満たされているかどうかを検証する手法を提案する．ペトリネット（有限オートマトン）と時相命題論理を融合した検証手法としては表１に示す研究成果があるが，ここで提案する手法の位置づけは斜線部である．なお，Suzuki&Lu[3]は，ペトリネットと時相命題論理を融合したTemporal Petri Netを提案している．これは我々の手法より両者の融合が密なためTuring Machineと同等の記述力を持つが，逆に検証能力は弱くなってしまう．

	ＰＴＬ 線形時間時相論理	ＣＴＬ 分枝時間時相論理
オートマトン	WolperのETL[4]	Clarke 他[5]
安全なペトリネット	片井 他[6]	Tuominen[7]
一般のペトリネット	本手法 （制限あり） (Suzuki&Lu[3])	Tuominen[7] （制限あり）

表１　時相論理とペトリネットによる検証

現在，MENDELの実行系はPrologマシンPSIおよび複数のパソコンをLANで接続した知的分散システム[8,9]上に実装されている．

２．並行プログラミング言語

MENDELはペトリネットをベースとする高レベル並行プログラミング言語である．MENDELプログラムの骨格はペトリネットでモデル化される．ここで，MENDELの骨格を素直にモデル化できるようにペトリネットを特殊化したMENDELネットを導入する．MENDELネットは等価なペトリネットに容易に変換できる．本章では，最初に視覚的なMENDELネット[10]を説明し，次に言語仕様を簡単に示す．

2.1 ＭＥＮＤＥＬネット

MENDELネットは以下の特徴を持つペトリネットである：

●モジュール化機構がある．MENDELの各オブジェクト（プロセス）がペトリネットのモジュールとなる．

●オブジェクト間には２種類の同期通信機構(データフロー型，ハンドシェイク型)がある．データフロー型はプレースの共有，ハンドシェイク型はトランジションの共有によってモデル化される．

【定義　ＭＥＮＤＥＬネット】

MENDELネットは以下の節点と枝から構成される階層的グラフである：

まず，グラフのモジュールとしてオブジェクトを導入する．

OBJ:オブジェクトの有限集合

SUBOBJ(obj)⊂OBJ:オブジェクトobjのサブオブジェクトの集合

ここでは，オブジェクト間の親子関係が木構造を形成するもののみを対象とする（図１）．

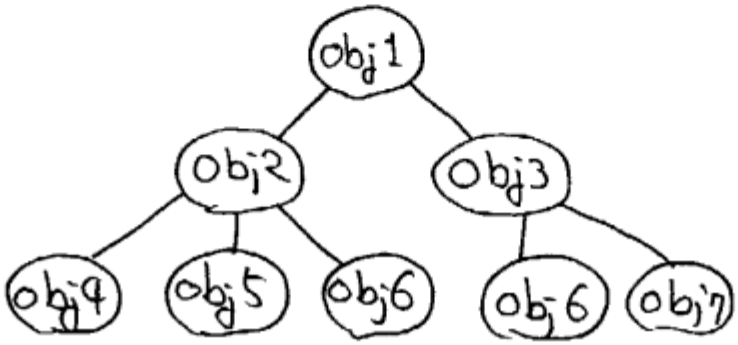

図１　オブジェクトの階層関係

次に，各オブジェクトobj∈OBJに属するプレースとトランジションを定義する．プレースとトランジションがMENDELネットの節点になる．

IP(obj): objの内部プレースの集合

EP(obj): objの外部プレースの集合

IT(obj): objの内部トランジションの集合

ET(obj): objの外部トランジションの集合

外部プレースおよび外部トランジションは親オブジェクトに共有される．それ以外のオブジェクト間の共有はない．

$P(obj) \equiv IP(obj) \cup EP(obj) \cup \bigcup_{obj' \ SUBOBJ(obj)} EP(obj')$

$T(obj) \equiv IT(obj) \cup ET(obj) \cup \bigcup_{obj' \ SUBOBJ(obj)} ET(obj')$

$P \equiv \bigcup_{obj \ OBJ} P(obj)$

$T \equiv \bigcup_{obj \ OBJ} T(obj)$

次に，プレースとトランジション間，あるいはトランジションとトランジション間の２種類の枝（アーク，リンク）を定義する．

$A(obj) \subseteq (P(obj) \times T(obj)) \cup (T(obj) \times P(obj))$:プレースとトランジション間のアーク（有向辺）の集合

pがtの入力プレース　iff (p,t)∈A

pがtの出力プレース　iff (t,p)∈A

L(obj)⊆(T(obj)×T(obj)):トランジション間のリンク（無向辺）の集合

このトランジション間のリンクは，トランジションの同一化を意味する．すなわち，同一化リンクを介してトランジションの同値類が形成される．

各プレースは0個以上のトークンを持つ．全プレースのトークンの分布をマーキングMと呼ぶ．つまり，マーキングMはプレースからトークン数への関数（M：P→正の整数の集合）である．

次に，システムの状態遷移を定義する．t∈Tの各入力プレースpに対して，M(p)≧（pからtへのアーク数）のとき，tは自己発火可能であるとする．さらに，tの同値類が全て自己発火可能なとき，tは発火可能であるとする．発火可能なトランジションtの発火とトークンの移動は次の規則に従う：

(R1) tが発火するとマーキングがMからM'に遷移する：

M'(p)←M(p)－（pからtへのアーク数）
（pがtの入力プレース）

M'(p)←M(p)＋（tからpへのアーク数）
（pがtの出力プレース）

M'(p)←M(p)
（それ以外）

(R2) tの同値類に属する全てのトランジションは同時に発火する．属さないトランジションは同時には発火しない．

【MENDELネットの図的表現】

MENDELネットは次の図的表現で視覚的に表現する．

○　内部プレース

□　外部プレース

▌　トランジション

→　アーク

……　リンク

●　トークン

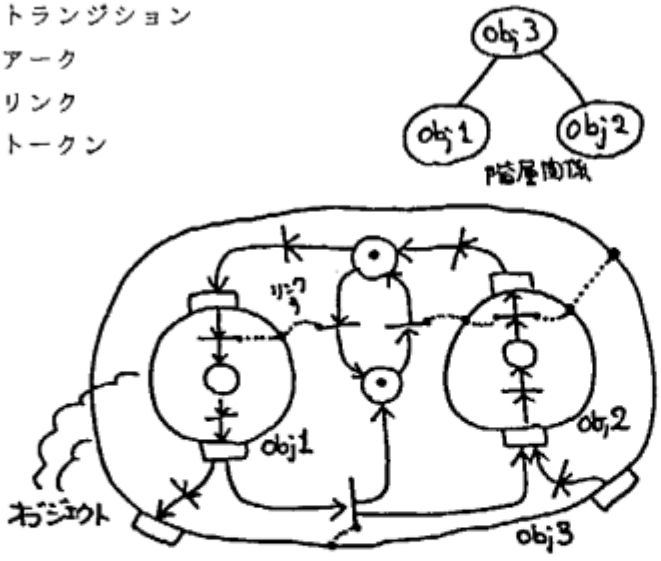

図2　ＭＥＮＤＥＬネットの例

2.2 ＭＥＮＤＥＬの言語仕様

MENDELはMENDELネットを基に，さらに細かい記述ができるようにした並行プログラミング言語である．細かい記述とは：

- ●トークンとして任意の型のデータが操作できる．
- ●トランジションの発火に伴ってＣ言語の関数が起動できる．

以下にMENDELの言語仕様の概要を述べる．MENDELのプログラム単位をオブジェクトと呼ぶ．

```
<オブジェクト>::=
object <オブジェクト名>
        (<外部メソッド>,...)
        (<入力ポート>,...)
        (<出力ポート>,...)  :{
    type:{  <トークンデータのタイプ宣言>  } ;
    dec:{
      state(<状態要素集合>,<初期状態要素集合>) ;
      slot(<初期スロット>,...) ;
    } ;
    body:{  <サブオブジェクト> ; ...     } ;
    meth:{  <メソッド定義>; ...          } ;
    junk:{  <Cの関数定義>; ...           }
  }.

<サブオブジェクト>::=
        <オブジェクト名>
        (<メソッド名>,..)
        (<出力ポート名>,..)
        (<入力ポート名,..>)

<メソッド定義>::=
  method(<メソッド名>, <交換項>,
        [<入力状態>,...],
        [<出力状態>,...],
        [<条件スロット>,...],
        [<作用スロット>,...],
        [<入力ポート>,...],
        [<出力ポート>,...]) <-
        <ガード用Cの関数呼び出し列> |
                        <Cの関数呼び出し列> ;
```

オブジェクトはサブオブジェクトをbody部で再帰的に呼ぶことにより図１の木状の階層構造を形成す

る．また，ポートがMENDELネットの外部プレースに，個々の状態要素が内部プレースにそれぞれ対応する．メソッドの同一化リンクについては例で示す．図３のプログラムではオブジェクトparentのメソッドm1,m2はそれぞれオブジェクトchild1,child2のメソッドgと同一化リンクで結ばれている．

```
object parent ()(in)(out):{
          type:{   int m1,m2,data ;  };
          dec:{    state([s1,s2],[s1]); slot(data,0); };
          body:{   child1(m1)()();
                   child2(m2)()();       };
          meth:{
                    method(m1,_,[s1],[s2],[],[data(N)],[in(N)],[])
                        <- true() | puts("method m1");
                    method(m2,_,[s2],[s1],[data(N)],[],[],[out(N)])
                        <- true() | puts("method m2");
          };
          junk:{};
}.
object child1(g)()() :{.....}.
object child2(g)()() :{.....}.
```

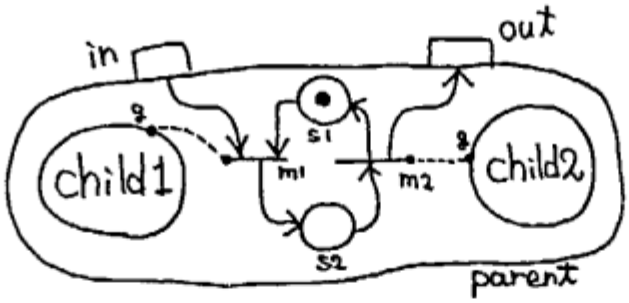

図３　MENDELプログラムとMENDELネットの例

ここで，MENDELオブジェクトの動きは全てメソッドが担っているのでメソッドの挙動を説明する．

(1)メソッドの扱うデータ

メソッドはペトリネットのトランジションに条件部と作用部を付加したものである．条件部，作用部にはそれぞれ状態，スロット,ポートが記述される．

状態要素：状態要素は有限であり，dec部で宣言される．オブジェクトの状態は状態要素の部分集合で表される．ある状態要素s1にトークンがある（トークンがない）ことをs1が「オン」（「オフ」）であると呼ぶ．メソッドが発火されると条件部の状態はオフになり，作用部の状態はオンになる．

スロット：スロットは内部データの格納場所である．メソッドが発火すると条件部のスロットの値

は参照され，作用部のスロットの値は新しい値に上書きされる．

ポート：入出力ポートは外部との通信用のFIFOのバッファである．メソッドが発火したとき，条件部のポートから先頭の値（ペトリネットのトークン）がポップされ，作用部のポートの最後尾に値がプッシュされる．

交換項：ハンドシェイク型同期通信時に，同一化されているメソッドの交換項間で単一化を行う．ＣＣＳ（ＣＳＰ）的なデータの受渡しができる．

(2) メソッドの発火

以下の５つの条件が成り立てばメソッドｍは発火可能である：

(c1) メソッドｍと同一化されている親およびサブオブジェクトのメソッドが同時に自己発火可能であり，かつメソッドｍの交換項が単一化可能である．

(c2) メソッドｍの全ての条件部の状態（入力状態）がオンである．

(c3) メソッドｍの全ての条件部のスロット（条件スロット）が単一化可能である．

(c4) メソッドｍの全ての条件部のポート（入力ポート）に値（トークン）が存在し，その先頭の値が単一化可能である．

(c5) ガード部の全てのＣ言語の関数のリターン値がTRUEである．

メソッドｍが発火場合，次の操作を行う：

(f1) 同一化されているサブオブジェクトのメソッドを発火する．また，同時にメソッドｍの交換項を単一化する．

(f2) メソッドｍの全ての条件部の状態（入力状態）をオフにし、全ての作用部の状態（出力状態）をオンにする．

(f3) メソッドｍの全ての作用部のスロット（作用スロット）を変更する．

(f4) メソッドｍの全ての条件部のポート（入力ポート）から先頭の値を１つポップし、全ての作用部のポートに値をプッシュする．

(f5) ガード部より後ろのＣ言語の関数を逐次実行する．

３．検証

3.1 時相命題論理

時相命題論理(PTL:Propositional Temporal Logic)は命題論理に「時間」の概念を導入した論理である．すなわち時間によって原子命題の真偽値が変化する世界を記述できる．ここで用いるPTLの厳密な定義は[11]に譲るとして，シンタックスと直感的意味を簡単に説明しておく．

◎シンタックス

構成要素

＊原子命題の集合：P={p1,p2,...}

＊論理記号：∧，∨，¬，⊃

＊時相オペレータ：□，◇，○，U

構成規則

＊原子命題p∈Pは論理式

＊f1，f2が論理式ならば，f1∧f2，f1∨f2，¬f1，f1⊃f2，□f1，◇f1，○f1，f1Uf2も論理式

◎直感的意味

∧(論理積)，∨(論理和)，¬(否定)，⊃(含意)

□f ≡　　常にfが真

◇f ≡　　いつかはfが真

○f ≡　　次の状態でfが真

(f1Uf2)≡　f2が真になるまでf1は真，あるいはf1が常に真

3.2　ペトリネット

MEMDELネットではなく，ピュアなペトリネットN＝（P，T，A，M0）を対象に議論する．ここで，Pはプレースの集合，Tはトランジションの集合，Aはアークの集合，M0は初期マーキングである．また，あるマーキング$m=(m_1,m_2,...)$に対して発火可能な$t\in T$を発火させたあとの新しいマーキング$m'=(m'_1,m'_2,...)$を求める関数をμとする（$m'=\mu(m),m'_i=\mu_i(m)$）．

3.3 ペトリネットとＰＴＬの対応

ペトリネットとＰＴＬ式の対応づけには２つのアプローチがある：

(1) 原子命題＝プレースのトークンの有無

ある時点でペトリネットのプレースpにトークンが存在するとき原子命題pは真であるとし，そうでなければ偽とする．

(2) 原子命題＝トランジションの発火

ある時点でペトリネットのトランジションtが発火すれば原子命題tは真であるとし，そうでなければ偽とする．ペトリネットで同時に発火できるトランジションは１つであるから，各時点で真である原子命題はただ１つである(single event condition)．発火するトランジションがない場合は，原子命題nop(no operationを意味する)が真であるとする．

ここでは，(2)の対応づけを採用する．

3.4 ＰＴＬによるペトリネットの検証

ペトリネットで与えられる並行システムが，ＰＴＬ式で与えられる制約を満たすかどうかを検証したい．

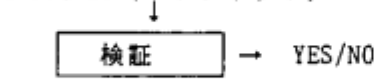

図４　ペトリネットの検証

ここでは，検証をω言語として定式化する．ただし，ペトリネットがデッドロックに陥る場合はω語にならないので，ダミーのトランジションnopを追加する（図５），nopの導入による不利益は3.6で述

べる．

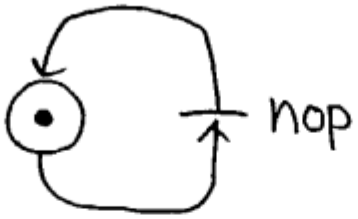

図５　ダミーのトランジション

Ｎ：ペトリネット（含むnop）

Σ：Ｎのトランジションの集合（含むnop）

ｆ：ＰＴＬ式

L(N)⊂Σ　：Ｎから生成されるトランジションの全発火系列の集合

L(f)⊂Σ　：single event conditionのもとでのｆの全モデルの集合

【定義】

Ｌ（Ｎ）⊂Ｌ（ｆ）のとき，「ペトリネットＮがＰＴＬ式ｆを満たす」という．

3.5 検証アルゴリズム

【補題１】

任意のＰＴＬ式ｆに対して，（sigle event conditionのもとで）次の式が成り立つ：

Ｌ（￢ｆ）＝Σ －Ｌ（ｆ）

証明：省略．

【補題２】

任意のＰＴＬ式ｆに対して，（sigle event condi-
tionのもとで）Ｌ（ｆ）を受理するB̈uchi ωオートマトンＡ$_f$が構成できる．

証明：single event conditionを仮定しない一般的な定理[11]の特殊ケースである．

【定理】

任意のペトリネットＮとＰＴＬ式ｆに対して，Ｌ（Ｎ）⊂Ｌ（ｆ）の真偽は決定可能である．

証明：補題１より，Ｌ（Ｎ）∩Ｌ（￢ｆ）＝φの真偽が決定可能であればよい．そのアルゴリズムを示す．

まず，Ｌ（￢ｆ）を受理するB̈uchi ωオートマトンＡ$_f$＝（Σ，Ｓ，δ，s0，Ｆ）を作る（補題２）．ここで，Σはトランジションの集合，Ｓは状態集合，δ：Ｓ×Σ→２^{S}は非決定性遷移関数，s0∈Ｓは初期状態，Ｆ⊂Ｓは目的状態の集合．また，ω語x1,x2,x3,...上のＡ$_f$のランとは，$s_i \in \delta(s_{i-1}, xi)$を満たす無限状態列s0,s1,s2,...である．定義から，任意のω語α∈Ｌ（￢ｆ）上のランに対して，Ｆのある状態を無限回通過するものが存在する．

次にＮの到達可能木を拡張した拡張到達可能木Ｔを作る．Ｎのプレース数をkとしたとき，k+2次元ベクトル$x=(x_1,\ldots,x_k,x_{k+1},x_{k+2})$によるＴのノードを考える．ここで，$(x_1,\ldots,x_k)$はＮのマーキング，$x_{k+1}$はＳの状態番号（便宜的に$s(x_{k+1})$で番号$x_{k+1}$の状態要素を表す），$x_{k+2}$はＦのいずれかの状態の通過回数を表す．また，ベクトルxがt∈Σによって遷移可能であるとは，マーキング$(x_1,\ldots,x_k)$においてtが発火可能，かつ$\delta(s(x_{k+1}),t)\neq\phi$であると定義する．Ｔは以下の手順で作る．

(step1) ルートノード$x^0=(x_1,\ldots,x_k,j,0)$を設定する．ここで，x_i＝プレースiのトークン数($1\leqq i\leqq k$)，jはs(j)＝s0を満たす状態番号．

(step2) x^0を未処理ノードとする．

(step3) 未処理ノードxを繰り返し選択し，３つのケースのいずれかで処理する．未処理ノードがなくなれば停止する．

<case1> Tの処理済みノードの中にxと同じベクトルのノードがあれば，xを複製ノードとする．

<case2> xにおいて遷移可能な$t\in\Sigma$が存在しない場合はxを終端ノードとする．

<case3> $x=(x_1,\ldots,x_k,x_{k+1},x_{k+2})$において遷移可能な全ての$t\in\Sigma$，および各tにおける全ての$s(x'_{k+1})\in\delta(s(x_{k+1}),t)$に対して，新しいノード$x'=(x'_1,\ldots,x'_k,x'_{k+1},x'_{k+2})$を以下の手順(3-3-1～3-3-5)で生成し，x'を未処理ノードとする．

(3-3-1) $x_i=\omega$ならば$x'_i=\omega$．(注：ωは無限を表し，任意の定数値aに対して，$\omega+a=\omega$，$\omega-a=\omega$，$a<\omega$，$\omega\leqq\omega$とする)

(3-3-2) ルートノードから$x=(x_1,\ldots,x_k,x_{k+1},x_{k+2})$までの経路上にあるノード$y=(y_1,\ldots,y_k,x'_{k+1},y_{k+2})$が，全てのi($1\leqq i\leqq k$)に対して$y_i\leqq\mu_i((x_1,\ldots,x_k),t)$，かつあるj($1\leqq j\leqq k$)に対して$y_j<\mu_j((x_1,\ldots,x_k),t)$であるとき，$x'_j=\omega$．

(3-3-3) それ以外のi($1\leqq i\leqq k$)の場合，$x'_i=\mu_i((x_1,\ldots,x_k),t)$．

(3-3-4) ルートノードから$x=(x_1,\ldots,x_k,x_{k+1},x_{k+2})$までの経路上にあるノード$y=(y_1,\ldots,y_k,x'_{k+1},y_{k+2})$が，全てのi($1\leqq i\leqq k$)に対して$y_i\leqq x'_i$，かつ「$y_{k+2}<x_{k+2}$あるいは『$y_{k+2}=x_{k+2}$かつ$s(x'_{k+1})\in F$』」のとき，$x'_{k+2}=\omega$とする．

(3-3-5) それ以外のx'_{k+2}は，

$$x'_{k+2}=\begin{cases}x_{k+2}+1 & s(x'_{k+1})\in F\\ x_{k+2} & \text{それ以外}\end{cases}$$

ペトリネットの到達可能木は有限であることが証明されている[2]．拡張到達可能木Tも等価な到達可能木に変換できるので有限である．生成されたTにおいて，$x_{k+2}=\omega$であるようなノードが１つでも存在すれば$L(N)\cap L(\neg f)\neq\phi$であり，存在しなければ$L(N)\cap L(\neg f)=\phi$である．■

3.6 nop導入で検証可能なこと不可能なこと

nopを導入することにより，デッドロックの可能性に関する検証は不可能になった，例えば，nopを含むペトリネットNのあるトランジションtに対して□◇t（無限回ときどきtが発火する）という制約はNに無関係に満たされない．（任意のNに対して$L(N)\cap L(\neg\Box\Diamond t)\ni$nop　が成り立つから）

デッドロックの可能性の判定問題は活性問題(liveness problem)と呼ばれ，決定可能ではあるが，到達可能木生成アルゴリズムのレベルの複雑さでは解けない．デッドロック可能性の検証については別の手法を用いることにする．

nopを導入した場合に，検証可能で有益な項目としては次のものがある：

●発火の禁止（例：□(t1⊃○□¬t2)）

●デッドロックの必然性（例：◇□¬t）．

●トランジション間の相対関係（例：t1,t2 は交互に発火する＝□(t1⊃○(¬t1Ut2))∧□(t2⊃○(¬t2Ut1)))

●状態のオーバーラップ禁止（例：区間（t1,t2)と区間(t3,t4)はオーバーラップしない＝□(t1⊃○(¬(t3∨t1)Ut2))∧□(t3⊃○(¬(t1∨t3)Ut4)))

3.7 MENDELネットの検証

ペトリネットと時相命題論理による検証手法を示したが，これはそのままMENDELネットに適用できる．ただし，MENDELネットにはオブジェクトによる階層性があるので，検証の対象となるトランジションを含まないオブジェクトに関しては縮約(reduction)してから検証する．縮約には[12]の手法が適用できる．

4．知的分散システム上の実現

現在，MENDELプログラムの実行系はPrologマシンPSI　上と，知的分散システム上に実装されている．PSI上の実行系は疑似並列環境である．ここでは，知的分散システム(IDPS:Intellectual Distributed Processing System)上の実装について簡単に述べる．

4.1 IDPSの概要

複数のパソコンをLANで接続し，管理分散を特色とするシステムである（図６）．各パソコンにはオブジェクト指向分散ＯＳが登載されている．分散ＯＳ上のプロセスはＣ言語で記述され，プロセス間通信は非同期放送方式で行われる．

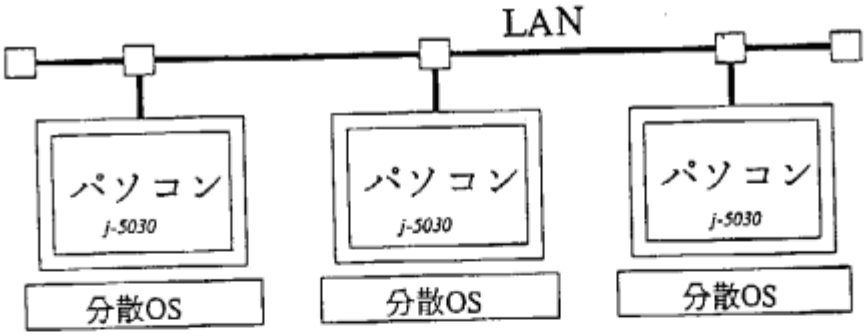

図６　知的分散システムのハードウェア構成

4.2 IDPS上のMENDEL実行系

MENDELのソースコードをIDPSのＣ言語にトランスレートする．MENDELの各オブジェクトがIDPSのプロセスとして実行される．そのプロセスは２種類のサブプロセスから構成される：

●本体部：MENDELのメソッド全体に対応する．MENDELオブジェクトの内部は逐次プログラムなので１つのサブプロセスになる．

●同期通信部：同期通信のためのサブプロセス．外部からのメッセージの受信用に本体部とは独立させる．本体部とはプロセス内のグローバルデータを介して情報を交換する．

4.3 同期通信方式

(1) ハンドシェイク型

オブジェクト間で同一化されたメソッドの基本的同期手順をまとめると図7のようになる.

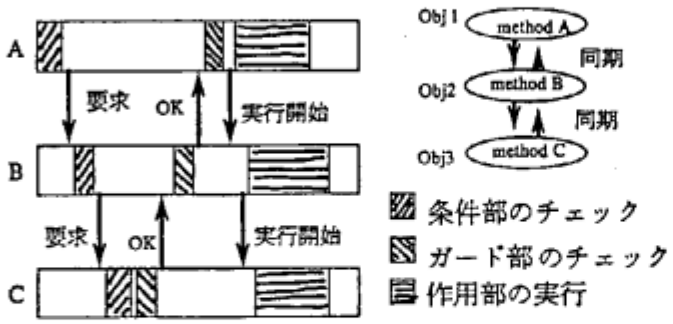

図7　メソッドの同期手順

(2) データフロー型

データの受信：同期通信部で受信しFIFOバッファに保存する．受信した結果はグローバルデータを経由して本体部で参照または読み込む．

データの送信：本体部の各メソッドからデータを送信する．

5．まとめ

ペトリネットに基づく並行プログラミング言語MENDEL/89を提案し，処理系を知的分散システム上に実装した．MENDELは強力な同期機構を持ち，視覚的にも理解しやすく，時相論理による検証も可能である．しかし，

（現状では）実行効率が悪く，かつ大きなプログラムだと検証に要する計算量が膨大になるので，分散協調システムのプロトタイプ作成用言語として有望である．現在，組立ロボットシステムの制御プログラムを対象に実用化を検討している．

本研究の一部はＩＣＯＴの再委託研究「並列プログラム構成支援システム」として行なわれた．
また，本論文は，電子情報通信学会のコンピュータ・システム研究会（１９８９年８月）で発表する．

参考文献

[1] R.Milner: Communication and Concurrency, Prentice Hall (1989).

[2] J.L.Peterson: Petri Net Theory and the Modeling of Systems,Prentice-Hall (1981).

[3] I.Suzuki,H.Lu:Temporal Petri Nets and Their Application to Modeling and Analysis of a Hand-shake Daisy Chain Arbiter,IEEE Trans. on Comput.,Vol.38,No.5 (1989).

[4] P. Wolper: Temporal Logic Can Be More Expressive,Information and Control 56 (1983).

[5] E.M.Clarke, E.A.Emerson, A.P.Sistla, Auto-matic verification of finite-state concur

rent systems using temporal logic specifications,ACM TOPLAS (1986).
[6] 片井 他：並列プロセスの動態に対する時制論理に基づく記述・検証体系，システムと制御,Vol.25,No.8 (1981).
[7] H. Tuominen: Temporal Logic as a Query Language for Petri Net Reachability Graphs, 7th European Workshop on Application and Theory of Petri Nets (1986).
[8] 田村 他：知的分散システムのアーキテクチャ，電気学会論文誌108-C[6](1988).
[9] 関 他：知的分散システムにおける高信頼化手法，電子情報通信学会コンピュータシステム研究会８月(1989).
[10] 内平 他：ペトリネットと時制論理による並列プログラミング言語,情処プログラミング言語研究会，17-2 (1988).
[11] P.Wolper,M.Y.Vardi,A.P.Sistla:Reasoning about Infinite Computation Paths,Proc. 24th IEEE FOCS (1983).
[12] K.Lee,J.Favrel:Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets, IEEE Trans.on SMC,Vol.SMC-15,No.2 (1985).