# π–計算を用いたエージェントのプランの形式化に関する研究

岩田員典[†] 伊藤暢浩[†]
杜小勇[††] 石井直宏[†]

エージェントが目的を達成するためには環境を観察し，環境から得られた情報を基に適切な行動列を作成・実行しなければならない．このような目的を達成するための行動列をプランと呼ぶ．さらに環境の変化を認識した場合にはプランを再構成する必要がある．しかし，これには時間がかかるのでプランを動的に変更することが望ましい．しかしながら，これまで提案されてきたエージェントモデルではプランの動的な変更については概念的に定義しているものがほとんどである．このためエージェントモデルの性質を実証するためには実験に頼らざるをえない．これはプランの動的な変更について形式的に表現することが難しいためである．そこで本論文では π–計算に着目し動的な変更が可能なプランの形式化を行った．また，π–計算の操作意味論を用いて性質の証明を行った．さらにプランを実際に使用するためのエージェントモデルを定義した．そして，これらの定義を用いて評価実験を行った．評価実験としては追跡問題を取り上げ，環境情報が一定確率でしか得られない状態で行った．これにより，環境の変化に合わせたプランの動的な変更が行われていることが確認できた．また，実験結果から定義したプランの記述方法が有効であることが確認できた．

# Formalization of Agents Plan by Using π-calculus

Kazunori Iwata,[†] Nobuhiro Ito,[†] Xiaoyong Du[††]
and Naohiro Ishii[†]

Agents must observe an environment around them and have to make a plan which is a set of action to satisfy their goals according to the information from the environment. If the environment is changed, agents must change the plan in a dynamic. However, the plan which has the dynamically-changing structure is conceptualy defined in most of the agent models which have been proposed. Because it is difficult to formalize how to describe the plan which has the dynamically-changing structure. In this paper, we use π-calculus to formalize a plan. Because π-calculus provides the dynamically-changing structures. First, we propose how to describe a play witten in π-calculus. Second, by using the operational semantics of π-calculus, we prove the properties of the plan. Third, we define the agent model to use the descriptions of the plan. Finally, we execute experiment with these definitions and show the plan is useful.

## 1. はじめに

近年，複雑に変化するような環境において適切に動作することを目的としたエージェント指向技術に注目が集まっている．このエージェント指向とは従来研究されてきたオブジェクト指向に次のような性質を加えたものである[1)]．

(1) 自律性
自分自身の知識や種々の情報を活用することにより自ら判断を下して行動する性質．

(2) 自発性
目標を達成するために自ら行動を起こす性質．自律性と関連が深い．

(3) 社会性
エージェントどうしもしくは人間とコミュニケーションを行うことにより協力しながら問題解決を行う性質．

(4) 反応性
環境の変化に応じて即応的に行動する性質．熟考的に行動する性質である自律性とは対照的な性質である．

この中でも特に自律性と自発性がエージェントを定義するうえで重要であると考えられている[1)]．

エージェントが自律的かつ自発的に行動するためには環境を観察する必要がある．そして環境から得られた情報を基に，目的を達成するための適切な行動列を

† 名古屋工業大学
Nagoya Institute of Technology
†† 中国人民大学
The Renmin University of China

作成・実行しなければならない．このような目的を達成するための行動列をプランと呼ぶ[2),3)]．さらに環境の変化を認識した場合にはプランを再構成する必要がある．しかし，これには時間がかかるのでプランを一部またはすべて動的に変更することが望ましい[4)~7)]．このようにエージェントが振る舞うためには動的に変更可能なプラン持つことと，それを作成・実行するためのエージェントモデルが必要である[8)~13)]．しかしながら，これまで提案されてきたエージェントモデルではプランの動的な変更については概念的に定義しているものがほとんどである．このためエージェントモデルの性質を実証するためには実験に頼らざるをえない．これはプランの動的な変更について形式的に表現することが難しいためである．そこで本論文では $\pi$–計算[14)~17)] に着目し動的な変更が可能なプランの形式化を行う．

本論文の構成は次のようになっている．次章で $\pi$–計算について説明する．3 章では提案するプランの表現を実際に実行するためのエージェントモデルを提案する．4 章で $\pi$–計算の操作意味論を用いて性質の証明を行う．5 章で $\pi$–計算を使って形式的にプランを表現する．6 章で実装面からも有効性の検証を行う．最後にまとめと今後の課題について述べる．

## 2. $\pi$–計　算

$\pi$–計算は名前通信機能を導入した CCS（the Calculus of Communicating Systems）の拡張系である．この機能によって動的なネットワークの表現が可能である．

### 2.1 $\pi$–計算の構文

$\pi$–計算では名前とプロセスという概念のみ存在する．名前が基本要素でありその組合せによってプロセスが定義される．以下にその定義を与える．ここで，$P$，$Q$ はプロセス，$a$，$b$ は名前とする．

| | | | |
|---|---|---|---|
| $P$ | := | $\overline{a}[\overline{v}].P$ | 出力 |
| | \| | $a(\overline{x}).P$ | 入力 |
| | \| | $P \mid Q$ | 並行合成 |
| | \| | $(\nu\ a)P$ | 名前制限 |
| | \| | $[a=b]P$ | マッチング |
| | \| | $P\ +\ Q$ | 外部非決定和 |
| | \| | $0$ | ヌル |
| | \| | $!P$ | 複写 |

### 2.2 操作意味論

以下に $\pi$–計算の基本的な操作意味論を示す[14)~17)]．

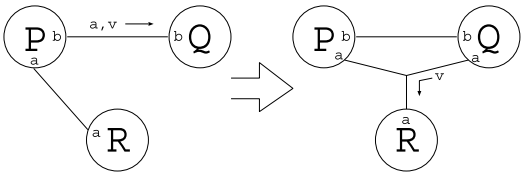


**図 1** ネットワーク構造の変化
Fig. 1 Changing the structure of the network.

ここで，$P$，$Q$，$R$ および $P'$，$Q'$，$R'$ はプロセス，$x$，$y$，$z$ は名前とする．

$$\text{COMM} \quad : \quad (\ldots + \overline{x}[\overline{y}].P) \mid (\ldots + x(\overline{z}).Q) \rightarrow P \mid Q\{\overline{y}/\overline{z}\}$$

$$\text{PAR} : \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \text{RES} : \frac{P \rightarrow P'}{(\nu\ x)P \rightarrow (\nu\ x)P'}$$

$$\text{STRUCT} : \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

### 2.3 ネットワーク構造の動的変化

操作意味論により提供されるネットワーク構造の動的変化について述べる．名前通信の例として次のプロセスを考える．

$$\begin{aligned} &(\nu\ a)(\nu\ b)P \mid Q \mid R \\ P &\stackrel{\text{def}}{=} \overline{b}[a].\overline{b}[v].P' \\ Q &\stackrel{\text{def}}{=} b(y).b(z).\overline{y}[z].0 \\ R &\stackrel{\text{def}}{=} a(x).R' \end{aligned}$$

操作意味論を用いて式変換を行うと次のようになる．

$$\begin{aligned} &(\nu\ a)(\nu\ b)P \mid Q \mid R \\ \rightarrow\ &\overline{b}[v].P' \mid\ b(z).\overline{a}[z].0 \mid\ a(x).R' \\ \rightarrow\ &P' \mid\ \overline{a}[v].0 \mid\ a(x).R' \rightarrow P' \mid\ R'\{x/v\} \end{aligned}$$

この様子を**図 1** に示す．名前 $a$ がプロセス $P$ からプロセス $Q$ に渡されることにより，プロセス $Q$ とプロセス $R$ の間に新たなリンク $a$ が生成される．その結果，プロセス $Q$ からプロセス $R$ に名前 $v$ を渡すことができる．このことから $\pi$–計算はネットワーク構造の動的変化を表現することができる．

## 3. エージェント モデル

本論文で定義するプランを使用するためのエージェントモデルを**図 2** のように定義する．

各ユニットの機能を以下で述べる．

(1) センサユニット

- 環境から環境情報を受け取る．

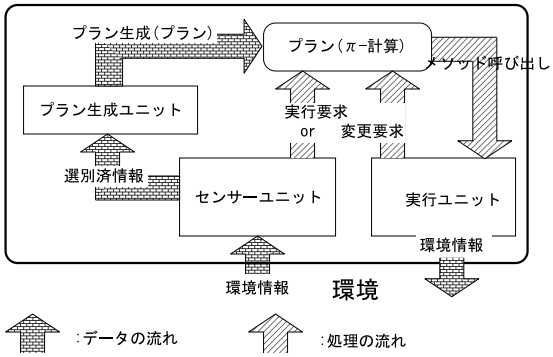


**図 2** エージェントモデル
Fig. 2 Agent model.

- 環境情報から必要な情報を選別しプラン生成ユニットに送る.
- 環境情報に従ってプランに実行または変更要求を送る.

(2) プラン生成ユニット
- センサユニットから選別済の情報を受け取る.
- センサユニットから受け取った選別済情報に従いあらかじめ与えられたプランを選択する.

(3) 実行ユニット
- プランからメソッドの呼び出しを受け取る.
- メソッドの呼び出しに従い実行した結果を環境情報として環境に送る.
- メソッドの実行結果からさらなるプランの実行もしくは変更が必要なとき，その要求をプランに送る.

センサユニットと実行ユニットが環境に対してフィルタの役割を果たす．これにより環境情報の形式が変わってもこの 2 つのユニットを変更するだけで対応できる．また，これらのユニットがエージェントの行動を制御する．つまり，それぞれのユニットが状況に応じてプランを実行または変更することによって適切な行動が選択できる．さらに 2 つのユニットから同時に要求が与えられたときに，どちらの要求を先に実行するかは π–計算の処理に依存する．これは π–計算の計算規則に従うと同時に要求が与えられた場合でも排他的に 1 つずつ実行するからである．

## 4. π–計算を用いたプランの表現

プランが持つ行動列をリスト表現にする．そして各行動間の関係を π–計算におけるネットワークで表現することにより構造の動的変化を可能にする．

### 4.1 π–計算によるプランの記述

まず，実行すべき行動であるメソッドを呼び出すためのプロセスを定義する．

**定義 4.1 メソッド 呼び出し**

$$Call \stackrel{\text{def}}{=} \texttt{exec}(\text{method}, \text{method}_{\text{end}}).\ Call(method).\overline{\texttt{method}_{\texttt{end}}}$$

$$Call(method) = \begin{cases} \text{何も呼び出さない} & method = \text{“nil”} \\ \text{method と名付けられた} \\ \quad \text{メソッドを呼び出す} & \text{その他のとき} \end{cases}$$

*Call* は π–計算の外部に存在するメソッドを呼び出すことを意味している．

次に，プランの基本的な要素の定義を行う．これはリストにおけるアトムに対応している．

**定義 4.2 基本要素**

$$\begin{aligned} &\text{Element}(\text{previous}, \text{method}, \text{next}) \\ &\stackrel{\text{def}}{=} \texttt{previous}.(\overline{\texttt{change}}[\text{method}].\texttt{change}(\text{method}). \\ &\quad \overline{\texttt{exec}}[\text{method}, \text{method}_{\text{end}}].\texttt{method}_{\texttt{end}}.\overline{\texttt{next}} \\ &\qquad + \ \texttt{execute}.\overline{\texttt{exec}}[\text{method}, \text{method}_{\text{end}}]. \\ &\quad \texttt{method}_{\texttt{end}}.\overline{\texttt{next}}) \end{aligned}$$

*Element* はプランを構成するための単位要素を意味している．これはリスト構造におけるアトムに相当しており直前と直後へのリンクも保持している．

`typewriter` 体で表された語句はリンクを示している．“*method*” は実行されるメソッドの名前を表す．“$method_{end}$” はメソッドの実行が終了したことを確認するための名前を表す．“*previous*” と “*next*” は他の要素へのリンクを表す．

定義 4.1，4.2 を使うことによりプランを定義する．プランが $method_1, method_2, \ldots, method_{n-1}, method_n$ を実行するものと仮定したとき，次のように定義する．

**定義 4.3 プランの定義**

$$\begin{aligned} &\text{Plan}(\text{method}_1, \text{method}_2, \ldots, \text{method}_{n-1}, \text{method}_n) \\ &\stackrel{\text{def}}{=} \quad (\nu \ \overrightarrow{n} \ \text{nil})\Big(\text{Element}(\text{start}, \text{method}_1, \text{n}_1) \\ &\qquad | \quad \text{Element}(\text{n}_1, \text{method}_2, \text{n}_2) \\ &\qquad\qquad\qquad \vdots \\ &\qquad | \quad \text{Element}(\text{n}_{i-1}, \text{method}_i, \text{n}_i) \\ &\qquad\qquad\qquad \vdots \\ &\qquad | \quad \text{Element}(\text{n}_{n-2}, \text{method}_{n-1}, \text{n}_{n-1}) \\ &\qquad | \quad \text{Element}(\text{n}_{n-1}, \text{method}_n, \text{nil}) \\ &\qquad | \quad \texttt{nil} \ | \ !\text{Call} \ \Big) \end{aligned}$$

ここで，“$\overrightarrow{n} \equiv n_1 \ n_2 \ \ldots n_{n-1} \ n_n$” とする．また，“*nil*” はプランの最後を示している．

*Plan* は引数にとったプロセスを順次実行することを意味する．しかし，その実行順序は外部からの変更要求によって変更可能である．

### 4.2 プランの実行と変更要求

プランは作成された後に適切なタイミングで実行されなければならない．そこで，まずプランを実行状態にするためのプロセスを定義する．次に，プランを構成するメソッドを順次実行するための実行要求を定義する．

**定義 4.4 プランの開始**

$$\text{Start} \stackrel{\text{def}}{=} \overline{\texttt{start}}$$

このプロセスと並行合成を行うことによりプランは実行可能になる．

**定義 4.5 実行要求**

$$\text{Execute} \stackrel{\text{def}}{=} \overline{\texttt{execute}}$$

これはプランを構成するメソッドを 1 つだけ実行するためのプロセスを定義する．つまり，プランをすべて実行するにはメソッドと同数のプロセスが必要である．しかしながら，これによりプランの変更を定義することが容易になる．

次に，プランを変更するためのプロセス “Substitute”，“Add”，“Delete” を定義する．“Substitute” はプランにおけるメソッドの置き換えを行う．“Add” はプランにメソッドの追加を行う．“Delete” はプランからメソッドの削除を行う．各プロセスの詳細については 5 章で述べる．

**定義 4.6 メソッドの取り替え要求**

$$\text{Substitute}(\text{method}') \stackrel{\text{def}}{=} \texttt{change}(\text{method}).\overline{\texttt{change}}[\text{method}']$$

**定義 4.7 メソッドの追加要求**

$$\text{Add}(\text{method}') \stackrel{\text{def}}{=} \texttt{change}(\text{method}).\overline{\texttt{exec}}[\text{method}', \text{method}'_{\text{end}}].\texttt{method}'_{\text{end}}.\overline{\texttt{change}}[\text{method}]$$

**定義 4.8 メソッドの削除要求**

$$\text{Delete} \stackrel{\text{def}}{=} \texttt{change}(\text{method}).\overline{\texttt{change}}[\text{nil}]$$

最後に，定義 4.1〜4.8 で用いた名前の直観的な意味を説明する．

*start* プランを実行可能にするために最初に使用されるリンクを意味する．

*exec* 外部のメソッドを呼び出すためにプロセス *Call* へ実行要求を送るためのリンクを意味する．

*execute* メソッドを実行をするきっかけとなる名前を意味する．

*method* 呼び出したいメソッドの名前を意味する．

*previous* プランに属する，ある要素の直前の要素へのリンクを意味する．

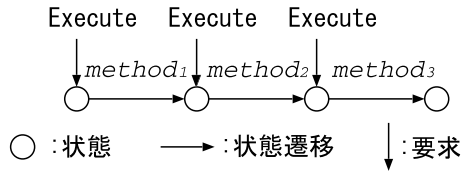


**図 3** 実行要求の例
Fig. 3 Example of “Execute”.

*change* プランを変更するきっかけとなる名前を意味する．

*next* プランに属する，ある要素の直後の要素へのリンクを意味する．

## 5. プランの性質

定義 4.5，4.6，4.7，4.8 により得られる性質を定理 5.2，5.4，5.6，5.8 として説明する．

**定理 5.1 プランの開始**

任意のプラン $Plan(method_1, \ldots, method_n)$ が存在するときにプロセス “*Start*” と並行合成を行うことによりそのプランが開始され実行可能になる．

**定理 5.2 メソッドの実行**

実行可能になったプラン $Plan(method_1, \ldots, method_n)$ が存在するときにプロセス “*Execute*” と並行合成を行うことにより $method_1$ が実行されプランは $Plan(method_2, \ldots, method_n)$ になる．すべてのメソッドを実行するには $n$ 個の実行要求が必要である．

**例 5.3 メソッドの実行**

$Plan(method_1, method_2, method_3)$ があったときに実行要求を受け取った場合の処理を**図 3** に示す．

**定理 5.4 メソッドの置き換え**

実行可能になったプラン $Plan(method_1, \ldots, method_n)$ が存在するときにプロセス “$Substitute(method_{new})$” と並行合成を行うことにより $method_{new}$ が実行されプランは $Plan(method_2, \ldots, method_n)$ になる．

**例 5.5 メソッドの置き換え**

$Plan(method_1, method_2, method_3)$ があったときに実行要求と置き換え要求を受け取った場合の処理を**図 4** に示す．

**定理 5.6 メソッドの追加**

実行可能になったプラン $Plan(method_1, \ldots, method_n)$ が存在するときにプロセス “$Add(method_{new})$” と並行合成を行うことにより $method_{new}$ と $method_1$ が実行され，プランは $Plan(method_2, \ldots, method_n)$ になる．

**例 5.7 メソッドの追加**

$Plan(method_1, method_2, method_3)$ があったときに実

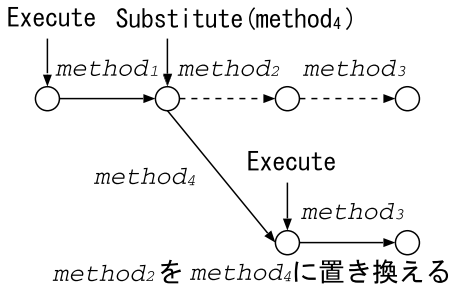


**図 4** メソッドの置き換え要求の例
Fig. 4 Example of “Substitute”.

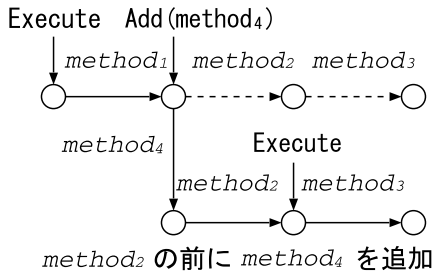


**図 5** メソッドの追加要求の例
Fig. 5 Example of “Add”.

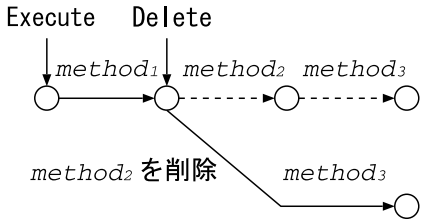


**図 6** メソッドの削除要求の例
Fig. 6 Example of “Delete”.

行要求と追加要求を受け取った場合の処理を**図 5** に示す.

**定理 5.8 メソッドの削除**

実行可能になったプラン $Plan(method_1, \ldots, method_n)$ が存在するときにプロセス “Delete” と並行合成を行うことによりプランは $Plan(method_2, \ldots, method_n)$ になる.

**例 5.9 メソッドの削除**

実行可能になったプラン $Plan(method_1, method_2, method_3)$ があったときに実行要求と削除要求を受け取った場合の処理を**図 6** に示す.

## 6. 実　　験

### 6.1 実験環境

3 章, 4 章で定義したプランの表現方法とモデルの評価実験を行う.

**表 1** 実験環境
Table 1 Conditions of experiments.

| | |
|---|---|
| 環境の概要 | N × N の格子点をエージェントは移動できる |
| 終了条件 | 両方のエージェントが隣接する格子点に<br>存在したとき目標物を捕えたとして終了する |
| 評価基準 | 追跡者が標的を捕えるのにかかった移動回数 |
| 共通の能力 | 自分の場所と相手の場所の認識ができる.<br>隣接する格子点への移動ができる |
| 追跡者のプラン | 目標物の位置情報が得られたら<br>その地点へ移動するためのプランを構成する |
| 目標物のプラン | 追跡者の場所を把握し<br>一定確率で遠ざかる方向に移動する |

評価方法としては標的を捕える追跡問題を取り上げた. この実験では追跡するエージェントとその追跡から逃げるエージェントの 2 種類が存在する. 基本となる実験環境を**表 1** に示す.

表 1 に示した実験環境のもとで 3 種類の実験を行った.

**最適な環境** 追跡者がつねに目標物の位置を把握できる環境

目標物の位置をつねに把握しているのでプランを生成するのではなく各ステップごとに目標物へ向けて移動する.

**最適でない環境** 追跡者が目標物の位置を把握できるとは限らない環境

(1) 追跡者は書き換え可能なプランを所有
- プランがない場合は目標物の位置を把握した時点でプランを生成する.
- 原則として一度生成したプランはすべて実行する. しかしながら, 新たな情報を得たときには随時プランの書き換えを行う.

(2) 追跡者は書き換え不可能なプランを所有
- プランがない場合は目標物の位置を把握した時点でプランを生成する.
- 原則として一度生成したプランはすべて実行する. しかしながら, 新たな情報を得たときにはプランを作り直す.

プランは環境情報が得られたときに認識した目標物まで移動するための最短の格子点を結んだものである. このとき, エージェントが実行できるメソッドは隣接格子点への移動であるため, 格子点を N 回移動しようと思ったときには N 回メソッドを呼び出す必要がある.

最適な環境とそうでない環境の大きな違いは目標物の位置をつねに把握できるかそうでないかである. 追跡問題においては相手の位置情報がプランに大きく関

**表 2** 実験結果
Table 2 Results of experiments.

| 広さ N (N × N) | 50 | 100 | 200 | 300 |
|---|---|---|---|---|
| 最適な環境 | 116.51 | 243.98 | 493.26 | 741.93 |
| 書き換え可能 | 151.41 | 313.64 | 639.99 | 947.7 |
| 書き換え不可能 | 229.77 | 463.32 | 936.55 | 1406.25 |
| 広さ N (N × N) | 400 | 500 | 600 | 700 |
| 最適な環境 | 986.00 | 1239.77 | 1487.08 | 1733.06 |
| 書き換え可能 | 1265.78 | 1595.34 | 1923.05 | 2224.63 |
| 書き換え不可能 | 1902.77 | 2358.61 | 2844.17 | 3347.41 |
| 広さ N (N × N) | 800 | 900 | 1000 | |
| 最適な環境 | 1982.96 | 2230.37 | 2481.66 | |
| 書き換え可能 | 2562.88 | 2870.39 | 3213.54 | |
| 書き換え不可能 | 3757.39 | 4253.91 | 4715.25 | |

わってくる．つまり，目標物の位置をつねに把握できればプランを生成しなくとも追跡するという行動を容易に行うことができる．しかしながら，位置情報がいつ得られるか分からないような環境においては，追跡を行うために最新の情報を使ってプランを生成しなければならない．このためエージェントはプランの実行を一時停止してプランの作り直しを行う必要がある．このような場合において実行中のプランを書き換えることができれば計算時間の無駄を省くことができる．この際，プランの変更にかかる時間よりもプランの生成にかかる時間が短い場合は生成したほうがよい．一般的にはプランの生成は変更に比べてかなりの時間を要する[4),5)]．

本論文で提案したモデルでは，プラン生成ユニットによりプランが作成された後にセンサユニットから `start` が送られプランが実行状態になる．そして，実行状態になったプランには最初はセンサユニットから実行要求が送られる．一度，実行が始まると実行ユニットから実行要求が送られる．プランの変更はセンサユニットが環境から新たな情報を得たときに行われる．

このときに，実行要求と変更要求により非決定的にどちらかの行動が行われる．ここで，非決定的にしているのは実行要求と変更要求のどちらでにも対応できるようにするためである．これは変更要求と実行要求のどちらが送られてくるか分からないうえ，つねに送られてくる可能性があるので決定的にすることができないからである．そのため非決定的に制御する機構が必要である．

また，非決定和を使うことによってこれらの要求のうちどちらかの実行中はもう片方を実行しないという排他性にすることで，無駄な計算を行わないようにできる．さらに出力する環境情報を計算する実行ユニットの実装が簡潔になる．

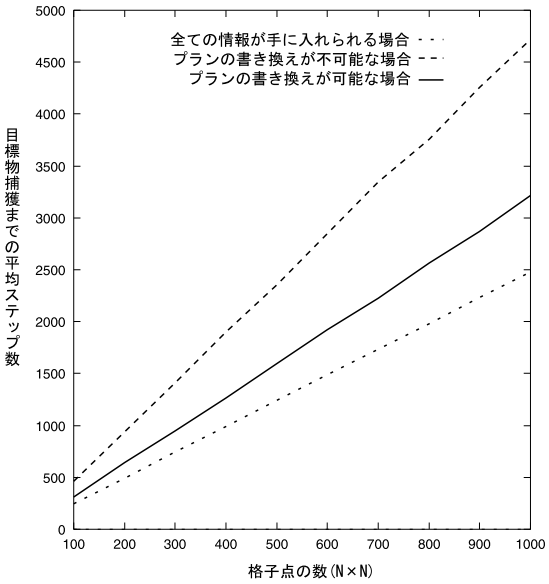


**図 7** 広さを変えた場合の実験結果
Fig. 7 Results in many states.

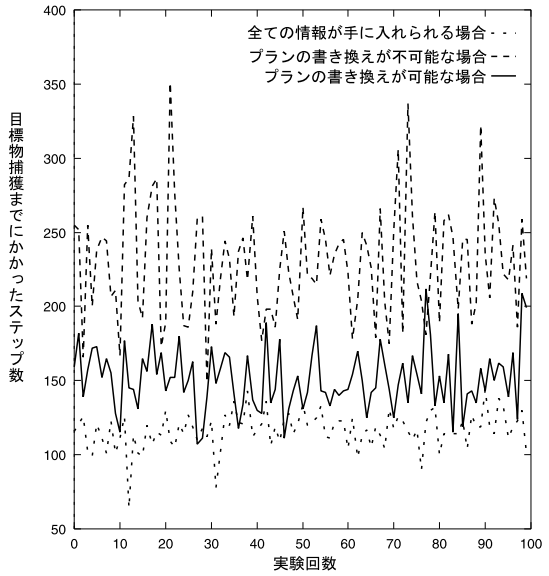


**図 8** 50 × 50 における実験結果
Fig. 8 Results in 50 × 50.

### 6.2 実験結果

6.1 節で示した条件に従い，様々な値の格子点の数で実験を行った．100 回実験を行った結果の平均を**表 2** に示す．それをグラフにしたものを**図 7** に示す．

さらに，50 × 50 の広さで行った結果を**図 8** に示す．

これらの結果から部分的な情報しか得られないことによりプランの動的な変更が必要となる環境において本論文の手法が有効であることが分かる．

## 7. ま と め

本論文ではまず，エージェントが動的に変化する環境に対応するために必要なプランの形式的な表現をπ–計算を用いることによって定義した．この定義はπ–計算が持つ構造の動的な変化という性質を利用することにより，プランの動的な変更を可能にした．さらにπ–計算の操作意味論を用いることによりその性質の証明を行った．

次に，その表現を実際に実行するためのエージェントモデルの定義を行った．これはプランを実行するだけでなく，環境情報の形式が変更された場合にも対応できる．

さらにこれらの定義の有効性を検証するために評価実験を行った．実験結果からプランの変更が必要な環境において有効であることが示せた．

本論文では簡単なプランを持つエージェントを用いて実験を行った．そのためさらに複雑なプランを持つ場合について検討するつもりである．また，学習機能を追加した場合について検討していくつもりである．

**謝辞** この研究成果の一部は文部省の科学研究費による．

## 参 考 文 献

1) Shoham, Y.: Agent-Oriented Programming, *Artif. Intell.*, Vol.60, pp.51–92 (1993).
2) 山田誠二：エージェントのプランニング，人工知能学会誌，Vol.10, No.5, pp.677–682 (1995).
3) 山田誠二：リアクティブプランニング，人工知能学会誌，Vol.8, No.6, pp.729–735 (1993).
4) Bell, J. and Huang, Z.: Dynamic Goal Hierarchies, *Intelligent Agent Systems*, Lawrence, C.A. and Wobcke, W. (Eds.), LNAI, No.1209, pp.88–103, Springer (1997).
5) Au, S. and Parameswaran, N.: PLAN EXECUTION IN A DYNAMIC WORLD, *International Conference on Computing and Information 98*, pp.331–338 (1998).
6) Blythe, J.: Planning with External Events, *UAI-94*, pp.94–101 (1994).
7) Branskat, S. and Unger, C.: Plan Execution in a Multi Agent System in an Uncertain Environment, *4th International Conference on Artificial Intelligence Planning System*, pp.1–6 (1998).
8) Morley, D.: Semantics of BDI Agents and Their Environment, *Intelligent Agent Systems*, Lawrence, C.A. and Wobcke, W. (Eds.), LNAI, No.1209, pp.119–134, Springer (1997).
9) Ito, N., Nakagawa, K., Hotta, T., Du, X. and Ishii, N.: EAMMO: An environmental agent model for multiple objects, *Information and Software Technology*, Vol.40, No.7, pp.397–404 (1998). ELSEVIER.
10) Haddadi, A. (Ed.): *Communication and Cooperation in Agent System*, LNAI, No.1056, Springer (1996).
11) Lee, J. and Durfee, E.H.: On Explicit Plan Language for Coordinating Multiagent Plan Execution, *Intelligent Agent Systems IV*, Munindar, P., Singh, A. and Wooldridge, M.J. (Eds.), LNAI, No.1365, pp.113–126, Springer (1997).
12) Takeuchi, K.H. and Kudo, M.: An Interaction-based Language and its Typing System, LNCS 817, pp.398–413, PARALE'95 (1995).
13) Moreno, A. and Sales, T.: Limited Logical Belief Analysis, *Intelligent Agent Systems*, Lawrence, C.A. and Wobcke, W. (Eds.), LNAI, No.1209, pp.104–118, Springer (1997).
14) 吉田展子，久保 誠，本田耕平：π–計算とその周辺，情報処理，Vol.37, No.4, pp.319–326 (1996).
15) Milner, R.: Polyadic π-calculus: a Tutorial, LFCS Report Series ECS-LFCS-91-180, Laboratory for Foundation of Computer Science (1991).
16) Pierce, B.C. and Turner, D.N.: Concurrnet Objects in a Process Calculus, LNCS 907, pp.187–215, TPPP '95 (1995).
17) Milner, R., Parrow, J.G. and Walker, D.J.: A Calculus of Mobile Processes, *Information and Computation*, 100(1), pp.1–77 (1992).

(平成 11 年 12 月 1 日受付)
(平成 13 年 7 月 2 日採録)

**岩田 員典**（学生会員）

1975 年生．1998 年名古屋工業大学知能情報システム学科卒業．2000 年同大学大学院電気情報工学専攻博士前期課程修了．同年同大学院電気情報工学専攻博士後期課程入学．現在，在学中．エージェント指向技術，計算論，プログラミング言語設計に興味を持つ．人工知能学会会員．

**伊藤　暢浩**（正会員）

1970 年．1994 年名古屋工業大学電気情報工学科卒業．1996 年同大学大学院博士前期課程修了．1999 年同大学院博士後期課程修了．博士（工学）．同年 4 月名古屋工業大学電気情報工学科助手．現在に至る．オブジェクト指向技術，エージェント指向技術に興味を持つ．電子情報通信学会，人工知能学会，IEEE 各会員．

**杜　　小勇**（正会員）

1963 年生．1997 年名古屋工業大学大学院電気情報工学専攻博士課程修了．工学博士．同年名古屋工業大学知能情報システム学科助手．現在，中国人民大学コンピュータサイエンス学科学科長，教授．データベース，情報検索，知能情報システム等の研究に従事．IEEE CS，中国計算機学会 CCF 各会員．

**石井　直宏**（正会員）

1940 年．1968 年東北大学工学部電気工学科卒業．1973 年同大学大学院工学研究科博士課程修了．工学博士．東北大学医学部助手，名古屋工業大学電気情報工学科助教授を経て，現在，同大学知能情報システム学科教授．この間，しきい値論理，生体情報処理，非線形システム解析，ニューラルネットワークの研究に従事．電気学会，電子情報通信学会，神経回路学会，IEEE 各会員．