マルチエージェント環境で他者の信念の変遷を推定する前進的アルゴリズム

磯崎秀樹†

人間同士の会話では，各人が他者の信念を推定することによって円滑なコミュニケーションを実現している．したがって，人間が他者の信念をどのように推定しているかを解明し，計算機上にその仕組みを実装することは，今後のユーザインタフェースの改良に大きく役立つ可能性がある．各人の信念は，その人がどのような情報を観測したかに大きく左右され，他者の信念を推定する場合にも，その人が何を観測したかの情報が大きな手がかりになる．そこで我々は，観測可能性に基づいて他者の信念を推定する遡行的アルゴリズムをこれまでに提案した．このアルゴリズムはデータ量の多項式時間で他者の信念を推定することができる．しかし，大量のデータを扱おうとすると，さらに効率の改善が必要である．いくつかの例を解析した結果，このアルゴリズムは無関係な変化に対して必要のない計算を行うという冗長性があるということが判明した．しかし無関係かどうかの判断は必ずしも簡単ではない．そこで本論文では，この問題を解決した別のアルゴリズムを提案する．さらに，以前のものが最新の時点での信念しか推定しないのに対し，本論文のアルゴリズムは，これまでの信念の変遷も同時に推定できるという特長を備えている．実験により，推定結果に関係のない変化の処理時間が大幅に削減されていることが確認された．

A Progressive Algorithm for Estimating Other Agents' Belief Histories in Multi-agent Environments

HIDEKI ISOZAKI†

In human conversations, people estimate others' beliefs in order to communicate smoothly with one another. Hence, if we shed light on how people estimate others' beliefs and implement such a mechanism on a computer, its user interface will be improved. We know one's belief heavily depends on one's observation. And we use this fact when we estimate others' beliefs. In a previous paper, we proposed a regressive belief estimation algorithm based on observability. Complexity of this algorithm is polynomial with respect to the size of given data. However, it should be more efficient for large data. By analyzing some cases, we found that this algorithm executes unnecessary computation for irrelevant changes. And it is not always easy to judge relevance of a change. In this paper, we present another algorithm that is more sophisticated in this respect. Moreover, the new one computes whole belief histories while the old one computes only current beliefs. Our experiments show that the new one takes much less time for irrelevant changes.

1. はじめに

人間同士の会話では，円滑なコミュニケーションを実現するため，聞き手などの他者の信じていること（信念）を推定することが重要である[1]．しかし，時間と信念の両方の様相演算子を含む様相論理は，そのままでは処理時間が膨大にかかり現実的でない．また，論理的に導きうることがすべて信念になってしまうという「論理的全知」の問題もある[7]．

そこで我々は，観測によって得られた情報から他者の信念を推定する遡行的アルゴリズムを提案した．そして，そのアルゴリズムを正当化する意味構造も提案した[11)~13)]．このアルゴリズムはデータ量の多項式時間で他者の信念を推定することができるという特長を持つ．我々はこのアルゴリズムを自然な日本語の生成に役立てたり[9]，ユーザの知らない情報を要約して伝えるなどの処理に利用することを考えている．

しかし，このアルゴリズムは，基本的なアイデアをそのままコーディングしただけで，処理上の工夫としては，結果を記録してまったく同じ計算を避けるというごく簡単な処理しか行っていない．ユーザとのイン

† NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories

タラクションが長くなるに従って事象数が増え，何百・数千のデータが与えられると，これだけでは処理時間がかかりすぎることが，簡単な会話の実験によって明らかになってきた．

本論文では，これまでのアルゴリズムと同じ意味構造をベースにしながら，無駄な計算を排除した別のアルゴリズムを提案し，処理時間の傾向を比較する．従来のアルゴリズムでも，昔の事象を無視し，ごく最近の事象だけを考慮することによって計算量を減らすことはできるが，人間の忘却のメカニズムにそぐわない形で昔の事象を忘れてしまうと，違和感が生じることが予想される．そこで忘却によらずに高速化する方法を考える．

以前のアルゴリズムはトップダウン型で，最近の観測から相手の信念を推定できないときには，過去の情報を思い出すという遡行的なものである．ところが，過去のある時点から現在までに，推定結果に影響する事象がまったく発生していないことが少なくなく，そのような場合に無駄な計算が行われるのである．

我々の比較的初期の実装[8]では，ある事象が推定結果に影響するかどうか経験則により判断することで計算を高速化したが，実際には同じ事象でも領域知識やそれまでの経緯次第で関係するかどうかが決まるため，このようなヒューリスティクスによる方法では不正確な結果が出てしまうことがある．

本論文ではこの点を改良し，初期状態から現在に向かって推論を進める前進的なアルゴリズムを提案する．そのため，まず関数表現を導入して，複数の命題を1つにまとめる．そして前進的に信念推定の計算を進め，実際に計算に必要になった関数の値の変化点だけに着目して計算を行うことで無駄を排除する．

なお本論文のアルゴリズムは，高速化のためにさまざまな工夫を取り入れており，意味構造との対応の証明がかなり繁雑になることが予想される．そこで本論文では，高速化技法の説明と実行時間の実験結果だけを報告することにして，意味構造との対応の証明は別の機会に譲る．

1.1 従来のアルゴリズム

遡行的信念推定アルゴリズムの詳細は論文11)~13)を参照されたい．ここでは新しいアルゴリズムを提案するうえで必要最低限の説明だけを行う．

遡行的アルゴリズムは，以下のような考え方に基づいて，特定の命題に関するあるエージェントの現在の信念を計算する．

観測 いま観測できている命題（fluent[15]）は信じる．はじめてある部屋に入った場合などには，自分が予想していなかった情報が自分の周囲を観測することにより得られる．

影響 たったいま発生し，観測した事象の結果，必ずただちに成立することが期待できる命題（事後条件）は，たとえそれが観測できない状態であっても信じる．たとえば，Johnが図書館に入っていくところを見かけたら，Johnが図書館の中にいるところが見えなくても，Johnは図書館にいると信じる．

記憶 上記2つの新しい情報によって肯定も否定もされない命題については，それまでの信念を引き継ぐ．

最後の「記憶」で引き継ぐ「それまでの信念」は，最新の事象が発生する直前の状態では最新の信念であるから，やはりこれも，その時点で「観測」できた命題と，その直前に発生し観測した事象の「影響」と，さらにその前の「記憶」から計算できるはずである．同様にして，指定された命題を否定するか肯定する観測結果が得られるまで，必要に応じて過去に遡っていくことによって，現在の信念が計算できる．

「記憶」が正当化されるのは，ここで考えるすべての命題変数が持続性を持っているという前提による．つまり，各命題変数は，いったん真または偽になれば，とくに明示的な理由がない限り，ずっとその状態を維持すると考えているのである．一方,「ディジタル時計がちょうど0時0分0秒を表示している」という命題は，時計が止まっていない限り，次の瞬間には成立しない．つまり,「記憶」が正しく保持されても，現状が変わってしまえば偽の情報なので，持続性の期待できない命題をここでは信念の対象と考えない．

さて，他のエージェントの信念を推定する場合も，上記の計算方法にしたがって計算するが，そのエージェントが何を観測できたか正確には分からない．そこで，どのエージェントがどのような状態で何を観測できるか表した**「観測条件」**というドメイン依存の知識を用いて，問題となっている命題や事象が推定対象のエージェントに観測できたかどうか判断する．

自然言語処理では，Johnの信念に関するBobの信念のような，いわゆる「ネストした信念」が扱われる[1]が，この場合も上記の考え方に従って計算できる．つまりJohnが何を信じているかをBobが推定するには，Johnの観測をBobが推定すればよい．つまり，Johnの観測条件の真偽をBobの信念に従って計算すればよい．

「観測」の概念はきわめて一般的であり，どのような情報の入手にも関係している．たとえばコンピュー

タのグラフィカルインタフェースでも，表示されていない情報はユーザには観測できず，ユーザの信念に変化を及ぼさないと考えられる．逆に非常に目立つ表示であれば，ユーザは多分それを見逃さないであろうと考えられる．音声による会話の場合も，その会話の聞こえる範囲にいない人には情報が伝わらないが，マイクがあればその場にいない人も聞くことができる．このように，観測できるかどうかは様々な場面で信念の変化を左右しているため，我々はとくに観測条件に着目しているのである．

もっと深くネストした信念が要求される場合もあると考えられている[3),4)]．そこでエージェント a_1 の信念におけるエージェント a_2 の信念における…エージェント a_k の信念（様相論理の記号を用いれば $B_{a_1}B_{a_2}\cdots B_{a_k}\phi$）を扱うために，この場合の推定対象のエージェントを $(a_1, a_2, \ldots, a_k)$ というエージェント名のリストで表す．これを「**エージェント添字列**」と呼ぶ．そして，$(a_1, \ldots, a_k)$ で表されるエージェントに対して，$(a_1, \ldots, a_{k-1})$ で表されるエージェントの信念をエージェント $(a_1, \ldots, a_k)$ の「**外界**」と呼ぶ．また，エージェント添字列が空 () の場合は，推論システム自身の信念を表す．

遡行的アルゴリズムは，システムの持つ以下の領域知識を利用して他者の信念を推定する．

命題間の競合関係 命題変数 p と同時に真にならない命題変数の集合を $n(p)$ で表し，p の「**競合命題集合**」と呼ぶ．

各事象 e の事後条件 事象 e によって真になる命題の集合を $\mathsf{Post}^+(e)$，偽になる命題の集合を $\mathsf{Post}^-(e)$ で表し，その事象の「**事後条件**」と呼ぶ．

システム自身が各時刻に観測した命題の集合 時刻 t に真であることが観測された命題変数の集合を $O^+(t)$，偽であることが観測された命題変数の集合を $O^-(t)$ で表す．

システム自身が観測した事象系列 時刻 $t-1$ の状態から時刻 t の状態に移るときに発生した事象を e^t で表す．$e^1, \ldots, e^m$ を「**事象系列**」と呼び，m を「**事象系列の長さ**」という．複数の事象が同時に発生することはなく，各事象は一瞬で終了するものと仮定する．

命題と事象の観測条件 命題変数 p（事象 e）が a さんに観測できるための条件を表す命題論理式（様相演算子を含まない）を $\mathsf{ob}[a,p]$（$\mathsf{ob}[a,e]$）で表し，その命題（事象）の「**観測条件**」と呼ぶ．$\mathsf{ob}[a,p]$ はその時刻の世界に関する条件であり，$p \wedge \mathsf{ob}[a,p]$ が成立する場合に a さんは p を信じ，$\neg p \wedge \mathsf{ob}[a,p]$（$\neg p \wedge \mathsf{ob}[a,\neg p]$ ではない）が成立する場合に $\neg p$ を信じるものとする[11)]．一方 $\mathsf{ob}[a,e]$ はその事象の発生する直前の状態の世界に関する条件とする．本論文ではとくに，観測条件が AND と OR だけで構成され，NOT を含まない単調関数であると仮定する．

遡行的アルゴリズムはこれらのデータを与えられたのち，推定対象となるエージェントを表す添字列 $(a_1, \ldots, a_k)$ と命題 p を問い合わせられると，上記の考え方に従って，トップダウンにそのエージェントの現在の信念を計算して答を返す．

つまり特定の命題変数 p の真偽に関するエージェント $(a_1, \ldots, a_k)$ の信念を推定し，p を真と信じていれば yes を，偽と信じていれば no を，どちらでもなければ unknown を出力する．

上記のように，遡行的信念推定アルゴリズムは，自然言語の曖昧さなどのために取扱いの難しい伝聞情報の処理をとりあえず無視して，他者の信念の推定を，命題や事象の観測条件の再帰的計算に置き換えるというアイデアに基づいている．もちろん伝聞も観測を通じて得られるため，ごく単純な伝聞情報の受理機構をこの枠組みに加えることは可能である[10)]．

このアルゴリズムは，論理的な観点からすると，事前条件の成立の確認や，観測していない事象の想定を省くことにより，処理に時間のかかる歴史的な整合性の確認を回避している．また，信念の満たすべき「**一貫性制約（integrity constraints）**」[17)] として，3つ以上の命題変数を含む制約を考えないことにより，信念の変更を容易にしている．

これらの単純化により，従来の信念の様相論理では達成しえなかった多項式時間での信念推定が可能になった．そして，その推定品質を定量的に評価した結果，ある程度もっともらしい答が得られることが分かった[11)]．

1.2 遡行的アルゴリズムにおける高速化とその問題点

論文 11) では，上記の遡行的アルゴリズムの実装方法について述べた．トップダウン計算に共通する問題であるが，上記のアイデアのとおりに実装したのでは，同じ部分問題を複数回解くことがあるため無駄である．そこで，すでに解いた部分問題をハッシュ表に記録して，まったく同じ計算をしないようにした．これにより，簡単な例でも数十倍高速化される場合があることが分かった．また，必要な場合にのみ過去に遡って調べるというアルゴリズムの性質から容易に予測されることではあるが，対象命題がつい最近観測されている

場合は，しばらく観測されていない場合に比べ，はるかに短い処理時間ですむことが確認された．

また，論文11)には明示的に書かなかったが，この実装ではさらに次のような高速化を行っている．上記のアルゴリズムに従うと，他者の信念を1つ計算するために，システム自身の信念が何百回，何千回とアクセスされる．そこで，システム自身の信念だけは，あらかじめ各命題変数に対して信念が変化した時刻を記録しておき，時刻の比較だけで計算できるように改良した．

しかし，これだけで冗長な計算が十分除去されたわけではない．とくに問題となるのは，推定結果に影響しない事象の発生による計算量の増大である．推定結果に関係のない事象が多いと，最新の時刻 m で観測条件が成立せず，1つ前に戻って $m-1$ でも観測条件が成立せず，$m-2$ でも…$m-r$ でも成立せず，という状況が頻繁に発生する．この場合，それぞれの観測条件の計算はトップダウン的に必要な計算だけが行われているが，関係ある命題は何も変わっていないので，その計算結果は変わらない．それにもかかわらず，遡行的アルゴリズムは時間を遡るたびに観測条件の計算を呼び出してしまう．

この無駄は，遡行的アルゴリズムの各関数に与えられる引数が毎回違うので，計算結果の再利用では回避できない．計算の内容を考えてアルゴリズムを改良しなければ，この無駄をなくすことはできないのである．本論文で除去しようとしているのは，まさにこの無駄である．

たとえば，Annが食卓のうえにノートパソコンを置いたあと，Bobと一緒に隣の書斎に移動し，そこでデスクトップパソコンにアルファベット26文字を入力したとしよう．なお，書斎から食卓は見えないものとする．ここで，ノートパソコンの場所についてAnnがどう思っているかについてBobが推定すると，遡行的アルゴリズムによれば，Annは「食卓の上にまだある」と信じている，というBobの推定結果が得られる．

これはもっともらしい答であるが，その計算過程が問題である．各文字の入力をそれぞれ別の事象として考えると，遡行的アルゴリズムはこの26個の文字入力を1つずつ遡っていき，そのたびにノートパソコンの所在に関する観測条件を計算することになる．これらの文字入力を処理し終わり，AnnとBobが食卓の見える場所にいる時点まで遡って，はじめてノートパソコンが食卓の上にあるという結果が得られる．

つまり，遡行的アルゴリズムは，観測された事象の数のほとんどが質問と無関係な場合には，きわめて無駄の多い計算方法である．もともと，持続性を持つ命題はあまり頻繁に変化するものではないため，計算結果に影響しない事象が事象系列の大半を占めることは珍しくない．我々はこの問題を回避するため，結果に影響を与えないと思われる命題や事象の観測条件の計算を回避する経験則の導入を考えたこともある[8]．

ところが，はじめにも述べたように，ある変化が信念推定の結果に影響するかどうかは，領域知識やそれまでの経緯に依存するため，事象だけを見て判断すると間違える可能性がある．ネストした信念を推定する場合には，推定対象の命題の観測条件に含まれる各命題への影響や，さらにその各命題の観測条件に含まれる各命題への事象の影響なども考えなければならない．

たとえば，この家の書斎とダイニングの間の壁の透明度が，このデスクトップパソコンで管理されていて，Aをタイプすると透明になり，Zをタイプすると不透明になるとしたらどうだろうか．この場合は，壁が透明になった状態まで遡ったところで，ノートパソコンが食卓の上にまだあるかどうか観測できる．また，この家では，Rをタイプするとロボットが食卓の上を片づけるのかもしれない．

つまり，命題どうしが領域知識を通じて依存しあっているため，ある事象が結果に影響するかどうかは，あらかじめ決まっているというよりは，そのときの状況にかなり依存しているのである．安易に関係あるかどうかを経験則で判断すると，間違った答を導くことも十分ありうる．そこで経験則に依存しない一般的な高速化技法が望まれる．

そこで2章では，命題がどの時点で変化したかを簡潔に表すデータ構造を用い，計算結果に関係する命題が変化した場合にだけ観測条件の再計算を行う前進的アルゴリズムを提案する．次に3章では，前進的アルゴリズムの実装を用いて，処理時間がどのような傾向になるか，遡行的アルゴリズムと対比しながら説明する．

2. 前進的アルゴリズムの提案

論文11)では，アルゴリズムを命題表現で記述し，実装時に決定的述語という関数表現を導入したが，本論文で提案するアルゴリズムでは，アルゴリズムの設計段階から関数表現を考慮する．つまり，$f(a)$ という関数とその値 b の対 $f(a) = b$ を命題変数と考え，$f(a)$ が複数の値を同時にとれないという制約を命題間の競合関係と対応づけることにより，領域知識を簡単に書けるようにする．そして，一貫性制約をこの関

数の値の一意性（関数従属性[17]）で表せるものだけに限り，それ以外の制約は考えない．この仮定により一貫性制約の記述が不要になる．そこでまず，データや領域知識をすべて関数表現に翻訳する方法を示す．

2.1 領域知識の関数表現

本章で提案するアルゴリズムでは，関数の値の変化を記録した「ログ（log）」というデータ構造を操作することによって，信念推定を行う．また，ここでいう関数とは，関数記号に具体的に定数引数が与えられたもの，たとえば $f(c_1, c_2, c_3)$ のことである．

ログは時刻 t とそのときの関数の値 v の対 $t@v$ のリストであり，$[0@a, 5@b, 8@c, 12@a]$ のように表す．関数はログ自体には含めず，$f(a) - [0@b, 2@c]$ のように外付けする．$[3@a, 5@a, 6@a, 7@b]$ のように同じ値が続くときは，最初のものだけ残してあとは削除した $[3@a, 7@b]$ という表現を標準形とする．

また，ある時点で特定の値を信じていないときは `null` という定数を仮の値にする．$[1@\mathtt{null}, 3@b]$ のように先頭の値が `null` の場合は，$[3@b]$ と表しても同じ意味と考えられるので，`null` がない方を標準形とする．以下の説明はすべてこの標準形を仮定している．

これに合わせて，システム自身の観測データ $O^+(t)\cdot O^-(t)$ を関数表現にする．たとえば，$(f(a) = b) \in O^+(0)$，$(f(a) = c) \in O^+(3)$，$(f(a) = c) \in O^-(5)$，$(f(a) = d) \in O^+(8)$ という関数 $f(a)$ の観測データが与えられたとしよう．時刻 3 では $f(a) = c$ が観測されたため，時刻 4 でも $f(a) = c$ と考えるのがもっともらしいが，時刻 5 では $\neg(f(a) = c)$ が観測されているため，もはや値 $f(a) = c$ を信じることはできない．しかし，かわりに信じられる値もないため，結局 $[0@b, 3@c, 5@\mathtt{null}, 8@d]$ というログに変換するのが妥当となる．このログ表現で表したときのシステム自身の信念における各関数 F のログを $\mathtt{sysbel}(F)$ で表す．

ただし，ログ表現は $O^-(t)$ によるデータを完全には表せない．たとえば，$(f(a) = b) \in O^-(0)$ という否定的情報だけをログで表現することはできない．しかし，遡行的信念アルゴリズムの背景理論[13]では，真偽不明の命題が偽の命題に準じた扱いを受けるため，観測条件が NOT を含まない単調関数である限り，これらの情報を表せなくても，推定結果には影響しないはずである．著者は，$O^+(t)\cdot O^-(t)$ による表現ではデータ量が膨大になるため，ログ表現を採用することによって得られるデータ量の圧縮と処理の負担の軽減の方が優先されてよいと考える．

また，$(a_1, \ldots, a_k)$ の信念で関数 $f(a)$ の値がどう変遷したかを表すログを計算する場合には，その外界である $(a_1, \ldots, a_{k-1})$ の信念における関数 $f(a)$ のログが重要な役割を果す．そこで，後者のログを前者のログの「ベースログ」と呼ぶことにする．

```
init(room(a), kitchen).
init(room(b), study).
init(room(c), living).
init(room(diningtable), dining).
init(room(glasstable), living).
init(room(desk), study).
init(support(tray), diningtable).
init(support(cup), diningtable).

effect(moveto(A, R), room(A), R).
effect(grasp(A, X), support(X), A).
effect(put(_, X, P), support(X), P).
```

図 1 初期状態と事後条件の関数表現
Fig. 1 Functional representation of an initial state and postconditions.

初期状態も関数名とその値の対で表す．たとえばエージェント a が台所にいることを $\mathtt{in}(a, \mathtt{kitchen})$ と表すかわりに $\mathtt{room}(a) = \mathtt{kitchen}$ と表す．また物体 `mug` がテーブル `tab` の上にあることを，$\mathtt{on}(\mathtt{mug}, \mathtt{tab})$ と表すかわりに $\mathtt{support}(\mathtt{mug}) = \mathtt{tab}$ と表す．**図 1** に DEC-10 Prolog の構文を用いた領域知識の例を示す．

事象の事後条件も関数表現を用いる．たとえば，事象 $\mathtt{moveto}(A, R)$ によって $\mathtt{room}(A) = R$ が生じることを $\mathtt{effect}(\mathtt{moveto}(A, R), \mathtt{room}(A), R)$ と表す．このような方法で表せるのは，事象 e によって真になる命題の集合 $\mathsf{Post}^+(e)$ だけである．偽になる命題の集合 $\mathsf{Post}^-(e)$ は表現できないが，著者の経験によれば，$\mathsf{Post}^-(e)$ が表せなくてもとくに困ることはない．たいていの場合は，$q \in \mathsf{Post}^-(e)$ ならば，$q' \in \mathsf{Post}^+(e) \cap n(q)$ となる新しい命題変数 q' を導入することで言い換えられるからである．もっとも，このように書き換えると問題が生じる場合があるかもしれない．これに関する検討は今後の課題である．

命題の観測条件は $\mathtt{obs\_flu}(F, Y, O, Cond)$ で与える．第 1 引数 F は対象の関数，第 2 引数 Y はその値である．つまり，$F = Y$ という命題が観測対象である．第 3 引数 O は観測者で，第 4 引数 $Cond$ が命題 $F = Y$ の観測者 O に対する観測条件である．この観測条件も関数表現で与える．

ただし実装の都合上，$h(f(a), g(b, c)) = d$ という観測条件であれば，$[f(a) = Y, g(b, c) = Z, h(Y, Z) = d]$ というように条件を計算の順に分解して表す

```
obs_flu(room(_), R, O, [room(O) = R]).
obs_flu(support(_), S, O,
    [room(O) = R, room(S) = R]).
obs_flu(support(_), S, O,
    [room(O) = R, support(S) = T, room(T) = R]).

obs_act(moveto(A, _), O,
    [room(A) = R, room(O) = R]).
obs_act(moveto(_, R), O, [room(O) = R]).
obs_act(grasp(A, _), O,
    [room(A) = R, room(O) = R]).
obs_act(put(A, _, _), O,
    [room(A) = R, room(O) = R]).
```

図 2 観測条件の関数表現
Fig. 2 Functional representation of observability conditions.

ことにする．つまり $Cond$ は，関数と値の対のリスト $[f_1(x_1) = y_1, \ldots, f_h(x_h) = y_h]$ で与え，これは $(f_1(x_1) = y_1) \wedge \cdots \wedge (f_h(x_h) = y_h)$ の意味である．そして引数 $x_i\,(i > 1)$ や値 y_i は，観測条件の宣言では変数を使ってもよいが，計算時に $f(x_i) = y_i$ に達したときには，すでに引数 x_i が定まっていなければならない．なお，これらの関数表記による条件の記述において，`null` を値として含む条件，つまり，特定の関数に値がないという条件を許さないことにする．値が `null` であるという条件を書きたくなることもありうるが，これは非単調推論ともからんでおり，今後の課題とする．

図 2 の `obs_flu` は，あるエージェントが部屋 R にいることを観測できるためには，観測者 O が部屋 R にいればよく，ある物体が S に支えられていることを観測できるためには，観測者 O が S のある部屋にいるか，あるいは S を支える物体 T のある部屋にいればよいことを表す．事象の観測条件 `obs_act` も同様に翻訳できる．

2.2 世界全体を見渡せるシステムの信念

システム自身の信念は，各関数 F のログ `sysbel`(F) の集合で表される．ロボットのように，自分の知らない世界があるシステムの場合には，これが前進的信念推定アルゴリズムの参照する入力データとなる．

しかし，計算機上の仮想世界などでは，システムがすべての変化を知っており，「観測」により得るデータが，初期状態に関する信念 `init` と事象系列 $e^1, \ldots, e^m$ の「影響」だけから完全に予想できることがある．このような場合には，`sysbel`(F) をこれらのデータから，**図 3** の関数 `effect_hist`(F) を用いて直接作ることができる．

```
function effect_hist(F) {
  if (init(F, V)がある) {L := [0 @ V];}
  else  {L := [];}
  for (T := 1; T ≤ m; T := T + 1) {
    if (effect(e^T, F, V) とマッチする
        ルールがある) {
        L の最後に T @ V を加える;}}
  return L;
}
```

図 3 初期状態と事後条件から関数のログを求める
Fig. 3 Getting a function's log from an initial state and postconditions.

これは単純な処理であるが，この考え方が他者の信念の変遷を計算する場合の基本となる．つまり，推定対象のエージェントがどの事象を観測できたか分かれば，そのエージェントの信念の変化がある程度うかがえる．これにそのエージェントが命題の「観測」によって見つける予想外の変化を適宜加えればよい．そこで 2.3 節では，事象の観測条件の計算方法を説明する．次に 2.4 節で命題の観測条件の計算を効率的に進めるために「失効時刻」という概念について説明する．そして 2.5 節で命題の観測条件の計算方法を説明する．さらに，命題の観測で問題となる否定的観測の処理について 2.6 節で説明する．そして最後に 2.7 節で，観測条件の計算結果を統合して他者の信念を推定する方法を説明する．

2.3 事象の観測条件の計算

以下では他者の信念を推定するために観測条件の計算方法を説明する．まず，事象の観測条件については，論文 11) の `obsAll` とほとんど同じように処理すればよい．つまり，エージェント添字列 $(a_1, \ldots, a_k)$ で表せるネストした信念を計算する場合には，まずシステム自身の信念で a_1 の観測条件が成立しているかどうかを調べ，次に (a_1) の信念において a_2 の観測条件が成立しているかどうかを調べ，さらに (a_1, a_2) の信念において a_3 の観測条件が成立しているかどうかを調べ，という作業を繰り返して，最後に $(a_1, \ldots, a_{k-1})$ で表されるネストした信念において a_k の観測条件が成立しているかどうかを調べて，以上のすべての観測条件が成立する場合に限って，a_k が e を観測できたと判断する．なぜなら，a_i が e に気づかないと，a_i が a_{i+1} の信念を推定する場合に e を考慮するとは考えにくいからである．

ただし，これまでとほとんど同じとはいっても，処理を効率化するため，データ構造を以下のように変更する．各事象がエージェント添字列の何番目のエージェントまで観測できたかを事象系列と同じ長さ (m) の整数値リストで表すことにしよう．このようなリストを「**事象観測リスト**」と呼び，以下では EN で表す．その i 番目の要素 $EN[i]$ が e^i の観測可能性に対応する．$EN[i]=d$ は次のような意味である．

- $d=0$：e^i の観測条件はまだ調べられていない．
- $d>0$：e^i は $(a_1,\ldots,a_d)$ まで観測条件が成立していることが確認された．
- $d<0$：e^i は $a1,...,a_{-d-1}$ まで観測条件が成立しているが，a_{-d} の観測条件は成立していないことが確認された．

たとえば $[0,-1,1,-3,0]$ というリストは，e^1,e^5 の観測条件が未計算で，e^2 が a_1 に観測できず，e^3 が a_1 に観測でき，e^4 が a_3 に観測できなかったことを表す．事象の観測条件は，その事象が推定対象の関数に影響しそうな場合にしか計算されないので，EN にはあちこちに 0 が含まれているのが普通である．

外界 $(a_1,\ldots,a_{k-1})$ での $\mathrm{ob}[a_k,e^T]$ の真偽を計算するには，**図 4** にしたがって `check_oe`$([a_{k-1},\ldots,a_1],a_k,e^T,k,T)$（oe は observability of event の略）を計算すればよい．図の前半では，すでにそれが計算ずみかどうかチェックしている．後半で使われている `head`(L) はリスト L の最初の要素を返す関数，`tail`(L) はリスト L の最初の要素を除いたリストを返す関数である．また `holds` は論文 11) の `obsOne` に相当するもので，事象の観測条件を，外界での真偽に従って計算する．このさいに論文 11) では信念推定関数 `belief` を再帰的に呼び出していたが，ここでもそれに相当する関数 `nbel` を 2.7 節で定義する．`holds` はこの `nbel` の結果を論理的に組み合わせて観測条件全体の真偽を求める．

```
function check_oe(Agents, A, E, K, T) {
  if (K = 0) {return 真;}
  if (K ≤ EN[T] or EN[T] < −K) {
    return 真;}
  if (−K ≤ EN[T] < 0) {return 偽;}
  if (check_oe(tail(Agents),head(Agents),
       K − 1, T) が偽) {return 偽;}
  foreach Cond s.t. obs_act(E, A, Cond) {
    if (holds(Cond, A, T − 1, Agents)) {
      EN[T] := K; return 真;}
  }
  EN[T] := −K;
  return 偽;
}
```

図 4 ある事象の観測条件のチェック
Fig. 4 Checking observability conditions for an event.

2.4 特定の時刻における関数の値と失効時刻

前述のとおり，事象の観測条件の処理はほとんど従来どおりであったが，命題の観測条件の処理は以下のようにやや複雑になる．事象は一瞬で終了するため，その観測条件の計算結果が次の時刻に使われることはほとんどないが，命題には持続性があるので，観測条件の計算結果がその後も有効である可能性が高く，その結果が有効な限り再計算を回避したいためである．

そこで，ある時刻について観測条件を計算する場合，得られる真偽値がその後のどの時点まで持続するかをいっしょに計算する．そのためには，観測条件に含まれる各関数の値が指定された時刻で何か，そしてその値がその後のいつの時点に次の値になるかを調べなければならない．

すでに述べたように，ログは各時刻での値を明示的に記録したものではないため，特定の時刻における関数の値が必要になれば，ログから求める必要がある．命題の持続性から，関数の値は持続すると考えてよいので，指定された時刻 T までで最新の値をログから探せば，それが T での値と考えられる．T までにまだ値が設定されていなければ `null` を値とする．

そして，関数値や命題の真偽値が次の値になる時刻をその値の「**失効時刻（expiration time）**」と呼び，以下では X で表す．次の値がない場合は命題の持続性により，その値が今後ずっと続くと考えられるので，無限大を表す定数 `inf` を失効時刻として返すことにする．このため，失効時刻は $\{0,1,2,\ldots\}\cup\{\mathtt{inf}\}$ の値域を持つ．

たとえば $f(a)=b$ という命題の場合，まず $f(a)=b$ が指定された時刻 T で真か偽かを $f(a)$ のログから求める．もし真なら次に真でなくなる時刻，偽なら次に偽でなくなる時刻がその真偽の失効時刻である．たとえば $f(a)$ のログが $[0\,@\,c,3\,@\,d,5\,@\,b]$ であるとき，命題 $f(a)=b$ の時刻 2 における値は偽で，その失効時刻は 5 である．$3\,@\,d$ は $f(a)=b$ の真偽を変えない．

また，$f(a)=Y$ のように，観測条件には具体的な値が与えられていない場合もあるが，この場合は，時刻 T での $f(a)$ の値をそのログから求めて Y に代入する．そして $f(a)$ が次の値になる時刻を失効時刻とする．

こうして，関数 F のログ L から，時刻 T における

命題 $F = Y$ の真偽 B と B の失効時刻 X を求めたり，Y が変数の場合には，Y の具体的な値と Y の失効時刻 X を求める関数を容易に定義できる．そこで，これを「$(Y, B, X) :=$ `value_at`$(L, T, Y);$」という代入文で表す．もちろん Y が変数の場合，B は真とする．

2.5 命題の観測条件の計算

命題 $f(a) = b$ に関する A さんの信念を計算するためには，その命題 $f(a) = b$ の真偽と，その観測条件 `ob`$[A, (f(a) = b)]$ の真偽を計算しなければならない．したがって，その観測条件に含まれる各関数の値が必要になる．

たとえば $f(a) = b$ の観測条件が $h(g(a)) = b$ の場合，$g(a)$ の値によって h の引数が変わる．もし $g(a) = c$ なら，$h(c)$ の値が命題 $f(a) = b$ に対する信念に影響を与える．しかし，$g(a) = d$ なら，$h(c)$ の値は A さんの信念に影響を与えない．このことからも，**ある信念に影響を与える関数をあらかじめ列挙しておくことは困難であること**が分かる．

そこで以下の計算では，信念の計算を進めながら，信念に影響を与える関数のログの集合も作成する．得られる集合を「**関連ログ集合**」と呼ぶ．$f(a) = b$ の観測条件が $h(g(a)) = b$ であり，外界では $g(a)$ のログが $[3 @ c, 6 @ d]$，$h(c)$ のログが $[2 @ b, 8 @ e]$，$h(d)$ のログが $[2 @ c]$ ならば，関連ログ集合に $[g(a) - [3 @ c, 6 @ d], h(c) - [2 @ b, 8 @ e], h(d) - [2 @ c]]$ のように登録する．

関連ログ集合は，エージェント添字列の各プレフィクス $(), (a_1), (a_1, a_2), \ldots, (a_1, \ldots, a_k)$ のそれぞれに対応して存在する．この $k+1$ 個の関連ログ集合を，それぞれ $LN[0], LN[1], LN[2], \ldots, LN[k]$ と表す．そして $LN[h]$ に関数 F のログ L を追加する関数を `lput`(F, L, h)，逆に $LN[h]$ から関数 F のログを得る関数を `lget`(F, h) とし，`lget` は対応するログが存在しない場合に `null` を返すものとする．

観測条件の計算に必要な関数のログが得られれば，条件中の各命題 $F = Y$ を 2.4 節の `value_at` に適用できるので，次の課題はそれを統合して観測条件全体の真偽と失効時刻を求めることである．そこで論理演算にともなう失効時刻の処理が問題となる．

失効時刻が与えられている複数の命題に対して AND/OR 演算を行った場合の結果の失効時刻は簡単に求められる（**図 5**）．

まず，$p \wedge q$ の場合は次のように考えられる．

- p, q がともに真の場合，$p \wedge q$ も真である．この場合，p, q のどちらか一方でも失効して真でなくなってしまえば，$p \wedge q$ も失効する．つまり，$p \wedge q$ の失効時刻は，p, q の失効時刻の小さい方に一致する．
- p, q がともに偽の場合，$p \wedge q$ は偽である．この場合，p, q のどちらか一方が失効して偽でなくなったとしても，もう一方が偽のうちは，$p \wedge q$ が偽のままである．つまり，$p \wedge q$ の失効時刻は，p, q の失効時刻の大きい方に一致する．
- p が真で q が偽の場合，$p \wedge q$ は偽である．この場合，p が失効して真でなくなったとしても，q が偽である限り $p \wedge q$ が偽であることにはかわりない．一方，q が失効して偽でなくなったとしたら，$p \wedge q$ が真になる可能性がある．したがって，$p \wedge q$ の失効時刻は q の失効時刻と一致する．

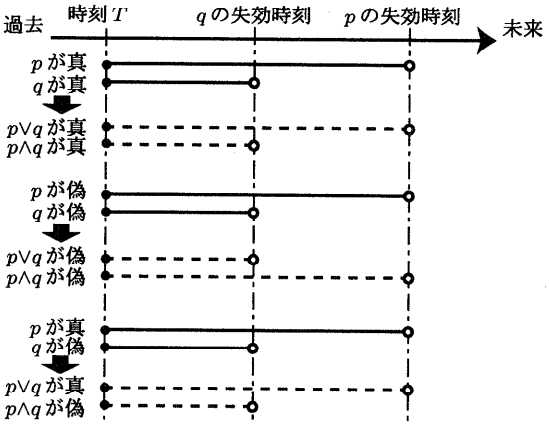


図 5 論理演算にともなう失効時刻の計算
Fig. 5 Computation of expiration times for logical operations.

ただし，$(f(a) = Y) \wedge (g(Y) = b)$ や $(f(a) = Y) \wedge (g(b) = Y)$ のように，命題間で変数が共有されている場合は別である．たとえば前者の場合，$f(a)$ の値 Y が c から d に変化したとすると，最初は $(f(a) = c) \wedge (g(c) = b)$ の真偽が問題となるが，d に変化した時点で，今度は $(f(a) = d) \wedge (g(d) = b)$ の真偽が問題となる．考えるべき命題が変わるため，単純に上記の失効時刻の計算を適用できない．そこで厳密に関数間の依存関係を調べ，どのような依存関係の場合に失効時刻がどのようになるか調べることも可能であるが，ここではもっと単純に，各命題の失効時刻の最小値を全体の失効時刻とする．

なお，関数の値が不明を表す `null` になった場合は，ただちにその連言全体の計算を終了する．たとえば，A と B が同じ部屋にいるという条件 $(\texttt{room}(A) = R) \wedge (\texttt{room}(B) = R)$ が $R =$ `null` で成立したとしても，これは単に両方の居場所が分からない，というだけで

あり，A が B を観測できると考えるべきではないからである．この場合は観測条件が偽と考え，その「偽」の失効時刻は，それまでの連言要素の失効時刻の最小値とする．というのも，その部分が変わらない限り，最後の関数の値が null 以外にならないからである．

一方，$p \vee q$ の失効時刻は以下のように考えられる．

- p, q がともに真の場合，$p \vee q$ も真である．この場合，p, q のどちらか一方が真であり続ける限り，$p \vee q$ は真のままである．したがって，$p \vee q$ の失効時刻は p, q の失効時刻の大きい方に一致する．
- p, q がともに偽の場合，$p \vee q$ も偽である．そして，p, q のどちらか一方が偽でなくなると，$p \vee q$ は真になる．したがって，$p \vee q$ の失効時刻は p, q の失効時刻の小さい方に一致する．
- p が真で q が偽の場合，$p \vee q$ は真である．この場合，p が真でなくなってしまうと，$p \vee q$ は偽になるが，q が偽でなくなっても p が真であり続ける限り，$p \vee q$ は真のままである．したがって，$p \vee q$ の失効時刻は p の失効時刻に一致する．

2.1 節で説明した観測条件の表現から分かるように，OR をまたがった変数の共有はないので，AND のときのような失効時刻の計算の問題は生じない．

以上の考察に従って，命題 $F = Y$ の a_k に対する観測条件の時刻 T における真偽 O と失効時刻 X の対 (O, X) を計算する手続きを考え，以下では「(O, X) := check_of$(F, Y, a_k, T, [a_{k-1}, \ldots, a_1])$;」で表す．

2.6 否定的観測の処理

以上の議論により，観測条件の計算ができるようになったので，これらを組み合わせれば，信念の推定がすぐにできそうである．しかし，論文 11) で議論した局所閉世界情報[6] による否定の問題が残っている．これも遡行的アルゴリズムの場合よりやや複雑な処理になる．

この否定の問題とは，一般には p が観測されないからといって，ただちに $\neg p$ であるとは断言できないが，多くの人が「この部屋には象がいる」というような命題をただちに否定できるように，局所的には特定の命題を否定できることがあるため，これをどう扱えばよいかという問題である．

我々は命題変数 p に対して，$p \wedge \mathbf{ob}[b, p]$ が成り立てば b さんが p を信じ，$\neg p \wedge \mathbf{ob}[b, p]$ が成り立てば b さんが $\neg p$ を信じる，と考えている[11]．これによれば，「この部屋に象がいる」という命題を b さんが観測できる状態で実際には象がいなければ，象がいないことが b さんに分かる．この否定はデフォルトによる否定というような弱いものではなく，たとえ b がそれまで p を信じていたとしても，$\neg p \wedge \mathbf{ob}[b, p]$ が成り立てば $\neg p$ を信じることになるという**強い否定**である．

たとえば，Bob ($= b$) が Alice ($= a$) に会おうと思ってリビングに行ってみたら（つまり $B_b(\mathtt{room}(a) = \mathtt{living})$），Alice はそこにいなかった（$\neg(\mathtt{room}(a) = \mathtt{living}) \wedge \mathbf{ob}[b, (\mathtt{room}(a) = \mathtt{living})]$）という状況を考えてみよう．そのとき Alice が台所にいて，リビングにいる Bob から Alice が見えれば（$(\mathtt{room}(a) = \mathtt{kitchen}) \wedge \mathbf{ob}[b, (\mathtt{room}(a) = \mathtt{kitchen})]$），Alice が台所にいることが Bob に分かる（$B_b(\mathtt{room}(a) = \mathtt{kitchen})$）．しかし，リビングから台所が見えないとすると，Bob には Alice がどこにいるのか分からない．つまり，それまでの値 living が否定されたが，それにとってかわる新しい値が見つからない，という状況が発生する．これが局所閉世界情報による否定の例である．

2.1 節でも述べたように，ログ表現では $f(a) = c$ のような肯定的な命題の表現は可能であるが，$\neg(f(a) = c)$ のような否定は表現できない．そのため，c にかわる新しい値は分からないが，それまで信じていた値 c でないことは確実，という場合には，null をログに入れて，それまでの信念 $f(a) = c$ を失効させる．

ここでは，以下のような順序で関数 $f(a)$ の b さんによる観測の処理を進める．まず，着目した時刻 T において，ベースログ（2.1 節）で $f(a) = y_0$ なら，まず $f(a) = y_0$ が b さんに観測できるかチェックし（**肯定的観測**），できないときに限り，b さんが直前まで信じていた $f(a) = y_1$ が局所閉世界情報により否定されないかチェックする（**否定的観測**）．そして肯定的観測ができた場合は T @ y_0 を，否定的観測ができた場合は T @ null をログに追加する．

このように，否定的観測を処理するには，否定されるべき値，つまりエージェント A の信念における最新の $f(a)$ の値 y_1 が必要である．したがって，この手法では，**過去に遡っていく遡行的計算より時間の経過に従う前進的計算の方が実装しやすい．**

観測条件を次に計算すべき時刻（観測条件の失効時刻）も計算しなければならないので，結局以下のように処理する．

- $f(a) = y_0$ の肯定的観測ができれば，ログの末尾に T@y_0 を追加する．そして，外界での $f(a) = y_0$ の失効時刻 X_1 に再計算する．なぜなら，一度肯定的観測ができてしまえば，値が y_0 以外になら

ない限り，y_0 が観測できてもできなくても，信念は変化せず，その間に観測条件を再計算することは，信念を求めるという観点からは無駄だからである．

- $f(a)=y_0$ の肯定的観測はできないが，直前の事象 e^T が観測でき，その事後条件により $f(a)=y_0$ が分かる場合：$\mathtt{effect}(e^T, f(a), y_0)$ であればよく，ログの末尾に T @ y_0 を追加する．次に観測条件を計算すべきは，やはり $f(a)$ が外界で次の値になる時刻 X_1 である．
- $f(a)$ に対して信じている値が今のところない場合：否定的観測について考える必要がなく，ログもそのままにする．$f(a)=y_0$ の観測条件の失効時刻 X_0 と外界での $f(a)=y_0$ の失効時刻 X_1 の早い方 X_2 に再計算する．
- それまで信じていた $f(a)=y_1$ の観測条件が真で，$\neg(f(a)=y_1)$ だけが分かる場合：ログの末尾に T @ null を追加する．その後，肯定的観測ができる可能性があるので，X_2 に再計算する．
- 上記に該当せず，$f(a)=y_0$ も $\neg(f(a)=y_1)$ も得られない場合：ログは書き換えない．その後，肯定的観測か否定的観測ができる可能性があるので，X_2 と $f(a)=y_1$ の観測条件の失効時刻 X_3 のうち，早い方の時刻に再計算を行う．

これらの考察により，$(a_1, \ldots, a_k)$ の信念における関数 F のログ L は，**図6** の add_of（of は observability of fluent の略）を利用するたびに，順に要素が確定する．図中の $\mathtt{length}(L)$ はリスト L の要素の数を返す関数である．add_of を時刻 T に関して呼び出すと，その時刻に関して加えるべき情報 ADD が get_of で計算されて L の末尾に加えられる．

add_of を利用するには，関数 F のベースログにおける時刻 T での値 Y_0，ベースログの T より後の部分のコピー BL，それまでの $(a_1, \ldots, a_k)$ の信念における F の最新の値 Y_1 が入力として必要であり，$\mathtt{add\_of}(F, Y_0, T, a_k, [a_{k-1}, \ldots, a_1], Y_1, L, BL)$ の形で呼び出される．なお add_of の最後では，時刻 T に肯定的観測ができなかった場合に X @ Y_0 を BL の先頭に加えているが，これは肯定的観測が後でできないか調べるためである．そして add_of は Y_1，BL，L を更新する．

2.7 ネストした信念の推定

これまで説明した処理を組み合わせると，ネストした信念の推定が可能になる．関数 F のログに関する $(a_1, \ldots, a_k)$ の信念は，**図7** に従って $\mathtt{nbel}([a_k, a_{k-1}, \ldots, a_1], F)$ を計算すれば得られる．

最初にこの信念の変遷がすでに計算されていないか

```
function add_of(F, Y0, T, A, Agents,
    Y1, BL, L) {
    if (Y0 = null) {return L;}
    (X, ADD) :=
    get_of(F, Y0, T, A, Agents, Y1, BL);
    if (ADD ≠ nothing) {
        Y1 := ADDの持つ値;
        Lの末尾にADDを追加;}
    if ((ADD = nothing or ADD = T @ null)
        and X ≠ inf and
        BLの先頭にX @ ...がない) {
        BLの先頭にX @ Y0を追加;}
    return (Y1, BL, L);
}
function get_of(F, Y0, T, A, Agents, Y1, BL) {
    // F = Y0の観測条件の真偽O0と
    // 失効時刻X0を調べる.
    (O0, X0) := check_of(F, Y0, A, T, Agents);
    K := length(Agents);
    (Y0, O1, X1) := value_at(BL, Y0, T);
    // 肯定的観測ができたか.
    if (O0 and O1) {return (X1, T @ Y);}
    // 事象の事後条件からF = Y0がわかるか.
    if (T > 0 and effect(e^T, F, Y0) and
        check_oe(Agents, A, E, K + 1, T)) {
        return (X1, T @ Y0)}
    X2 := min(X0, X1);
    if (Y1 = null) {return (X2, nothing);}
    // 否定的観測ができるか調べる.
    (O3, X3) := check_of(F, Y1, A, T, Agents);
    if (O3が真) {return (X2, T @ null);}
    return (min(X2, X3), nothing);
}
```

図6 ある時刻にあるエージェントが関数値をどう観測したかの計算
Fig. 6 Computing how one observed a function value at a certain time.

を lget で調べ，まだ計算されていなければ，それが自分の信念かどうか調べ，自分の信念なら sysbel から持ってくる．自分の信念でなければ，先に F のベースログ BL を nbel の再帰呼び出しで計算して，その各要素 T @ Y を前から順に add_of に与えて，それが a_k に観測できるかどうか計算するのである．こうして得られた結果が lput で関連ログ集合 LN に記録される．なお，$EN[i]$ の初期値はすべて 0，$LN[i]$ の初期値はすべて空リスト [] である．

```
function nbel(Agents, F) {
  K := length(Agents);
  L := lget(F, K);
  if (L ≠ null) {return L;}
  if (K = 0) {L := sysbel(F);}
  else {
    L := []; Y1 := null; A := head(Agents);
    Ext := tail(Agents); BL := nbel(Ext, F);
    while (BL ≠ []) {T @ Y := head(BL);
      (Y1, BL, L) :=
      add_of(F, Y, T, A, Ext, Y1, tail(BL), L);}
  }
  lput(F, L, K); return L;
}
```

図 7 ネストした場合の信念の変遷の計算
Fig. 7 How to get a belief history for nested cases.

3. 実験結果

前章で提案した前進的アルゴリズムを Prolog で実装した．本プログラムは SICStus Prolog，Quintus Prolog，BinProlog などの DEC-10 Prolog に準拠した多くの Prolog で実行可能である．以下の実験では，図 1 と図 2 に示した領域知識を仮定する．そして，複数のエージェントがキッチンやリビングなどの複数の部屋からなる家の内外を動きまわっているという状況を与える．

ここで用いた領域知識によると，A さんの移動や居場所の観測条件に含まれるのは A さんと観測者 O の居場所だけであり，エージェントの数や身長，部屋の数や形や面積，隣接関係，照明の有無，家具の配置，窓ガラスの光学的性質，ドアの数や位置や開閉，日時などは観測条件に含まれていない．したがって，これらについてはとくに情報を与えていない．論文 11) にもう少し複雑な観測条件の例があるので参考にされたい．

具体的に与える事象系列は以下で実験結果とともに説明する．エージェントの移動をとくに取り上げる理由は，エージェントの居場所がそのエージェントの観測を大きく左右しており，観測可能性に基づく信念推定の基本的かつ重要な課題だからである．

以下の実験では，まず否定的観測が正しく動作していることを確認し，次に処理速度を遡行的アルゴリズムと比較する．なお，推論システムが全事象を知っていると仮定し，sysbel(F) = effect_hist(F) とした．

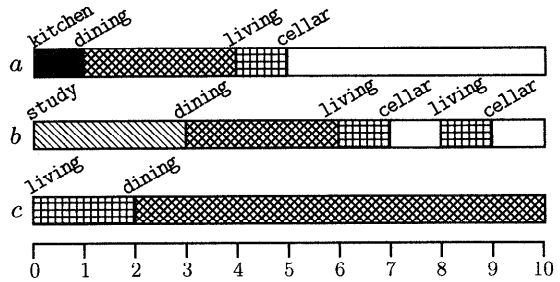


図 8 否定的観測のチェックのための変遷の例
Fig. 8 An example history for checking negative observation.

3.1 否定的観測に関する結果

2.1 節で例として用いた領域知識をそのまま利用して，事象系列が次の場合を考えてみよう．

[moveto(a, dining), moveto(c, dining), moveto(b, dining), moveto(a, living), moveto(a, cellar), moveto(b, living), moveto(b, cellar), moveto(b, living), moveto(b, cellar)]

これは，最初に a，b，c の 3 人がダイニングに集まったあと，a がリビングに移った．a はさらにセラーに移るが，b はすこし遅れてリビングに移ったので，すでに a はリビングにいない．ここで b は a を見失うが，b がセラーに行ってみると，a がそこにいた．b がリビングに一度戻ったあと，セラーに再度行ってみると，まだ a はセラーにいた，という状況である．この事象系列において，各エージェントが各時刻にどこにいたのかを**図 8** に示す．横軸は時刻 t である．たとえば初期状態 $t = 0$ では，図 1 に示したとおり，a がキッチンに，b が書斎に，c がリビングにいる．

このケースに対して room(a) のログに関する b の信念を推定してみると，次のようなログが得られ，6@null は b が a を見失っていることを表している．

[3 @ dining, 4 @ living, 6 @ null, 7 @ cellar]

もちろん，リビングからはセラーにしか行けないのであれば，b さんは 6 の時点ですでに room(a) = cellar に気付いているかもしれない．このような歴史的整合性を考慮した推論は，一般に計算量を爆発させるので，これまで我々はあえて取り入れなかったが，本アルゴリズムが関数のログを出力することを利用すると，場合によっては観測できなかった部分を補間し，整合的な歴史を作ることが可能になるであろう．このような推論の実現も今後の重要な課題の 1 つである．

また $t = 9$ で b がリビングからセラーに戻って a の居場所を再確認しても，a がセラーにいるという b の信念は変化しないはずである．このログには $t = 7$ よ

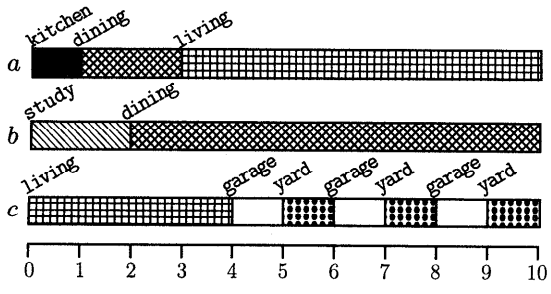


図 9 実行時間の測定のための変遷の例
Fig. 9 An example history for measurement of execution time.

りあとにデータがなく，信念が変化していないことが確認できる．

同様に room(b) のログに関する a の信念を推定してみると，次のようになり，ダイニングにいると思っていた b が $t=7$ に突然セラーに現れたことが分かる．また，b が $t=8$ でリビングにいったん戻っていることも正しく a に認識されている．

[3 @ dining, 7 @ cellar, 8 @ living, 9 @ cellar]

一方，room(a) についての b の信念についての a の信念，つまり (a,b) の信念の変遷を調べると次のようになり，a は，自分がどのように移動しているか b にだいたい把握されていることを認識していることが分かる．

[3 @ dining, 4 @ living, 7 @ cellar]

なお，a は b がリビングに移ったことを知らなかったため，(b) にはあった 6 @ null が (a,b) にはない．

3.2 実行時間の測定

前進的アルゴリズムが遡行的アルゴリズムに比べてとくに効果を発揮することが期待される極端な例として，3 つの動作 moveto(a, dining), moveto(b, dining), moveto(a, living) の後に 2 つの動作 moveto(c, garage) と moveto(c, yard) が交互に N 回繰り返されている長さ $2N+3$ の事象系列を考え，room(a) の値に関する信念の変化を推定してみよう．先ほどの例と同様に，各エージェントの居場所の変化を図 9 に示す．

room(a) の値に関する b の信念の変化を推定すると，N に関係なく [2 @ dining, 3 @ living] が得られる．2 @ dining は b が a のいる dining に移動したことによって room(a) = dining を発見したことによる．3 @ living は b が moveoto(a, living) を見かけたことによる．関数 room(a) 自身も，room(a) の b に対する観測条件に含まれている関数 room(b) も，このあとまったく変化していない．

したがって，b の room(a) に関する信念を計算するときは，後ろの $2N$ の事象を無視してよい．ところが遡行的アルゴリズムは，この $2N$ の事象を 1 つずつ遡って毎回観測条件を計算しなおす．これに対して前進的アルゴリズムは，ob[b, room(a)=...] が room(c) に依存しないため，room(c) が変化しても命題の観測条件を再計算しない．

一方，room(a) に関する c の信念の変遷を推定すると，N によらず [3 @ living] だけが得られる．これは，a が living にやってきたことは c に観測できたが，それ以前の変化もそれ以降の変化も c に観測できなかったためである．ところが，room(a) の c に対する観測条件に含まれている room(c) は，room(b) と違い，その後も変化し続けるため，本論文のアルゴリズムも，その変化のたびに観測条件の計算をしてしまう．(b,a,b) と (c,a,c) のようなネストした信念の推定では，この差がさらに増幅されると考えられる．

そこで実装したプログラムを Sun SPARCstation-20 Model 71, SuperSPARC II 75 MHz, 416 MB メモリ，SunOS 4.1.4 上の SICStus Prolog3#3 で実行し，(c,a,c) などのネストした信念において room(a) のログを求めるのに要した CPU 時間を statistics(runtime, T) で計測した．5 回計測して得られた平均 CPU 時間を，事象系列の長さ m で割った値を**表 1** に示す．たとえば，エージェント添字列 (c,a,c) の行の $m=603$ の列の場合，room(a) のログを求めるのに要した時間は 3.82 秒，3.70 秒，3.77 秒，3.68 秒，3.68 秒となり，平均 3.73 秒だったので，1 事象あたり $3.73/603 \times 1000 = 6.19$ ミリ秒要したことになる．表 1 を見ると，予想どおり (b,a,b) と (c,a,c) では計算時間に大きな差があることが分かる．

これ以外にも，(c,a) と (c,a,c) における room(a) の計算時間の間に大きな差があることに気づく．その理由を考えてみよう．まず (c,a,c) における room(a) のログを得るためには，(c,a) における room(a) と room(c) のログが必要である．(c,a) における room(a) の計算は短時間であることが分かっているので，room(c) の計算について考えてみよう．(c,a) での room(c) を計算するには，room(c) の値を変える各 moveto(c, ...) に対して，ob[a, moveto(c, ...)] を (c) において計算しなければならない（e^5 以降の moveto(c, ...) に対する結果はすべて偽になるので，$5 \le t \le 2N+3$ に対して $EN[t] = -2$ が得られる）．これに対して (c,a) で room(a) を計算する場合は，moveto(c, ...) が room(a) の値を変えないため，(c) における ob[a, moveto(c, ...)] の計算が不要になる．以上の理由によ

表 1 リスト表現を用いた実行時間（ミリ秒/事象）
Table 1 Execution time using lists (msec/event).

m	603	1203	1803	2403	3003
()	0.01	0.00	0.00	0.00	0.00
(b)	0.01	0.01	0.01	0.01	0.01
(b,a)	0.01	0.01	0.01	0.01	0.00
(b,a,b)	0.01	0.01	0.01	0.01	0.01
(b,a,b,a)	0.01	0.01	0.01	0.01	0.01
(b,a,b,a,b)	0.02	0.01	0.01	0.01	0.01
(c)	0.02	0.02	0.02	0.02	0.02
(c,a)	0.02	0.02	0.02	0.02	0.02
(c,a,c)	6.19	11.52	16.96	22.46	27.56
(c,a,c,a)	6.16	11.54	16.89	22.39	27.77
(c,a,c,a,c)	6.16	11.64	17.03	22.37	27.65

表 2 木表現を用いた実行時間（ミリ秒/事象）
Table 2 Execution time using trees (msec/event).

m	603	1203	1803	2403	3003
()	0.01	0.01	0.01	0.01	0.01
(b)	0.01	0.01	0.01	0.01	0.01
(b,a)	0.02	0.01	0.01	0.01	0.01
(b,a,b)	0.02	0.02	0.01	0.01	0.01
(b,a,b,a)	0.02	0.01	0.01	0.01	0.01
(b,a,b,a,b)	0.02	0.02	0.01	0.01	0.01
(c)	0.08	0.09	0.09	0.10	0.09
(c,a)	0.09	0.08	0.09	0.09	0.10
(c,a,c)	1.18	1.24	1.27	1.28	1.30
(c,a,c,a)	1.22	1.24	1.28	1.29	1.30
(c,a,c,a,c)	1.23	1.24	1.27	1.29	1.31

り，(c,a) では room(a) が room(c) より短時間で計算できると考えられる．実際に計算時間を測定すると，(c,a,c) における room(a) の計算時間のほとんどを (c,a) における room(c) の計算が占めていることが分かる．

次に (c,a,c,a) や (c,a,c,a,c) での room(a) のログを得るための時間が (c,a,c) とほとんど変わらない理由について考えてみよう．まず，(c,a,c,a) での room(a) は (c,a,c) での ob$[a,$ room$(a) = \ldots]$ と ob$[a,e^t]$ から計算できる．事象のほとんどを占める moveto$(c,\ldots)$ は room(a) に影響を与えないので，$t \leq 5$ では ob$[a,e^t]$ の計算が不要である．また ob$[a,$ room$(a) = \ldots]$ は room(a) のログから計算できるが，e^4 以後 a と c がお互いを観測できなくなるため (c,a,c) における room(a) のログは $t = 4$ までで終わりである．したがって，ob$[a,$ room$(a) = \ldots]$ にもほとんど時間がかからず，処理時間はあまり増えない．

同様に (c,a,c,a,c) での room(a) の計算を分解していくと，(c,a,c) における room(a) と room(c) の計算にたどりつく．どちらのログも $t = 4$ で終わりであり，(c,a,c,a) での ob$[c,$ room$(c) = \ldots]$ の計算なども $0 \leq t \leq 4$ の部分だけで終了する．また $EN[t] = -2$ が得られると，(c,a,c) での ob$[a,e^t]$ $(t \leq 5)$ や (c,a,c,a) での ob$[c,e^t]$ $(t \leq 5)$ が偽であることは，図 4 の $-K \leq EN[T] < 0$ の行からただちに分かるため，これらを実際に計算する必要がなく，やはり時間があまりかからない．したがって，(c,a,c,a,c) も (c,a,c) の処理時間（$2N$ 回 ob$[a,$ moveto$(c,\ldots)]$ を計算する必要がある）からほとんど増えないと考えられる．

表を見ると，(c,a,c,a) は $m = 603$ で 6.16 ミリ秒で，(c,a,c) の 6.19 ミリ秒よりわずかに小さい．ネストの深い信念の方が短いのはおかしいように思えるが，(c,a,c,a) の実際の測定値は 3.72 秒，3.68 秒，3.74 秒，3.73 秒，3.69 秒であり，(c,a,c) の測定値の分布とほとんど差がない．処理時間の差がごくわずかであれば，測定値のばらつきのために，数回の測定値の平均において逆転が確率的に発生することは不思議ではない．同様の逆転は何カ所かに見られるが，その差がいずれもきわめて小さいことから，同様の理由によるものと考えられる．

さらに表 1 をよく見ると，(c,a,c) などでは 1 事象あたりの時間が m にほぼ比例していることが分かる．したがって，長さ m の事象系列全体の実行時間は m^2 にほぼ比例していることが分かる．これは，room(c) のログや事象系列や事象観測リストの長さが m に比例し，必要な情報をアクセスするのに $O(m)$ の時間がかかるためと思われる．

そこで，これらのデータをバランスのよい二分木に変換して記録するようにプログラムを書き換えた．そのため Quintus Prolog の trees と assoc という 2 つの二分木のライブラリを用いた．ログの処理では指定された時刻 T にデータがあるとは限らず，T の前後の値を参照する必要があるため assoc が，事象観測リストは値を更新する必要があるため trees が適している．assoc の一部は文献 16)（p.148）にも掲載されている．これにより，アクセスにかかる時間を $O(\log m)$ に抑えられる．二分木を利用した場合の実行時間を**表 2** に示す．

表 1 と表 2 を比較すると，とくに時間のかかっていた (c,a,c) などについて，実行時間が下がっていることが分かる．たとえば (c,a,c) で $m = 3003$ の場合，木構造の採用により，約 20 倍の高速化が達成されている．(c,a) のように，もともとあまり時間のかかっていないケースでは，かえって時間が増えているが，これはリストと木構造の間の変換に要する時間と考えられる．

参考までに，遡行的アルゴリズムによって最新の値だけを推定させた場合の実行時間（3 回計測した平均値）を**表 3** に示す．こちらは java で実装したシステム[11]

表 3 遡行的アルゴリズムの実行時間（ミリ秒/事象）
Table 3 Execution time by the regressive algorithm (msec/event).

m	603	1203	1803	2403	3003
()	0.00	0.00	0.00	0.00	0.00
(b)	8.01	8.31	8.69	8.31	8.46
(b,a)	8.17	8.33	8.63	8.33	8.46
(b,a,b)	37.21	38.31	39.01	40.01	41.74
(b,a,b,a)	37.53	38.31	38.99	40.02	41.88
(b,a,b,a,b)	98.14	105.23	103.65	107.54	114.67
(c)	10.56	13.32	16.35	18.87	21.61
(c,a)	10.62	13.34	16.30	18.89	21.74
(c,a,c)	40.14	42.71	46.22	49.90	54.28
(c,a,c,a)	42.64	45.99	48.64	52.10	56.00
(c,a,c,a,c)	113.65	124.81	131.86	140.93	152.48

を SGI Indy XZ MIPS R4400 200 MHz, 64 MB メモリ, IRIX5.3 上の Java Development Kit 1.1 でコンパイル・実行した．前進的アルゴリズムとは実装言語もハードウェアも違ううえに，遡行的アルゴリズムは最新の値を求めることしかしないので，表 2 と表 3 の間での数値の直接の比較はあまり意味がない．

しかし，それぞれの表の中での実行時間の比は，プラットフォームよりもアルゴリズムの性質に大きく依存するものと考えられる．たとえば，(b) と (b,a,b,a,b) の実行時間を比較してみよう．表 2 ではほとんど両者に差がない．これは，$\mathtt{room}(a)$ の観測条件に含まれる $\mathtt{room}(a)$, $\mathtt{room}(b)$ のどちらもログがきわめて短く，$\mathtt{room}(c)$ が変化しても，これらのログの長さが変化しないためと考えられる．

これに対し表 3 では，(b,a,b,a,b) が (b) の 10 倍以上かかっている．これは，遡行的アルゴリズムが $\mathtt{room}(c)$ の変化を無視できず，すべての変化に対して観測条件を計算しているためと考えられる．

また，(b,a,b,a,b) と (c,a,c,a,c) を比較すると，表 2 では 100 倍前後の差がある．(b,a,b,a,b) の場合は $\mathtt{room}(a)$ の観測条件に $\mathtt{room}(c)$ が含まれていないが，(c,a,c,a,c) の場合は観測条件に $\mathtt{room}(c)$ が含まれている．つまり前進的アルゴリズムが関係ある変化とない変化を識別しているための差と考えられる．これに対して，表 3 では両者に大きな差がない．これもまた，遡行的アルゴリズムが関係ある変化かどうかにかかわりなく，観測条件の計算を行ってしまうためと考えられる．

以上の結果から，推定結果に関係のない変化の処理時間が大幅に削減されていることが裏づけられた．

4. おわりに

他者の信念を推定するという課題の重要性は，自然言語処理や知的教育システムなどの研究分野で幾度も強調されてきたが，実現方法として考えられてきたのは，様相論理の証明手続きである．これは計算量的に現実的でなく，論理的全知の問題をかかえ，また，現実の領域知識をどのように記述すべきかに関して明確な指針がない．経験則の研究[1]は行われているが，十分理論化はされていない．マルチエージェントの知識や信念の処理に関する研究もいくつか知られている[2),14)]が，時間変化まで含めた研究は少ない．

我々のこれまでの研究により，他者の信念の推定という困難な課題に対して，効率良く処理できる問題のクラスが 1 つ明らかになった．そして観測条件という領域知識の記述の指針を与えるとともに，意味論と入出力の明確なアルゴリズムを提案してきた．このようなアプローチの研究は，著者の知る限りほかに行われていない．

本論文では，意味構造との対応の明確な遡行的アルゴリズムの計算の無駄を指摘し，ログ表現に基づく前進的アルゴリズムを提案した．そして実行時間の測定により，前進的アルゴリズムが無関係な変化に対してほとんど時間を費していないことが分かった．

また遡行的アルゴリズムが最新の信念しか求められなかったのに対して，前進的アルゴリズムは過去の信念の変遷も推定できる．つまり，時間推論にとって重要な「時間軸を横から見る」という視点[5)]がネストした信念でも可能になったのである．これにより，ユーザとの会話にこのアルゴリズムを用いれば，最新の信念しか推定できないアルゴリズムに比べ，会話の幅が広がる．また，観測できなかった事象を推定して補うなどの推論能力を加えることも可能になる．

とはいえ，2.1 節でも指摘したように，$O^-(t)$ や $\mathrm{Post}^-(e)$ は，ログ表現でカバーしきれない．また，かなり複雑な処理を行っているため，意味構造との厳密な対応の証明も今後の課題である．

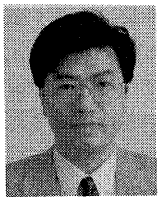
しかし我々は，むしろユーザとの対話などにおいて，他者の信念を推定する技術が（一般論としてではなく）具体的にどのような状況でどのように役立つかを明確にすることに重点を移したい．さらに，本論文で用いた手法は，単に他者の信念の推定に限らず，他の時間推論に適用できる可能性がある．これについても，今後研究を進めていきたい．

謝辞 本研究を支援してくださった石井健一郎元情報科学研究部長と，内藤誠一郎部長，勝野裕文グループリーダに感謝します．また，貴重なコメントをくださった査読者の方々にも感謝いたします．

参考文献

1) Allen, J.: *Natural Language Understanding 2nd. Ed.*, Benjamin Cummings (1995).
2) Ballim, A. and Wilks, Y.: *Artificial Believers, The Ascription of Belief*, Lawrence Erlbaum Associates (1991).
3) Clark, H.H.: *Arenas of Language Use*, The University of Chicago Press (1992).
4) Cohen, P.R., Morgan, J. and Pollack, M.E.: *Intentions in Communication*, MIT Press (1990).
5) Dean, T.L. and McDermott, D.V.: Temporal Database Management, *Artificial Intelligence*, Vol.32, pp.1–55 (1987).
6) Etzioni, O., Golden, K. and Weld, D.: Tractable Closed World Reasoning with Updates, *Proc. 4th International Conference on Principles of Knowledge Representation and Reasoning*, pp.178–189, Morgan Kaufmann (1994).
7) Fagin, R., Halpern, J.Y., Moses, Y. and Vardi, M.Y.: *Reasoning About Knowledge*, MIT Press (1995).
8) Isozaki, H.: Reasoning About Belief based on Common Knowledge of Observability of Actions, *Proc. 1st International Conference on Multi-Agent Systems*, pp.193–200, MIT Press (1995).
9) 磯崎秀樹：信念推定算法の終助詞選択への適用，電子情報通信学会 NLC 96-6 (1996).
10) 磯崎秀樹：誠実な主張と内省による否定を取り入れた信念推定アルゴリズム，人工知能学会全国大会論文集，pp.5–8 (1996).
11) 磯崎秀樹：エージェント指向プログラミングの実現に向けて—他者の信念の推定法，コンピュータ・ソフトウェア，Vol.14, No.4, pp.19–32 (1997).
12) Isozaki, H. and Katsuno, H.: A Semantic Characterization of an Algorithm for Estimating Others' Beliefs from Observation, *Proc. National Conference on Artificial Intelligence*, pp.543–549, MIT Press (1996).
13) 磯崎秀樹，勝野裕文：マルチエージェント環境における遡行的信念推定アルゴリズム，情報処理学会論文誌，Vol.38, No.3, pp.429–442 (1997).
14) Konolige, K.: *A Deduction Model of Belief*, Morgan Kaufmann (1986).
15) McCarthy, J.: *Formalizing Common Sense*, Ablex Publishing (1990).
16) O'Keefe, R.: *The Craft of Prolog*, MIT Press (1992).
17) Ullman, J.D.: *Principles of database and knowledge-base systems I*, Computer Science Press (1988).

(平成 10 年 4 月 3 日受付)
(平成 11 年 7 月 1 日採録)



磯崎 秀樹（正会員）

1983 年東京大学工学部計数工学科卒業．1986 年同工学系大学院修士課程修了．同年，日本電信電話（株）入社．1990〜91 年スタンフォード大学ロボティクス研究所客員研究員．現在，NTT コミュニケーション科学基礎研究所主任研究員．博士（工学）．人工知能の研究に従事．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，言語処理学会，AAAI 各会員．