

物語理解のための知識表現について

小 川 均
(大阪大学 基礎工学部)

1. まえがき

人工知能研究における種々のテーマに共通した問題は知識表現、および、それを用いた理解・推論メカニズムの構成である。計算機中に入人間の持つ知識を表現し、推論エンジンで用いられれば、各テーマへのアプローチが容易になると考えられるからである。

現在までに、セマンティック・ネットワークや、フレーム形式の応用として KRL や FRL 等多くの知識表現法が提案されている。これらのほとんどは、対象物体の状態や他の対象との関係を主に表現している。しかしながら、これらの知識を扱う方法に関する知識も存在する。例えば、物語を理解する際には、得られた事実が常識や前提と矛盾しないか調べる必要がある。無矛盾性のチェックのための手法も知識を扱うための知識であり、このような知識を表現する方法が問題となる。また、物語理解のための推論は、物体の複雑な関係からある状態を理解したり、時が経れば起る事象を予測したりしなければならぬ。本研究では、上述した知識、および、推論機構を実現するために μ -actor を用いた。さらに、物語の結果を予想するためにシミュレーションシステムを作成したが、シミュレーションのための知識も μ -actor を用いて表現した。

本論では、物語の結果予測に関する次の事柄について述べる：文理解に用いられる知識構造、シミュレーションに必要な機能、および、それに対する μ -actor の改良。

2. μ -actor

μ -actor は Hewitt の提案した actor の特徴の一部を実現したものであり、1977 年に発表された。本研究で用いた μ -actor は、幾つかの機能が追加され強力になっている。本章では、 μ -actor の概念、構造、特徴について述べる。

2.1 μ -actor の概念

大きな問題解析システムにおいては、新しい能力、追加、既にある能力の削除、修正が困難になる。この問題の解決法の一つは、システムを多くのスペシャリストに分けることである。すなわち、各スペシャリストは固有の知識と技法を持っており、小さな仕事を他と独立して行う。スペシャリスト間の情報交換や処理依頼の方法は唯一であり、メッセージを送ることである。大きな問題は以下のように解決される：問題を数個の副問題に分ける。一つの副問題に関して、それを解決できるスペシャリストが存在しなければ、それをさらに数個の副問題に分ける。あるスペシャリストがある問題を解決する場合、そのスペシャリストは、解決過程では、必要な情報を他から獲得し、解決後は、推論結果を他に送ったり、解の正当性をチェックする。このようにして、大きな問題が解かれる。

上記のスペシャリストを、計算機上で実現したのが μ -actor である。 μ -actor

を旨く定義すれば、システムのよい実行が得られる。また、 μ -actorの考え方を、各 μ -actorに適切な役割を与えることにより、種々の分野に応用することができる。例えば、一台の計算機を一つの μ -actorとみなすならば、複数台の計算機を用いた並列処理や分散処理の遂行のための強力な手法を供給する。

2.2 μ -actorの構造

μ -actorは図1に示すように行動情報部とデータ部から構成される。行動情報部は、メッセージが送られた時 μ -actorがとるべき行動を述べている。すなわち、データ部中のデータを使用するための手続的知識を示している。データ部はその μ -actorが知っている情報(例えば、持つべき固有値、属性、直接知っている他の μ -actorの名前とそれとの関係等)を蓄積している。すなわち、宣言的知識を示している。

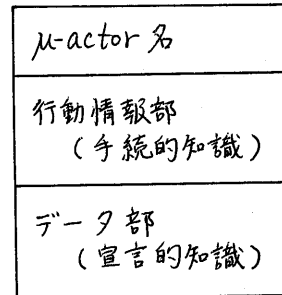


図1 μ -actorの構造

(i) 行動情報部

行動情報部は図2に示すように、メッセージパターンと複数個のルールとの集合である。

((メッセージパターン₁ ルール₁₁ ... ルール_{1m})
 (メッセージパターン₂ ルール₂₁ ... ルール_{2n})
 ...
 (メッセージパターン_p ルール_{p1} ... ルール_{pg}))

図2 行動情報部

(ii) メッセージパターン

現在のところ次の2つのメッセージパターンを用いている。

- (1) (MES: ?%N 内容 TO: ?CONT)
- (2) (RE: ?%N 内容)

?%Nはメッセージ番号%n(nは正数)とのみ照合される。メッセージ番号は他のメッセージとの区別や分類に用いられる。また、蓄積されるデータの識別にも用いられる。接頭辞“?”が付いた名前(例えば?CONT)はパターン照合用変数を表わす。この変数を参照する場合、“?”の代りに“!”(例えば!CONT)を用いる。“MES:”はメッセージを、“RE:”は返答を意味する。“TO:”は返答の送り先(continuation)を意味する。!CONTに“NO-ONE”を割り当てた場合、返答が不用であることを示す。!CONTに“ME”を割り当てた場合、メッセージを送った μ -actorに返答することを示す。これはサブルーチン・コールと同じである。

((P-C: プログラム_{p1} ... プログラム_{pg})
 (N-C: プログラム_{n1} ... プログラム_{nr})
 (C-E: プログラム_{c1} ... プログラム_{cs}))

図3 ルール

(iii) ルール

ルールはプログラムの集合であり、行うべき仕事が記述される。ルールは図3に示すように3つの部分から構成される。“P-C:”は前提条件(pre-condition)の略語である。“N-C:”は付随条件(next-condition)を意味し、随意である。“C-E:”

は発生事象 (caused-events) の略語であり、前提条件と付随条件の両方共満足された時のみ実行される。前提条件はルールを起動するための条件を述べている。付随条件は主に μ -actor 内での処理と他の μ -actor へのデータ参照を述べ、発生事象では他の μ -actor への仕事の依頼、又は、返答転送を記述する。

(iv) プログラム

プログラムは制限された LISP 言語で書かれる。リストやアトムへのポインタに影響を与えない LISP 関数 (MEMBER, ASSOC 等) は自由に使用できる。ポインタに影響を与える LISP 関数 (SETQ 等) に対しては $!m$ か $!!n$ (m と n は正数) の形に制限する。この制限は、ある μ -actor の行動が他の μ -actor へ意図しない影響を与えないようにする為である。変数 $!m$ は PROG 変数と同様に使用できる。データ部を操作する基本関数 (FGET, FPUT, REMOVE) が用意されている。関数 FGET はデータ部からデータを獲得するのに使われる。関数 FPUT と REMOVE はデータの挿入、削除をそれぞれ行う関数である。これらの関数は FRL のそれとよく似ているが、フレーム名が不要であることが相違点である。本研究で用いられる関数は記述場所により対象となるデータ部が決まるため、他の μ -actor に影響を与えない。メッセージ転送の記述は次のように行う：

($\Rightarrow$ μ -actor 名 メッセージ)。

(v) データ部

データ部は図4に示すようなフレーム記述によって表現される。フレームの副構造はスロット、フェイスット、データ、コメントから成る。スロットはその中のデータがどのような特性と結び付けられているか示す。フェイスットはデータがどのような種類か (例えば、値、制限、手続き等) を示す。データはフェイスットと関係した基本情報、コメントはデータに関連した付加情報である。

2.3 μ -actor の行動

μ -actor は、メッセージ M_i が送られた時、起動され、その役割を果たそうとする。もし、 M_i が照合されるメッセージパターン P_i があれば、 P_i と対応するルールの中から、前提条件が満足されるルール R_i を探す。 R_i 中に付随条件があれば、そ

```

((スロット1 (フェイスット11 (データ111 コメント111)) ... (データ11h コメント11h)))
 (スロット2 (フェイスット21 (データ211 コメント211)) ... (データ21i コメント21i)))
 (スロット3 (フェイスット31 (データ311 コメント311)) ... (データ31j コメント31j))
               (フェイスット32 (データ321 コメント321)) ... (データ32k コメント32k)))
      ...
 (スロットs (フェイスットs1 (データs11 コメントs11)) ... (データs1m コメントs1m)))

```

図4 データ部のフレーム記述

の付随条件を実行する。付随条件が満足されれば、発生事象を実行する。付随条件が満足されない場合、又は、どのルールのも前提条件も満足されない場合は、 M_i が照合されるメッセージパターンを探す。このようにして、発生事象が実行されるか、又は、すべてのメッセージパターンについて調べられるまで処理を行う。

2.4 μ -actor用デモン

デモン関数は、物語理解システムの構成のために μ -actor に付けられた。物語中に現われる人、動物、物体に対応して μ -actor が作られる。それぞれの行動は行動情報部に記述される。データ部には特定の値や他物体との関係が記述される。デモン関数を用いなければ、物語から得た世界状態のシミュレーションの結果を得るのは非常に困難である。例えば、太郎という名の人に対応した μ -actor 太郎 (μ -actor の名前には下線を引く。)について考える。太郎がある場所 X に到達した時、 μ -actor 太郎 が他の μ -actor にメッセージを送る手法をどう表現するのか。この場合、太郎 は、彼が場所を変更した時は何時でも場所をチェックしなければならない。これを行動情報部に書くとうすると、場所を変更する為のメッセージパターンが複数個ある場合、それぞれに場所のチェックのためのプログラムを記述しなければならない。これを旨く記述するにはデモン関数が必要である。デモンは常に定められた状態かどうか監視し、もしそうであれば適当な行動をする。この例では、デモンは 太郎 が場所 X に居るかどうか調べ、そうであればメッセージを転送する。

我々は、FRLと同様スロット中にデモンを記述することが出来る。しかしながら、 μ -actorの起動方法(すなわち、メッセージが送られた時起動される。)も考慮して以下の2方法が考案された。これらの手法によって記述されたデモンは、 μ -actorが起動された時に有効となるので、データ部の各スロットに関係ない事象も扱うことができる。現在は *post-action* のみが用いられている。

(i) *Pre-action*

pre-action デモンは、データ部の *pre-action* スロットにルール形式で記述される。 μ -actor がメッセージを受理した時、その μ -actor が行動情報部に従ってその役割を実行し始める前に、*pre-action* のルールが評価される。

(ii) *Post-action*

post-action デモンは、データ部の *post-action* スロットにルール形式で記述される。 μ -actor が行動情報部に従ってその役割の実行が終了した後、*post-action* スロット中のルールが評価される。

2.5 μ -actor の特徴

(i) 宣言的知識と手続的知識

知識は、宣言的形式、又は、手続的形式で表現される。以下の例は、 μ -actor 中の知識の表現法について述べている。人間に関する知識を扱う μ -actor Human について考える。我々は人間について多くの知識を持っているが、ここでは2つの簡単な例を示す。「人間は動物である。」という知識は宣言的に表現される。こ

の知識は、 μ -actor Human のデータ部の AKO スロットに値 ANIMAL を挿入することにより表現される。手続的に表現される知識もある。次の文について考える：「ある人の進行方向に障害物（木や壁等）がある時、もしそれが低く、小さければ飛び越す。そうでなければ、回り道をする。」この知識はルール形で表現され、 μ -actor Human の行動情報部、又は、デモンとして pre-action スロットか post-action スロットに記述される。

(ii) モジュール性

モジュール性を得る為には、大きな知識を小さなモジュールのグループで表現できることは必須である。2. 1 で述べたように、 μ -actor は他と独立して小さな仕事を達成することができ、メッセージ転送のみが他との相互作用方法である。メッセージには受理した μ -actor が何をすべきかが記述されている。逆に、メッセージを送った μ -actor が何を要求しているのかが書いてある。したがって、転送又は受理するメッセージからその μ -actor の動きが理解できる。 μ -actor を修正する場合には、転送又は受理するメッセージのチェックを行えばよい。

モジュール性は、知識のクラス分け、および、階層的構造を容易に記述する構成法をもたす。以下の例は、階層的構造の知識表現について示している。一般に人間に対する障害物は木や火、壁等である。もし次郎が犬を嫌いならば、犬は次郎の障害物である。 μ -actor システムにおいては、これらの障害物は図 5 に示されるように階層的構造に明記される。

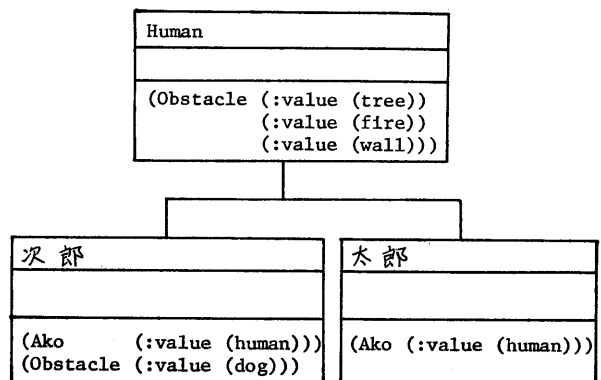


図 5 階層的構造の例

(iii) 擬似並列処理

本節では綱引きを例として述べる。太郎と次郎の綱引きをシミュレートするためには、 μ -actor 太郎と μ -actor 次郎 が同時に起動される必要がある。しかしながら、このことは、我々の設備で遂行が不可能であるので、擬似的に並列処理を行う。

ここで、綱引きに関する知識を持つ μ -actor SIMTW を用いる。図 6 に各 μ -actor の関係を示す。

SIMTW は、太郎と次郎の両方からメッセージが送られないと起動されない。両者からのメッセージを受理すれば、綱引きの状態を調べる。例えば、太郎と次郎が同じ力で引き合っておれば、ゲームは続行される。もし片方が他より力

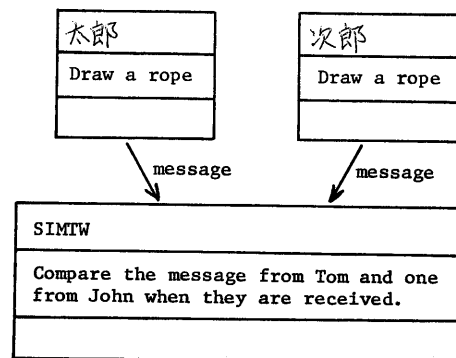


図 6 綱引きの μ -actor の構成

が強ければ、勝利者となる。したがって、片方が倒れた場合、ロープを引く力がないとみなし、その者が敗者となる。

もし、各 μ -actor がそれぞれ異なる計算機で実現されれば、並列処理が可能となる。現在のシステムでは一台の計算機を使用しているので、 μ -actor の起動の同期をとるために μ -actor synchronizer を用いた。詳細は第4章で述べる。

3. 問題点とアプローチ

物語を計算機によって理解するためには多くの困難な問題が存在する。ここでは、言語的問題ではなく、各文が単独に解釈された後、文と文の関係の推論、得られた情報から次に起こりうる事象の予測等、物語の筋を理由付けする推論機構における問題点およびそのアプローチについて述べる。

3.1 文の理解度

物語を理解するためには、文を詳しくすぎず、大ざっぱすぎないように解釈し意味を抽出しなければならない。文理解の詳細さは、物語の文脈や読者の興味等に依存する。「太郎が歩いている。」という文を例にして説明する。ある人はこの文から「太郎が前方に進む。」という意味を得る。これは、「太郎が町に着いた。」という文を理解するのに十分な理由を提供する。しかしながら、「太郎は疲れた。」という文に対しては十分ではない。この文を理解するには次のように、歩行に関する知識と推論が必要となる：「歩く」行為は筋肉の緊張と緩和から成る。ところで、一般的に疲労の原因の一つは筋肉の緊張と緩和の繰り返しである。そこで、他に特別な理由がない限り、太郎が長い距離を歩いたため疲れたと推論する。

上記の推論における重要点は、我々は知識を階層的に記憶しており、それを旨く利用していることである。計算機においては、Minskyの提案したフレーム形式が知識の階層的表現に適している。前章で述べた μ -actor はデータ部の表現にこのフレーム形式を用いており、また、フレーム形式で書かれた知識を扱う知識（参照・追加・修正・削除を行うもの）を行動情報部に記述できる。

3.2 変化する状態の記述

物語では筋が進むにつれて、すなわち、時間変化に従って状態が変化する。これを表現するためタイムフラグを用いた。この方法は、記述法としては便利ではないが、履歴を記憶するための特別な機能が不要であることから本研究ではこれを使用する。

図7は μ -actor 太郎のデータ部を示している。Snは Stepn (添字nは正数) の略語である。S1はシステムの初期状態を示す。添字は時が経つにつれて1つずつ増加する。図7は、太郎が S1 では P16 (Place 16)、S2 では P17 に居ることを示している。

```
(AKO (:VALUE (HUMAN)))  
(MIGHTY (:VALUE (5)))  
(HEIGHT (:VALUE (LITTLE)))  
(PLACE (S1 (:VALUE (P16)))  
(S2 (:VALUE (P17))))  
(ACT (S1 (:V (MOVE)) (:G (P20)))  
(S2 (:V (MOVE)) (:G (P20))))
```

3.3 特別な状態において起こる事象

物語に関する質問に答えるために、知識ベースの参照、将来起こる事象の予測が必要な場合がある。予測のための一手法は、物語の

図7 μ -actor 太郎のデータ部

シミュレーションである。シミュレーションにおいて、種々の事象が発生するが、特定の事象の発生を見付け、処理を行うために、2.4で述べたデモンを使用する。

4. シミュレーションシステム

4.1 プランニング

本システムでは、人や犬、木等の物体に関する一般的知識を持つ μ -actor が存在する。これらの μ -actor は具体的な物体に対応する μ -actor にデフォルト値や一般的な行動、特徴を与える。シミュレーションは、これらの物体物に対応する μ -actor の持つ情報に基づいている。例えば、 μ -actor John は「John は人間である。」という文に対応して作られる。John は人間の特徴を得る。例えば、彼は哺乳動物としては小さい。彼は自由に動け、ゲームもできる等である。もちろん、新しい情報が得られた時それは自由に修正できる。

もし、文が既に作成されている μ -actor に関することであれば、情報がその μ -actor に追加され、行動情報部又はデータ部に記述される。

行動に関する情報は主に行動情報部に記述される。文「John は P20 へ歩く。」に対して、 μ -actor Walk は John に歩くためのルールを送る。このルールは目標へ動くための行動が表現されており、必要ならば、飛び越す等の他のルールを呼び出す。行動に間接的に関係する情報は、デモンの形で表現される。文「John は犬が嫌いだ。」に対して、彼が犬と出会う又はその恐れがある場合、犬から避けるルールを post-action スロットに入れる。

ここで問題となるのは「出会う」ことの判断である。本システムでは、図8に示すように世界を15個に分割した。2つの物体が同じ場所があれば2つは出会っている。隣の場所があれば近づいていると判断する。上記以外は離れていると判断する。また、シミュレーションの結果を表示する際にも便利である。

P11	P12	P13	P14	P15
P16	P17	P18	P19	P20
P21	P22	P23	P24	P25

図8 15に分割されたシミュレーションの世界

4.2 シミュレーション

シミュレーションの際、 μ -actor は擬似的に並列処理するため、旨くプランニングされなければならない。本システムでは μ -actor Synchronizer を用いており、自力で動く物体に対応する μ -actor に周期的にメッセージを送る。すなわち、1ステップに1回しかメッセージを送らない。

プランニングされた μ -actor の動きは、物語から得られた世界の変化をシミュレートする。このシミュレーションの結果を得るために、意味処理部 (mean processing part) が必要である。例えば、綱引きの場合、力の強い人Aが綱を引くという事実として世界を変化させる。この事はAが勝ち、相手が負けたことを意味している。この意味付けは、シミュレーション部ではなく意味処理部が行う。意味処理部もまた μ -actor で実現されている。シミュレーション部から意味処理部への相互作用はメッセージ転送で行われる。綱引きの例では、デモンが使用され、綱が引けたかどうか監視する。図9に2つの部分の構成・関係を示す。

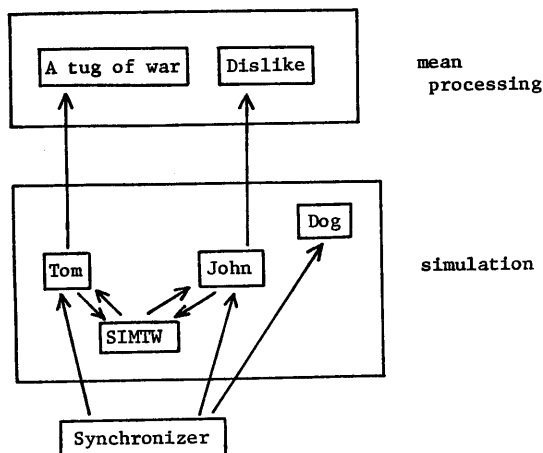


図9 意味処理部とシミュレーション部の関係

4.3 例

シミュレーションの前提となる情報を図10に示す。文中、P16, P18, P20は図8で示した世界の位置を示している。図10の内容が与えられた後に、疑問文

"Can Tom become a winner of the game?" (9)

の入力により、シミュレーションが始まる。本システムは自然語解析機を持っていないので、実際には、図11に示すCase文法の形で入力する。1から9までの各文は、図10の文と疑問文(9)に対応している。文字V, A, O, L, I, Gはそれぞれ、動詞、主体、目的物、場所、道具、目標を示している。リスト(M(?G))は疑問文であることを示している。

写真1はシミュレーションの結果を示している。Spot(犬)がJohnに近づいたので彼が逃げ出し、それ故、Tomがゲームに勝ち、喜んでいるところである。

5. おすび

物語の結果を予測するための知識表現法および推論機構について述べた。また、これらにおける問題について考察しアプローチを取った。このアプローチおよび予測機構のためのシミュレーションシステムはμ-actorを用いて実現した。μ-actorを用いれば、事実やその関係の記述だけでなく、それらを扱う手法までも記述でき、知的システムの開発が容易となる。μ-actorは、京大下研究室で開発されたLISP1.7で遂行されている。

John and Tom are human. (1)
Ropel is a rope. (2)
Spot is a dog. (3)
John, Tom and Ropel are at place P18. (4)
Spot is at place P20. (5)
John dislikes a dog. (6)
John and Tom play at a tug of war. (7)
Spot walks toward place P16. (8)

図10 シミュレーションに用いた情報

1. ((V IS-A) (A (TOM JOHN)) (O HUMAN))
2. ((V IS-A) (A (ROPE1)) (O ROPE))
3. ((V IS-A) (A (SPOT)) (O DOG))
4. ((V IS-AT) (A (TOM JOHN ROPE1)) (L P18))
5. ((V IS-AT) (A (SPOT)) (L P20))
6. ((V DISLIKE) (A (JOHN)) (O DOG))
7. ((V PLAY-AT-A-TUG-OF-WAR)
(A (TOM JOHN)) (I ROPE1))
8. ((V WALK) (A (SPOT)) (G P16))
9. ((M (? G)) (A (TOM))
(V PLAY-AT-A-TUG-OF-WAR) (G WINNER))

図11 Case文法を用いた入力文と質問文。

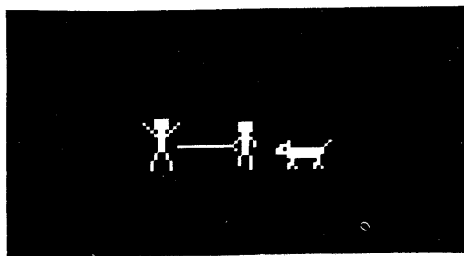


写真1 シミュレーションの結果