6L-6

多重世界機構による時間の表現と問題解決

中島秀之，諏訪基
(電子技術総合研究所)

1．はじめに

論理に基づく知識表現システムUranus［中島 1985］の多重世界機構には，知識の階層構造や状態の変化などを記述する能力がある．ここでは，この機構を時間の経過にともなう状態変化の表現に用い，問題解決を行なうシステムについて述べる．

他の論理型言語では，状態の変化の記述が不得意であるが，Uranusでは各状態をひとつの世界として記述し，各々の世界には前の状態との変化だけを記述しておくことにより状態の変化が容易に記述できる．また，これによりいわゆるフレーム問題が避けられる他，過去に遡った状態の再現も容易であるので，状態の変化を含む問題解決（プランニング）に向いている．

バックトラックを用いたプランニングでは一つの解しか求めることはできず，いくつかの解の比較などはできない．それに対し，多重世界機構を用いると，いくつかの並行世界が記述できるため，それらの間の比較なども容易である．

2．状態変化の記述とプランニング

状態の変化を記述するには，その状態に対応するデータベースを作り，それを実際に書換えてしまうのが最も簡単である．プロダクションシステムなどはこの方式をとっている．しかし，これには以下のような欠点がある．

1)過去の状態が再現できない．

2)仮想的状態をつくってみることが困難である．

特に，複数のゴールを同時に解かなければならないような問題の場合に，プラン間の相互作用があり，後のプランが先のプランの結果を壊してしまうかもしれないので，単純に部分解を連結するわけにはいかない．このような場合は，まずあるゴールを解き，続いて先のプランの時間を遡ったりしながら適当な時点に次のゴールの解を挿入するという手段がとられる．この場合，過去の状態の再現が必要である．

プランにいくつかの可能性がある場合に最適解を選ぶ問題では，プラン間の比較を必要とする．この場合，仮想状態間の比較を行うことになる．

このようにプランニングにおいては実際にデータベースを変更する方式は好ましくない．一方，古典論理の枠組みを踏襲し，状態の変化を状態間の関係として記述する方式では様々な状態を同時に扱うことが可能になるが，計算量や記述量が膨大になるというフレーム問題［McCarthy 1977］がある．

Uranusで用いる方法は両者の中間的手法である．各中間状態は一つの世界として保存されるが，その世界には一つ前の世界との差しか書いてない．前の状態に新たに追加された状態は

```
(assert P)
```

として，削除された状態は

```
(deny P)
```

として記述される．

```
(with time-slice1
   (assert (on A B))
   (deny (clear B)))
```

のようになる．時刻の経過にともなって世界を重ねていけばよい（Uranusによる時間の表現の詳細に関しては［中川 1986］を参照されたい）．t1, t2, ..., tnという世界を考え，t1からtnの方向に時間が経過した状態は

```
(with t1 (with t2 ... (with tn ...)...))
```

として表現できる．

また，新しい状態に対応する世界を取り除くことにより過去の再現ができる．例えば

```
(with t1 (with t2 (with t3 ...)))
```

では，時間がt3までしか進んでいない状態を表現していることになる．

世界の重ねかたはプログラムで指示することができるからどのような組み合わせでも作ることができる．例えば途中のアクションを起こさなかった仮想世界というの

Representation of Time Sequence and Problem Solving Using Multiple World Mechanism
Hideyuki Nakashima and Motoi Suwa
Electrotechnical Laboratory

は，世界の並びからそのアクションに相当する世界を抜くだけで実現できる．

3．Uranusによる問題解決プログラム

ここで述べる問題解決プログラムは一般の問題を扱えるようになっており，それに各々の問題に応じたドメイン固有の手段を入力することになる．これにはachieveという述語：

(achieve <ゴール> <前提> <手段>)

を定義すればよいことになっている．

例えば，積木の世界では状態を変更するオペレータはmoveしかない．また積木の状態としては，何の上にあるか(on)と，上に何も乗っていないか（clear)，の二つの情報だけがあればよい．<ゴール>の部分にonあるいはclear，<手段>の部分にmoveが書かれることになる．

オペレータによる状態の変化はaddingとremovingという述語により定義する．例えば積木*xを*yから*zに移すと(on *x *z)と(clear *y)が新たに成立するようになるということは，

```
(assert (adding (move *x *y *z)
                ((on *x *z) (clear *y)))
```

のように表現する．

4．問題解決プログラムの自己適用

プランどうしの比較などをさらに押進めることにより，プラン自身に関する考察が可能になる．例えば，プランニングの途中でのループの検出などがそれである．これは，プランニングに関するプランニングという意味でメタ・プランニングになる（以下ではもう少し一般的にメタ問題解決という言葉を使う）．

メタ問題解決機構としては，現存の一般問題解決機構を順次機能強化していくことにより，Uranusのプログラム一般を受付けうる問題解決機構を作る．続いて，この一般問題解決機構を自分自身に適用することによりメタ問題解決機構とする．この結果できたものは，何か問題を与えられた場合にそれを解決（問題解決機構）しながら，同時にその問題解決の仕事自身を問題解決（メタ問題解決）していくことになる．後者には，問題解決の仮定におけるループの発見，新しい手順の発見などの問題解決のストラテジー自身に関する問題解決が含まれる．

3．で述べた問題解決システムに簡単なUranusプログラムを実行させる実験を行っている．これはachieveという述語を定義する代わりに，Uranusの述語定義をそのまま使うものである．現在，システム組み込みの述語は呼べないが，ユーザ定義のものだけからなる“ピュアな”プログラムは実行可能である．

例えばフィボナッチ数列の再帰的定義を与えると，本システムはそれを順次下から計算するプランを生成する．また，フィボナッチの再帰的定義をそのまま実行すると，同じ計算が繰り返され，効率が悪いが，ここでは一度達成したゴールを再実行することはないので，同じ計算は繰り返さない．

このように，生成されたプランは非常に効率が良いが，プランニング自信はしらみ潰し的な戦略を用いているので，プランニングに時間を要するという欠点がある．一般的なプランを生成してくれるのならそれでもよいのだが，ここで扱っているのは

(fibonacci (s s s s) *f)

のような特殊な問題に限られる（組み込み述語が使えないので，数をリストとして，s の並びで表現している）ので，プランニングの間に，再帰的ににでもよいから計算してしまった方が早い．プランニングを効率よくするには，もう1段外にプランニングシステムをかぶせれば良い．

このように外側にどんどん膨らましてゆけば，内側の動作はどんどん効率の良いものに変化していく．しかし，これは問題の本質的な解にはなっていない．先に述べたように，自分自身を自分でプランニングする必要がある．この点が今後の課題である．

参考文献

John McCarthy: "Epistemological Problems of Artificial Intelligence" Proc. of IJCAI-V, pp. 1038-1044(1977)

中川裕志，中島秀之，柳田昌宏：“Uranusの多重世界機構による時間推移の表現法”，情報処理学会論文誌，Vol. 27, No. 3, pp.297-303 (1986)

中島秀之：“超時空プログラミングシステムUranus”，第26回プログラミングシンポジウム予稿集，pp.13-23 (1985)