知識工学と人工知能 39－8
（１９８５．３．１９）

多重世界環境におけるＴＭＳと概念生成

中川裕志・野村　馨・岸川徳幸
（横浜国立大学　工学部　情報工学科）

１．はじめに

多重世界環境下での学習について考えてみると、新たな知識が入力されたときに新しい世界を作り出す場合が学習のひとつと考えられる。そこで、新しい世界を生成する契機を得る方法として多重世界環境下にＴＭＳを持ち込み、ＴＭＳによる矛盾解消を世界の生成へ向けることにする。

ＴＭＳは元々、平坦な定義空間で無矛盾性を保持するシステムだが、多重世界環境下でも、ある世界のみを見れば単一な定義空間とみなせる。そこで、ある世界に新しい情報が入力されたときに注目して、その世界でＴＭＳを実行し、状態を書き換える必要があれば、新しい世界を作り出すことにした。

２．ＴＭＳ

２．１　データ構造

本節では有名なＴＭＳについて簡単に復習する。

［ＮＯＤＥ］

それぞれの node は知識の belief を表わしていて、beliefはinとout という２つのstatusを持つ。beliefの status が in であるということは、その belief を信じ得るだけの十分な根拠があるということである。その為には、その belief は少なくとも１つの正当（valid)な justification（justification については次節で述べる。）を持っていなくてはならない。なぜなら、believed in （信じ得る状態）である為にはその根拠が必要であり、その根拠となるものが正当な justificationであるからである。これとは逆に、out であるということは、正当ではない（invalid な)justificationのみ存在するか、あるいは根拠となるようなjustification が存在しないということである。

る。繰り返しになるが、 in , out ということは根拠の有無によるものであり、基本的に真理値とは異なるものである。

[JUSTIFICATION]

justification はnode間の関係を表わしているが、それは次のようになっている。

<node> （ＳＬ <inlist> <outlist> ）

ＳＬ(Support-List justification)は validであるか invalidであるかのどちらかである。ＳＬ justificationが validである為には次のどちらかでなくてはならない。

① <inlist>に含まれるnodeがすべてinであって、<outlist> に 含まれるnodeがすべてout である。

② <inlist>と<outlist> が共に nilであって要素を持たない。

これらの場合にＳＬjustification は validとなる。ＳＬがvalid であれば、<node>は正当な根拠を持つことになり、状態は in となる。これら以外（次のような場合）ではこのjustification はinvalid である。

① <inlist>に outであるような node が含まれている。

② <outlist> に in であるような node が含まれている。

２．２ 依存関係を表わす術語

Support-status

その node が表わしている belief の status である。前にも述べたようにstatusがinであるということは、そのnodeが believed in であることを表わしている。

Supporting-justification

そのnodeのstatusを決定しているjustification である。nodeがinである場合、そのnodeのvalid なjustification のうちの１つである。nodeがout であれば、そのnodeのjustification のすべてがSupporting-justificationになる。

Supporting-nodes

ある node の status を決定している node である。その node がinである場合、Supporting-justificationのinlistと outlistに含まれる node がすべて supporting-nodes になる。逆に、そのnodeがout である場合にはその node のすべての Justifications が invalidであるから、それらの justifications の inlist に含まれる node のうちで outであるものと outlistのうちで in であるものがすべてSupporting-nodesになる。

Consequences

ある node をjustificationsの inlist 、outlist に含んでいる node がその node の Consequences となる。

Affected-consequences

ある node の Consequences 中の node の Supporting-nodes にその node 自身が含まれているとき、 Consequences に含まれるその node は Affected-consequencesに含まれる。つまり、あるnodeの Affected-consequencesはその node の status が変わったとき、 status が変わる可能性のあるConsequencesある。

Repercussions

Affected-consequences からなる closureである。あるnodeの Repercussionsはその node のAffected-consequences とAffected-consequences に含まれる node の Repercussionsを含んでいる。つまり、ある node の Repercussionsはその node の status が変わったとき、 status が変化する可能性がある node の集合である。

３．ＴＭＳ的記述での世界間の関係

ＴＭＳは平坦な世界ですべての知識を扱っている。これらの知識を階層構造を持つような多くの世界の知識として分割するとき、それぞれの世界間の関係によってどのような分割をするべきかを考えなければならない。

３．１　super-world と sub-world

図１のような例を考える。世界Ｂでは

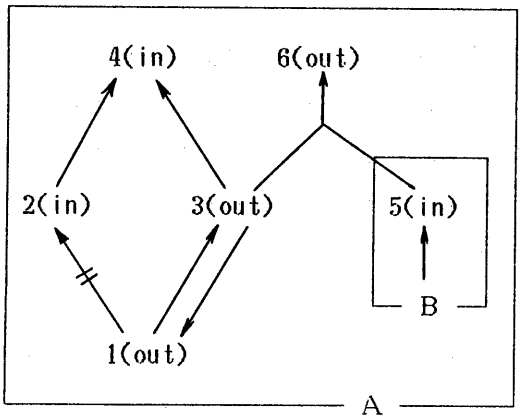
node 5 (SL ( ) ( ))

が、世界Ａでは

node 1 (SL (3) ( ))
node 2 (SL ( ) (1))
node 3 (SL (1) ( ))
node 4 (SL (2) ( ))
(SL (3) ( ))
node 6 (SL (3 5) ( ))

が定義されている。それぞれのnodeのstatusは

図１．１，２，…，６は個々のnode。矢印はすべてjustification 。交差のない矢印はinlistを、交差のある矢印はoutlist を表わしている。



世界Ｂ　node 5　in　　世界Ａ　node 1　in
node 2　out
node 3　in
node 4　in
node 6　out

となっている。世界Ａでは世界Ｂでのnode 5のstatusを継承することによって、それぞれのnodeのstatusを決定しているから、世界Ｂが世界Ａのsuper-world になっている。世界ＡではＢでのnodeのstatusを参照しているだけで、Ａへの新しい情報の入力によってＢのnodeのstatusを書き換えられない（super-world のnodeのstatusを現在の世界で書き換えることを許さない）。よって、super-world へのjustification は存在せず、特にそのようなjustification が必要であれば、世界の階層構造を組み替える必要があると考える。このようなことからＡからはＢのstatusだけが見えればよく、他のデータについては見える必要がないことがわかる。

３．２　並列な世界

互いに同じsuper-world を持つ世界を並列な世界という。図２の世界Ａと世界Ｃのように並列な世界間では互いに共通のsuper-world からstatusを継承する。

図の例では世界Ｂのsub-world として世界ＡとＣがある。これらの世界では世界Ｂの

node 5のstatusを継承する。これによって世界Ａでは２．１で述べたようにstatusが割り当てられ、世界Ｂでは

node 1　in

node 2　out

node 3　in

node 4　in

node 6　out

となる。

４．多重世界環境におけるTruth Maintenance Process と新しい世界の生成

これまでにＴＭＳにおける世界間の関係がわかった。この関係を基にしてＴＭＳを用いた新しい世界の生成を考える。

世界の生成が必要なのは、ある世界で矛盾が生じた場合であるから、矛盾が生じなければ、その世界内で処理をしても構わない。

ＴＭＳで、いくつもの世界で継承されるのはnodeのstatusであるから、statusが変更されたときに新しい世界を作り出すのが自然である。

さて、新しい世界を作るにあたって、注意すべき事柄がいくつかある。これらをまとめてみると、

① super-world のnodeへ向かうjustification は存在しない。よって、nodeはsuper-world のnodeの根拠にはならない。

② 並列な世界のnodeへ向かうjustification は存在しない。

ある世界で宣言されているjustification は、その世界内かその世界のsuper-worldのnodeを根拠にしているものに限られる。

ここで新しい世界を生成する方法を考えてみる。新しい世界では元になる世界のデータをコピーし、そこにjustification を付加して、statusをそれぞれのnodeに新しく割り当てる。さて、こうして作られた世界と、元の世界との関係を考えると、この２つの世界がsuper-world とsub-world の関係にあるとすれば、どちらかの世界がもう一方の世界のstatusを継承していなければいけない。ところが、これらの世界は互いにコピーであったわけだから、宣言されているnodeは共通である。しかし、そのstatusはjustif

ication を加えたことによって異なっているため、super-world とsub-world の関係ではない。

もう１つ注意すべきことは、①の条件によってsuper-world のstatusは変化しないということである。つまり、新しい世界とその元になった世界とで、super-world から継承してくる部分については等しく、共通であり、従って、新しい世界と元になった世界は並列であることがわかる。

これで新しい世界を生成する方法と、元の世界との関係がわかったからそのアルゴリズムは次のようになる。

Step1　justification を付加する世界の現在のデータをセーブしておく。justification を付加するnodeのJustification-set に新しいjustification を加える。そのjustification のinlistとoutlist に含まれるnodeのConsequencesに、新しくjustification を付加するnodeを加える。

もし、そのnodeがinであれば、これでこの行程は終了。

out であれば、新しく付加されるjustification がvalid かどうかを調べる。そのjustification がinvalid であれば、inlistのnodeのうちのout であるものとoutlist のnodeのうちのinであるものをSupporting-nodesに加え、そのjustification をSupporting-justificationに加える。加えるjustification がvalid であれば、Step2 へ進む。

Step2　新しい世界を並列に作り出す。新しい世界に元の世界のデータをコピーし、元の世界を書き換えを行なう前の状態に戻す。以後の処理は新しい世界で行なう。

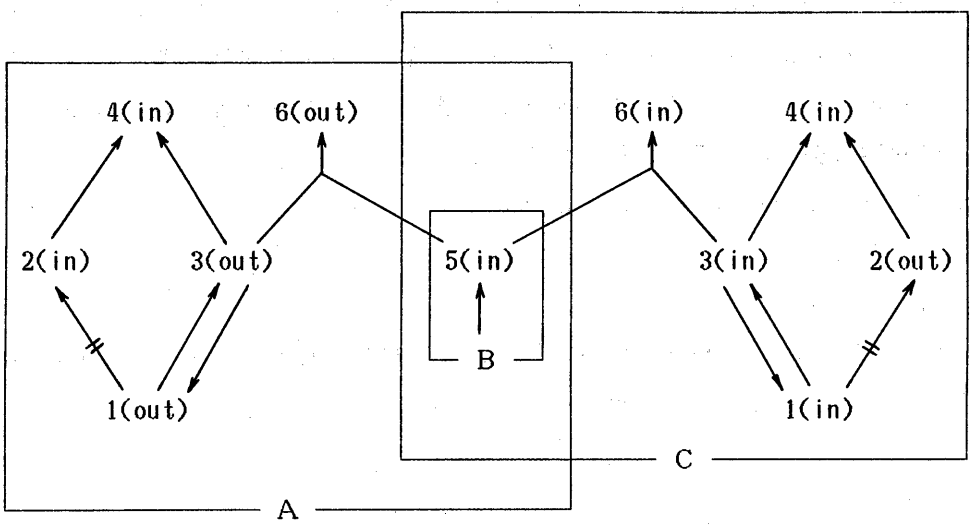
justification を加えるnodeにAffected-consequences がなければ、そのnodeのstatusをinに変え、加えられたjustification のinlistかoutlist に含まれるnodeをSupporting-nodesにし、そのjustification をSupporting-justificationにして、終了。

Affected-consequences があれば、justification を加えるnode自身とそのnodeのRepercussions からなるリストＬを作り、Step3 へ進む。

Step3　Ｌに含まれるnodeのstatusをnil であると仮定する。Step4 へ進む。

Step4　Ｌのそれぞれのnodeに可能な限りのstatusを割り当てる。その方法としては、まず、そのnodeをinであるとして、Step4aによってそのnodeのjustification を調べる。次にそのnodeのすべてのConsequencesをStep4aによって調べる。不適であればout であるとして同様にjustification とConsequencesを調べる。うまくいけば、Ｌの他のnodeに対してStep4 を実行する。Ｌのすべてのnodeについてstatusの割り当てが済めば、St

図２．　Ａは世界Ａから見える範囲。同様に、Ｂは世界Ｂで、Ｃは世界Ｃで見える範囲を示す。Ａ及びＣからは世界ＢのstatusがＭえる。即ち世界ＡとＣのsuper-world として世界Ｂがある。



ep5 へ進む。

Step4a　そのnodeのstatusがnil であれば、何もしない。

そのnodeがinであれば、十分に根拠があってinvalid なjustification （inlistにout であるnodeが含まれるか、outlist にinであるnodeが含まれる）以外のjustification が少なくとも１つはなくてはならない。　そのnodeがout であれば、十分に根拠があってvalid なjustification （inlistに含まれるnodeがすべてinで、outlist に含まれるnodeがすべてout ）があってはならない。

Step5　Ｌのそれぞれのnodeに対してStep5aを実行し、それぞれのnodeのSupporting-nodesとSupporting-justificationを決定すると、終了。

Step5a　そのnodeがinであれば、そのnodeのjustification のうちのvalid であるものをそのnodeのSupporting-justificationとし、inlistとoutlist に含まれるnodeをすべてSupporting-nodesにする。

そのnodeがout であれば、そのnodeのjustification-set をSupporting-justificationにして、それぞれのjustification のinlistのnodeのうちのout であるものと、outl

ist のnodeのうちでinであるものをSupporting-nodesにする。

例を用いて説明する。図２で世界Ａと世界Ｂだけがある状態を考える。ここで

node 3 (SL (2) ( ))

を付加する。このとき、node 3のstatusはout であり、(SL (2) ( ))はvalid なjustification である。よって、矛盾が生ずるから、新しい世界を生成する。ここで生成される新しい世界をＣとする。世界Ａのデータを世界Ｃにコピーする。世界Ａのデータを処理の前の状態に戻す。以後の処理は世界Ｃで行なう。

node 3のAffected-consequences はnode 1,6である。そこでnode 3のRepercussionsとnode 3からなるリストＬは

L=(1,2,3,4,6)

これらのnodeにstatusを割り当てると図のようになり、世界Ｂと世界Ｃの共通のsuper-world が世界Ａであることがわかる。

６．おわりに

ＴＭＳを利用した世界の生成では今のところ根拠の書き換えを行なっていないが、知識が蓄えられていく過程で根拠を書き換える必要が出て来るだろう。そのためには依存関係を辿っていって書き換えを行なうDependency Directed Backtrackingが必要である。また、根拠の書き換えはsuper-world のnodeの書き換えになることもあり、その場合の世界の生成の方法も問題となろう。

【参考文献】

１）中島他、「Prolog/KR からＵranus へ」
第36回知識工学と人工知能研究会資料（1984）

２）中島　「Prolog/KR における知識表現」
情報処理学会第28回全国大会論文集　pp.1139-1140（1984）

３）J.Doyle　"A Truth Maintenance System"
Artificial Intelligence (1979)

４）Nils J.Nilson　"Principles of Artificial Intelligence"
Tiago Publishing Company