5L−1

状態記述系列からのオペレータの生成

下畑 光夫　山田 誠二　安部 憲広　辻 三郎
(大阪大学・基礎工学部)

simohata@tsuji-lab.ce.osaka-u.ac.jp

1　はじめに

これまでプランニングについては，STRIPS,NOAH,SIPE等で色々と研究されてきている.プランニングとは，あらかじめプランナーに組み込まれたオペレータを用いて環境を初期状態から目標状態まで変化させるオペレータの系列を求めるものである．従来のプランニングにおいては，目標まで到達するために必要なオペレータはあらかじめプランナーにすべて組み込まれている事が前提となっている.

しかし，目標を達成するために必要なオペレータが初めからすべて組み込まれているとは限らない．むしろプランナーに与えられるであろう問題をあらかじめ想定できて，そのために必要なオペレータをすべて定義しておくことが可能な場合というのは希であろう．そこで与えられた問題に対し，オペレータの不足からプランニングができない場合が生じてくる．このようなオペレータの不足に陥った場合，なんらかの方法で新たにオペレータを定義する必要が生じてくる．その時にシステムに初期状態から目標状態への状態変化を表わす状態記述系列を与え，システムが各状態記述間からオペレータを生成するという方法で定義する．この方法であれば，与えた問題に必要とされるオペレータが容易に定義できる.

また，言い換えればこういった機能があればあらかじめプランナーにオペレータが定義されている必要はない．プランナーに問題を与えて，もし解けなければ状態記述系列を教示することでオペレータを生成させ，後の問題にそのオペレータを用いる．このサイクルを繰り返すことで必要なオペレータを徐々に増大させることが可能となり，プランナーの設計者はプランナーに要求される問題を想定したオペレータを定義する必要がなくなる.

以下にオペレータ生成のための方法について報告する.

2　オペレータの生成

今回，STRIPSをプランナーとして用いている.STRIPSはプランナーとしてはシンプルな構成をしており，そのオペレータも簡単な構造をしている．STRIPSでは世界を表わす状態記述は述語の集合で表わされ，オペレータは条件リスト，削除リスト，追加リストと呼ばれる3つのリストから構成されている．条件リストはオペレータを作用させるための条件となる述語を表し，削除，追加リストはそれぞれ，作用させる状態から削除する述語と追加する述語を表す.

状態の記述間に成り立つオペレータを生成するということは，作用前の状態と作用後の状態からオペレータを生成するということに帰着する．つまり，作用前後の状態から，STRIPSのオペレータのための3つのリストを決定しなくてはならない．以下に各リストの生成手順を，そして図1に実際のオペレータの生成例を示す.

2.1　削除リストと追加リスト

削除リスト，追加リストはその定義から作用前後の状態記述があれば容易に求めることができる．削除リストについては，作用前の状態記述から作用後の状態記述と共通する述語を取り除くことで求めることができる．また，追加リストは逆に作用後の状態記述から，作用前の状態記述と共通する述語を取り除けばよい．この事を簡単な式に表わすと以下のような式になる.

削除リスト＝作用前の述語－作用後の述語
追加リスト＝作用後の述語－作用前の述語

2.2　条件リスト

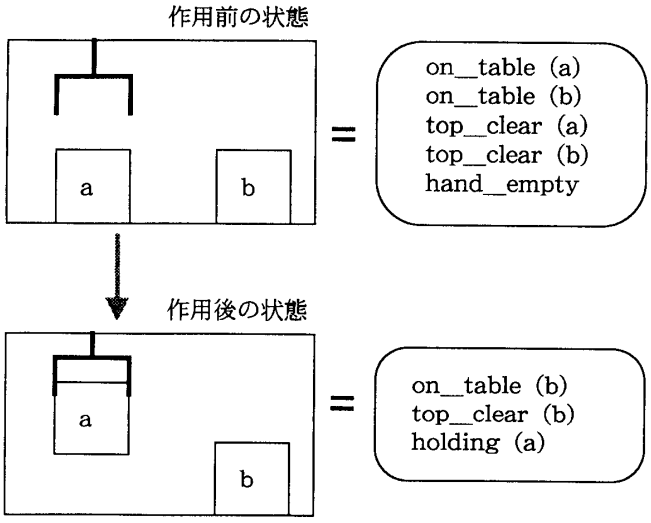
条件リストは作用前の述語の部分集合であり，どの述語を取り出せばよいかは現在のところ確たる方法はない．どのようなオペレータについても正しく条件リストを抽出できるような方法は恐らく存在しない．前提リストについては妥当と思われるヒューリスティックを用いて抽出しなくてはならない．このヒューリスティックには2通り考えられる．一つは作用前の状態記述すべてを前提リストとする方法，そしてもう一つは作用前の述語のうち，削除または追加リストに含まれる変数を持つものだけを取り出して条

Generating Operations from Scequences of State Descriptions
Mitsuo Shimohata , Seiji Yamada , Norihiro Abe , and Saburou Tsuji
Osaka University

件リストとする方法である．両者の方法は一長一短であるが，前者の方法では，本来不必要な述語まで入ってくることが多く，そのために後で述べるオペレータの統合が困難になる．また，今回はプランナーの世界として積木の世界を用いており，いくつかの例題を与えてオペレータを生成させてみた結果，両方の方法で生成させて比較すると後者の方が前者の方法より良好であったのでここでは後者の方法で条件リストを抽出する．

2．3　変数化

2．1,2．2で取り出した各リストに対し，述語の中の引数を変数に変換しなくてならない．今回の積木の例ではすべての述語のすべての引数について変数化を施しているが(完全変数化)，これは完全変数化可能なように状態述語体系を作成してあるためである．述語によっては値をそのまま残すという部分変数化と比較すると，同じ世界の記述でも体系が大きくなってしまうが柔軟性があるという点で優れている．



条件リスト　⇨ on__table (A),
　　top__clear (A),
　　hand__empty

削除リスト　⇨ on__table (A),
　　top__clear (A),
　　hand__empty

追加リスト　⇨ holding (A)

図1　オペレータの生成

3　問題点

前述したような方法でオペレータを生成した場合，様々な問題が起こる．これは条件リストが正確に求められないところに起因しており，不可避なものである．以下，そういった問題点を列挙してみる．

3．1　変化しない状態記述の排除

オペレータの作用前後で変化しない物体およびその物体に関連した述語は条件リストから削除している．これは「作用前後で状態が変化しない物体はそのオペレータに関係していることは少ない．」という考えに基づいている．しかし，作用前後で変化はしないがそのオペレータの実行に際し必要となる述語は当然存在する．(例えば，書道において字を書くというオペレータの場合，文鎮はオペレータの作用前後で変化はしないがその存在は必要である．)ただ，今回の例題で用いた積木の世界では反例が少なかったためこの方法を選択した．2．2で述べた2つの方法の優劣は与える問題の領域にも関係し，一般的にどちらがよいとはいえない．

3．2　primitiveな変化系列

システムに与える環境の変化系列はprimitiveな変化でなくてはならない．もし，あるprimitiveな変化をスキップした環境の変化系列をシステムに与えるようなことがあれば，誤ったオペレータを生成する可能性がある．．例えば，隣室へ行くという動作を教示するときにドアを開けた状態を示さなければ，壁をすりぬけて隣室へ行くという本来実行不可能なオペレータを生成してしまう．この問題についてはすでに持っているオペレータを利用することで，ある程度は対応できるのではないかと考えている．

4　まとめと今後の課題

以上，状態記述系列からオペレータを生成する方法について述べた．

今回のシステムではオペレータを生成することに重点がおいてあり，生成したオペレータはあまり有効に用いていない．オペレータの不足からプランニングが失敗した際に初期状態から手持ちのオペレータで到達できるところを示し，本当に不足しているオペレータのみを教示してもらうことができる機能も必要であろう．

また，教示されたオペレータ同士で削除リストと追加リストが等しく，条件リストも似かよっているものは本来同一のオペレータを分けて定義している可能性がある．そういったオペレータを統合して，より一般的なオペレータを再定義していくことも必要であろう．