プランニング

プランニング＝「理想的な目標を達成するための物事の間の関係を変更する行為の系列」を作成すること

プラン

STRIPS （STanford Research Institute Problem Solver）

プラン＝現在の状態を表すアサーション集合にあるアサーションを付加したり削除する行為の系列

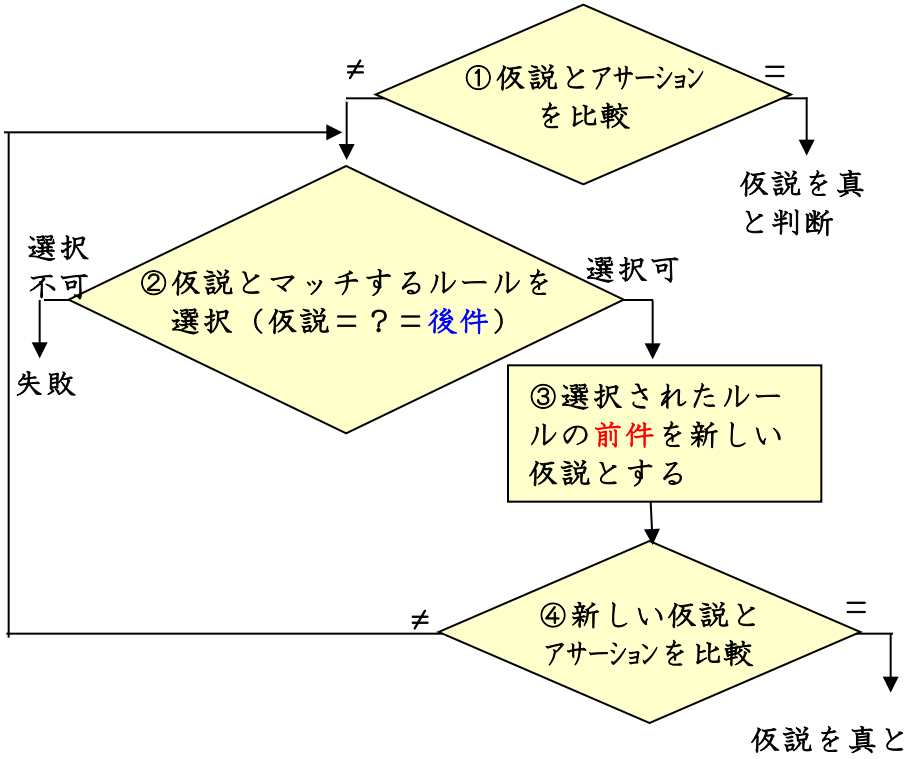
オペレータ（ルールの一種）
- 前件：IF部
- 後件：ADD部+DELETE部

プランの生成＝初期状態のアサーション集合を目標状態のアサーション集合へ変換するオペレータ系列の作成

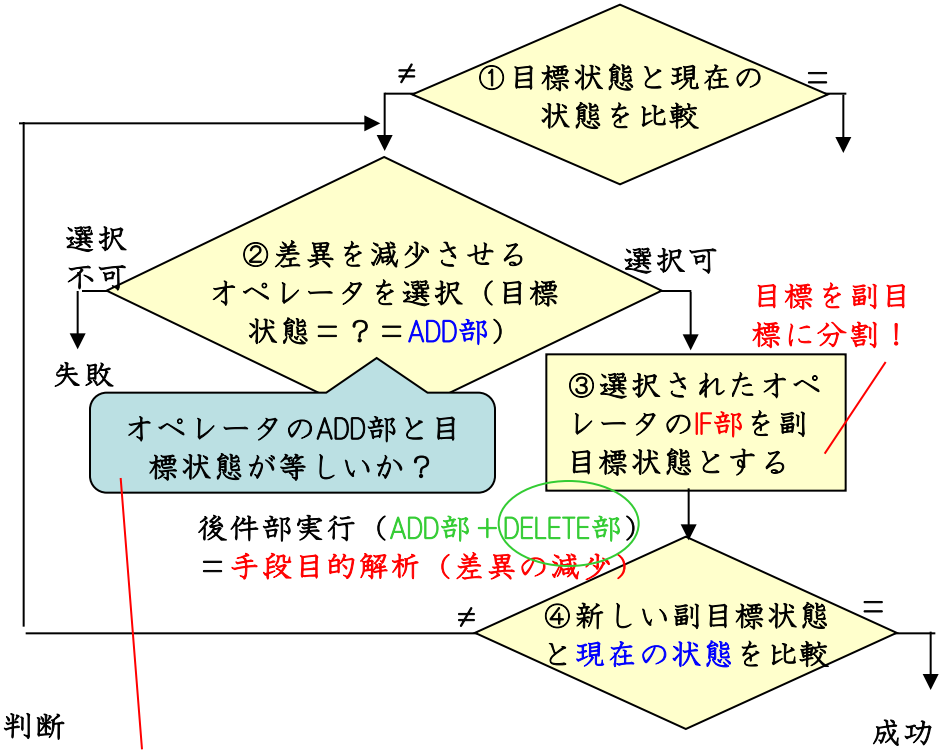
プランニング

・後ろ向き推論→目標を副目標に分解し，プランニングを再帰的に実行（問題分割）

【ルールベースシステムの後ろ向き推論】

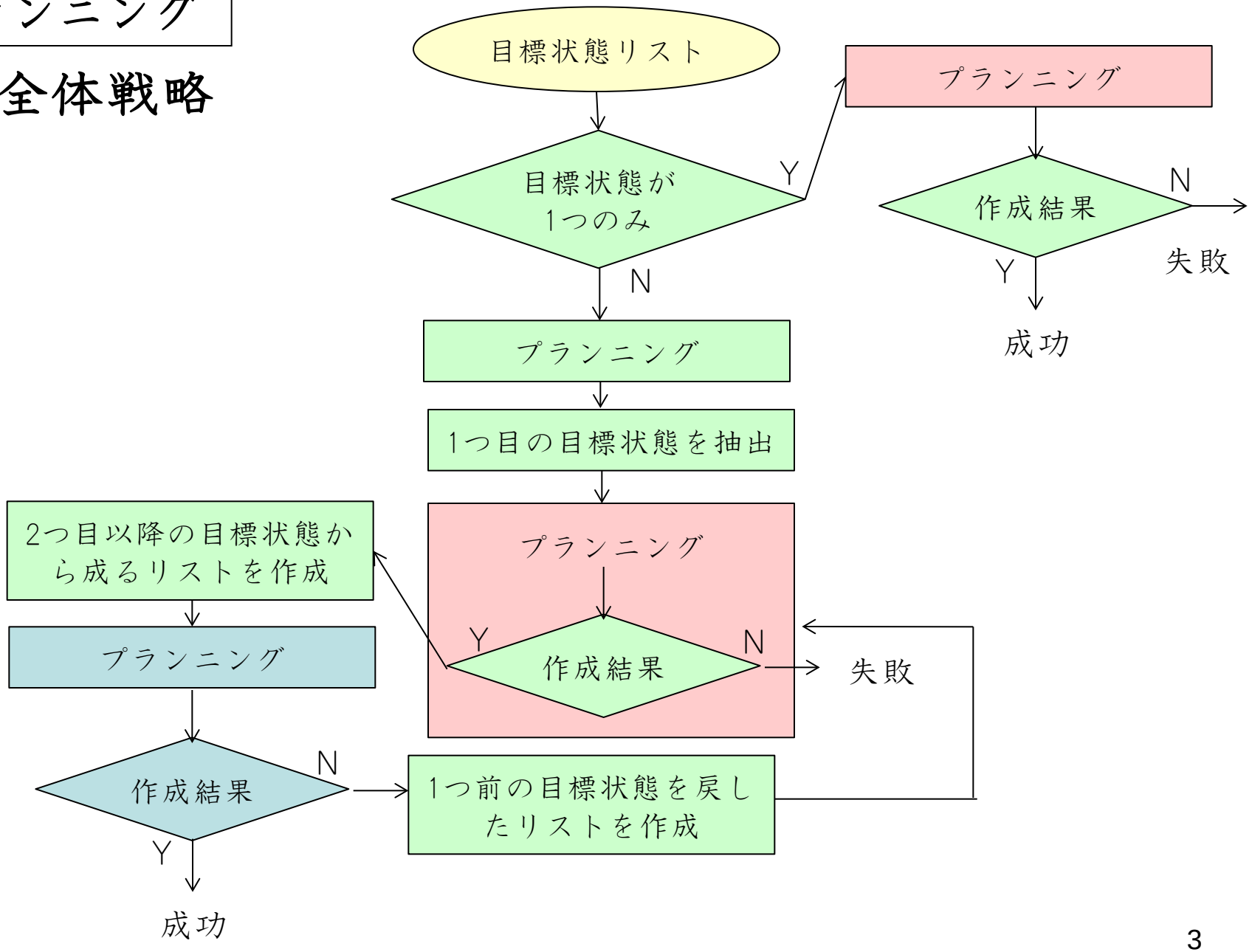


【後ろ向き推論を用いたプランニング】



プランニング

全体戦略


目標状態リスト
目標状態が
1つのみ
Y
N
プランニング
作成結果
N
失敗
Y
成功
プランニング
1つ目の目標状態を抽出
プランニング
Y
作成結果
N
失敗
2つ目以降の目標状態か
ら成るリストを作成
プランニング
作成結果
N
Y
成功
1つ前の目標状態を戻し
たリストを作成

プランニング

```
32: private boolean planning(Vector theGoalList,
33:                          Vector theCurrentState,
34:                          Hashtable theBinding){
35:  System.out.println("*** GOALS ***" + theGoalList);
36:  if(theGoalList.size() == 1){
37:   String aGoal = (String)theGoalList.elementAt(0);
38:   if(planningAGoal(aGoal,theCurrentState,theBinding,0) != -1){
39:    return true;
40:   } else {
41:    return false;
42:   }
43:  } else {
        . . . . .
99:  }
100: }
```

与えられた目標状態集合

目標状態が
１つの場合

１つの目標状態を目指したプランニング（オペレータ系列の作成）を実行

プランニング

目標状態が複数の場合

```
43:  } else {
44:    String aGoal = (String)theGoalList.elementAt(0);
45:    int cPoint = 0;
46:    while(cPoint < operators.size()){
47:     //System.out.println("cPoint:"+cPoint);
48:     // Store original binding
          （変数束縛を退避させる）
              . . . . . .
60:     int tmpPoint = planningAGoal(aGoal,theCurrentState,
                                     theBinding,cPoint);
61:     //System.out.println("tmpPoint: "+tmpPoint);
62:     if(tmpPoint != -1){
63:      theGoalList.removeElementAt(0);
64:      System.out.println(theCurrentState);
65:      if(planning(theGoalList,theCurrentState,theBinding)){
66:       //System.out.println("Success !");
67:       return true;
68:      } else {
69:       cPoint = tmpPoint;
70:       //System.out.println("Fail::"+cPoint);
71:       theGoalList.insertElementAt(aGoal,0);
72:
73:        theBinding.clear();
          //変数束縛を戻す
          . . . . . .
79:       theCurrentState.removeAllElements();
80:       for(int i = 0 ; i < orgState.size() ; i++){
81:        theCurrentState.addElement(orgState.elementAt(i));
82:       }
83:      }
84:     } else {
85:      theBinding.clear();
         //変数束縛を戻す
          . . . . . .
91:      theCurrentState.removeAllElements();
92:      for(int i = 0 ; i < orgState.size() ; i++){
93:       theCurrentState.addElement(orgState.elementAt(i));
94:      }
95:      return false;
96:     }
97:   }
98:   return false;
99:  }
100: }
```

プランニング

```
102: private int planningAGoal(String theGoal,Vector theCurrentState,
103:                           Hashtable theBinding,int cPoint){
104:  System.out.println("**"+theGoal);
105:  int size = theCurrentState.size();
106:  for(int i =  0; i < size ; i++){
107:   String aState = (String)theCurrentState.elementAt(i);
108:   if((new Unifier()).unify(theGoal,aState,theBinding)){
109:    return 0;
110:   }
111: }
112:
113: int randInt = Math.abs(rand.nextInt())%Operators.size();
114: Operator op = (Operator)operators.elementAt(randInt);
115: operators.removeElementAt(randInt);
116: operators.addElement(op);
117:
```

１つの目標と現在の状態との間でユニフィケーション実行

ある与えられた目標状態と現在の状態のどれか１つとの間でユニフィケーションが成立→planningに戻る

ユニフィケーションが成立しない→オペレータのADD部とのユニフィケーションを試みる

抜けている！

オペレータの並びをランダム化

プランニング

オペレータを１つずつ調べる

```
118:  for(int i = cPoint ; i < operators.size() ; i++){
119:   Operator anOperator = rename((Operator)operators.elementAt(i));
120:   // 現在のCurrent state, Binding, planをbackup
              . . . . . .
136:   Vector addList = (Vector)anOperator.getAddList();
137:   for(int j = 0 ; j < addList.size() ; j++){
138:    if((new Unifier()).unify(theGoal,
139:                             (String)addList.elementAt(j),
140:                             theBinding)){
141:     Operator newOperator = anOperator.instantiate(theBinding);
142:     Vector newGoals = (Vector)newOperator.getIfList();
143:     System.out.println(newOperator.name);
144:     if(planning(newGoals,theCurrentState,theBinding)){
145:      System.out.println(newOperator.name);
146:      plan.addElement(newOperator);
147:      theCurrentState =
148:       newOperator.applyState(theCurrentState);
149:      return i+1;
150:     } else {
```

ADD部のアサーションを１つずつ調べる

```
151:      // 失敗したら元に戻す.
152:      theBinding.clear();
153:      for(Enumeration e=orgBinding.keys();e.hasMoreElements();){
154:       String key = (String)e.nextElement();
155:       String value = (String)orgBinding.get(key);
156:       theBinding.put(key,value);
157:      }
158:      theCurrentState.removeAllElements();
159:      for(int k = 0 ; k < orgState.size() ; k++){
160:       theCurrentState.addElement(orgState.elementAt(k));
161:      }
162:      plan.removeAllElements();
163:      for(int k = 0 ; k < orgPlan.size() ; k++){
164:       plan.addElement(orgPlan.elementAt(k));
165:      }
166:     }
167:    }
168:   }
169:  }
170:  return -1;
171: }
```

変数束縛結果を用いてそのオペレータのIF部アサーション群を新しい目標状態群とする

新しい目標状態を目指してプランニングを実行

成功したら作成したオペレータをプランに追加．オペレータを実行（前向き推論的）．

目標状態とADDのあるアサーションとの間でユニフィケーションが成功

プランニング

与えられた状態にオペレータを実行して，
その状態にADD部の実行結果を追加し，
その状態からDELETE部の実行結果を削除．

メソッド　applyState

```
319: public Vector applyState(Vector theState){
320:   for(int i = 0 ; i < addList.size() ; i++){
321:     theState.addElement(addList.elementAt(i));
322:   }
323:   for(int i = 0 ; i < deleteList.size() ; i++){
324:     theState.removeElement(deleteList.elementAt(i));
325:   }
326:   return theState;
327: }
```

与えられた状態

状態にADDリストを追加

状態からDELETEリストを削除

更新された状態

プランニング

planningに入る．

初めの目標状態を選びplanningAGoalに入る．

オペレータ（OP)のADD部にある”?x on ?y”をみて，そのOPを具体化．左記は当該OP名．

当該OPのIF部を新たな副目標状態に追加．

現在の状態

OPのADD部をみてそのOPを具体化．

当該OPのIF部を検証（整合）

IF部が満足→OPをプランに追加

1つ前のOPに戻る

実行

ontable Bがなくなっている

```
% java Planner
*** GOALS ***[B on C, A on B]
**B on C
Place B on C
*** GOALS ***[clear C, holding B]
**clear C
[clear A, clear B, clear C, ontable A, ontable B, ontable C, handEmpty]
*** GOALS ***[holding B]
**holding B
pick up B from the table
*** GOALS ***[ontable B, clear B, handEmpty]
**ontable B
[clear A, clear B, clear C, ontable A, ontable B, ontable C, handEmpty]
*** GOALS ***[clear B, handEmpty]
**clear B
[clear A, clear B, clear C, ontable A, ontable B, ontable C, handEmpty]
*** GOALS ***[handEmpty]
**handEmpty
pick up B from the table
Place B on C
[clear A, ontable A, ontable C, B on C, clear B, handEmpty]
*** GOALS ***[A on B]
**A on B
```

```
Place A on B
*** GOALS ***[clear B, holding A]
**clear B
[clear A, ontable A, ontable C, B on C, clear B, handEmpty]
*** GOALS ***[holding A]
**holding A
pick up A from the table
*** GOALS ***[ontable A, clear A, handEmpty]
**ontable A
[clear A, ontable A, ontable C, B on C, clear B, handEmpty]
*** GOALS ***[clear A, handEmpty]
**clear A
[clear A, ontable A, ontable C, B on C, clear B, handEmpty]
*** GOALS ***[handEmpty]
**handEmpty
pick up A from the table
Place A on B
***** This is a plan! *****
pick up B from the table
Place B on C
pick up A from the table
Place A on B
```