Classical Planning:

Compilation of notes from a
Seminar course held at ASU in Spring 93

by

Subbarao Kambhampati

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406

Working Notes, ASU-CS-TR 94-008

(Please send mail to rao@asuvax.asu.edu, if you
retrieve this document. Thanks.)

The following is an unedited, and unpolished compilation of notes from a planning seminar that I ran at Arizona State University in the spring of 94. The seminar turned out to be mostly about classical planning techniques. In each class meeting, a designated student took notes and mailed them to the class. A compilation of these notes appears below. Notes from spring of 93 are also available in the ftp site at enws318.eas.asu.edu:pub/rao. A couple of times, interesting mails from outside colleagues were cc'd to the list.

Subbarao Kambhampati
[Jun 22, 1994]

From daemon Wed Jan 26 11:44:36 MST 1994
Return-Path: huson@enuxsa.eas.asu.edu
Return-Path: <huson@enuxsa.eas.asu.edu>
Received:
From asuvax.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA10557; Wed, 26 Jan 94 11:44:34 MST
Received:
From parikalpik.EAS.ASU.EDU by asuvax.eas.asu.edu with SMTP id AA10488
(5.65c/IDA-1.4.4 for rao@parikalpik.eas.asu.edu); Wed, 26 Jan 1994 11:40:29 -0700
Received:
From enuxsa.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA10554; Wed, 26 Jan 94 11:44:29 MST
Received:
From localhost (huson@localhost) by enuxsa.eas.asu.edu (8.6.4/8.6.4) id LAA16206
for plan-class@enws228.eas.asu.edu; Wed, 26 Jan 1994 11:44:08 -0700
Message-Id: <199401261844.LAA16206@enuxsa.eas.asu.edu>
X-Mailer: ELM [version 2.4 PL23]
Content-Type: text
Content-Length: 12506
Status: RO
From: Mark Huson <huson@enuxsa.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu
Subject: Notes for 24 Jan 94
Date: Wed, 26 Jan 1994 11:44:07 -0700 (MST)

Notes for 24 Jan 94 Mark L. Huson (huson@asu.edu)

Agenda
- State Space Planners
  - ProgWS
  - RegWS
- Plan Space (Partial Order) Planners

Article: "An Introduction to Partial Order Planning", Weld

Given the general problem of planning, are there better ways of modeling the problem as a search?

State Space: start with an initial state and produce all possible subsequent states by applying "applicable" actions, and repeat this process for the states generated until a (the) goal state is found.
Plan Space: start with a null plan, and add those actions which achieve a selected subgoal, repeat until all actions required to achieve the goal state have been identified and, where necessary, ordered.

-----------Planning as a Search in the Space of World States

The search space can be represented by a connectivity graph with a distinguished node representing the start state for the search. All states are connected by arcs representing atomic actions. There are two primary methods of performing search, forward search which using the initial world state as the start node for its graph, and backward search which uses the goal state.

*** see Figure 3 for a State Space connectivity graph ***

For the search problem we must have a way to describe the states. For blocks world we may have an initial state such as:

```
    on(A, Tab)     clr(A)               _      _       _
    on(B, Tab)     clr(B)             |A|    |B|     |C|
    on(C, Tab)     clr(C)       --------------------------
```

If our goal state is a conjunction { on(A, Tab) /\ clr(C) } then each time we attempt to create new states by applying actions, we first check to see if we have matched the goal state conditions. Goal states can be defined either as a fully specified state, or as an equivalence class of foal states.

```
  Equivalence Class:    on(A, B) /\ on(B, C)
  Fully specified  :    on(A, B) /\ on(B, C) /\ on(C, Tab) /\
                           clr(A)
```

We also need a set of actions. These actions should contain the following information: an identification; a set of preconditions identifying states where the action is applicable; a set of effects describing the results of applying the action.
For example:

```
 identification  -   move (A-Tab-B)
 preconditions   -   on(A, Tab) /\ clr(A) /\ clr(B)
 effects         -   on(A, B) /\ ~on(A, Tab) /\ ~clr(B)
```

There are both positive and negative effects for most actions, indicating the conditions in the new state. The new state is described by the conditions of the old state, conjoined with the effects. Contradictions are resolved by setting the new states conditions to the effects.

The STRIPS assumptions is that everything changed by an action is explicitly represented in the effects list of the action.

For the ProgWS (progression world state) Algorithm, we are performing a Forward search in the state space. The forward search proceeds by selecting a state, then applying all applicable actions (those whose preconditions are met) and adding these new states to the set of states to be considered. For non-trivial domains there are many more applicable actions than there are useful actions for achieving the goal state. For this reason, forward (progression) searches usually have a large branching factor. If we know ahead of time which branch leads to the solution, then forward search is as good as any other. Typically this type of control information is not available, and since strategies are ranked according to the worst case, forward search generates a lot of useless states and is therefore seldom the 'best' strategy (though counter examples can be constructed.

As an alternative, we can start
From the goal and work backwards,
which is backwards or regression search. In this case we use the "backwards" application of the actions to arrive at the previous state. In this way the plan is generated

From the last
step to the first step. To arrive at the plan
From a forward
search, the actions are taken in order along the path
From the
start node to the final node. For backward search, these actions
taken in reverse order comprise the plan.

The state description for a state generated in a regression search is obtained in much the same way as for a forward search. There are two main differences. First, the new state must satisfy the preconditions of the action, and therefore the state conditions are a conjunction of the old node’s conditions, the negation of the effects for the action, and any preconditions not expressly mentioned in the effects (just as in forward search the effects will be taken for opposites). The second difference is that a goal state which is not completely specified may result in an equivalence class of ’previous’ states being identified.

For example:

```
          Start node           Goal = on(A, B) /\ on(B, C)
                                 |
                                 |
          --------------------------------------------------
          |                             |
          | move (A, Tab, B)            |
          |   pre ={ clr(B)             |
          |          on(A, Tab)         |    {THE OTHER ACTIONS}
          |          clr(A) }           |
          |   eff ={ on(A, B)
          |          ~on(A, Tab)
          |          ~clr(B) }
          |
          |
   Prev = on(A, Tab) /\
       on(B, C) /\ clr(B) /\
            clr(A)
```

When a state is selected it is checked to see if its conditions are a subset of the conditions for the initial state. If so the search is done. There is one other problem with backwards search. Suppose we have a goal state defined as

G = on(A, B) /\ clr(B)

this results in an undefined state during the search. Such an undefined state and similar problems occur if we attempt to pursue the path in the problem above which starts with the action

move (B, Tab, C)

The effects cause us to add ’clr(B)’ as a condition for the new state, but we also have ’on(A, B)’ carried over
From the start
node. To prevent attempts to search this branch, the delete list (negated items) in the effects are checked, and any action which deletes anything in the state is not used to generate a new state. While a particular state may still end up with a contradiction, any search of the path will fail when no action can be applied.

The reason for using a backward search is to focus the selection of actions to those actions which are *useful* rather than those which are *applicable* which are used in the forward search. The idea being that there are fewer useful actions (those which help produce a goal) than applicable, and the branching factor for the search is therefore reduced.

State Space Searches both construct a plan by planning in the same order as execution. With a forward planner, a prefix for a plan is constructed, and the rest of the plan is concatenated to the end.

```
     |-----plan prefix-----| . . . rest of plan . . . goal state
```

Backward planners construct the plan in reverse order, by creating a suffix and prepending the rest of the plan.

```
     initial state . . . rest of plan . . . |----plan suffix---|
```

Because the order of plan execution is driven by the order of the plan generation, backward searches result in many branches which lead to states where no action can be applied. The forward planning equivalent to this is a non-minimal plan, where a state occurs twice in a plan.

We would like to deal with goals in an arbitrary order, rather being forced to consider them in the (reverse) order we achieve them. This is not possible in State Space planning because the order considered in planning is the same as the order of execution. We need to separate Planning order
From Execution
order.

-----------Searching in the Space of Plans

The main motivation for Plan Space Search is we can avoid back- tracking because we look at goals in an order different
From
execution order.

Each search node represents a partial plan, with a set, A, of actions and an ordering relation, O.

<A, O>

An plan which executes the actions according to the ordering O is legal.

For example, to solve the problem

```
Initial state:                       Goal state:
    on(A, Tab)     clr(A)                on (A, B) /\ on (B, C)
    on(B, Tab)     clr(B)
    on(C, Tab)     clr(C)
```

We can have the following results:

```
    A = { move (A, Tab, B)             O = { A2 < A1 }
          move (B, Tab, C) }
```

Here the order relation A2<A1 indicates action A2 precedes A1.

Two additional dummy actions are added, A0 and Ainfinity.
These are used to indicate the first action (*start*) and last action (*end*) of the plan. All other actions are preceded by A0 and followed by Ainfinity.
For example

```
        A = { A1, A2 } and O = { A0<A1 A0<A2 A1<Ainf A2<Ainf }
```

This represents an unordered plan with actions A1 and A2 (i.e., the actions are independent and can be applied in any order).
Graphically,

```
                ------ A1 ------
               /                \
              /                  \
            A0                    Ainf
              \                  /
               \                /
                ------ A2 ------
```

Here we have a partial order between the actions, where all actions are ordered with respect to A0 and Ainf, but may or may not be ordered wrt other actions. With this representation as an action enters a plan, it is checked to see if we can leave them unordered. By analyzing the results of adding the action, required orderings can be provided. This analysis relies on maintaining a set of "causal links". These links identify actions which provide the preconditions for other actions.

For this processing, planning continues (adding actions and order relations) until all goal conditions have been worked on. Actions are chosen such that they satisfy a goal. The overall goals of the plan are set as preconditions for Ainf

```
                on(A, B) @ Ainf                 A
                on(B, C) @ Ainf                 B
                                            ____C____
```

In addition, all initial conditions are considered effects of A0. The set of goals then are added to an "agenda", which is a list of goals which still need to be satisfied (planned for). Actions are added which make a goal on the agenda true. Causal links show which actions provide the preconditions for other actions. When an action is added to the plan, its preconditions are added to the agenda.

The order analysis makes it desirable that nothing comes between an action and the action which has its effects as a precondition.

For example, return to the problem

```
Initial state:                      Goal state:
    on(A, Tab)     clr(A)               on (A, B) /\ on (B, C)
    on(B, Tab)     clr(B)
    on(C, Tab)     clr(C)
```

We have an agenda

```
    { on(A, B) @ Ainf
      on(B, C) @ Ainf}
```

Suppose we work on "on (B, C) @ Ainf" first. We choose the A1 as the action "move (B, Tab, C)", and the resulting causal links and agenda are:

```
     Causal Links                   Agenda
                                    { on (A, B)   @ Ainf
        A0 ------------------------>  clr (B)     @ A1
        A0 ------------------------>  on (B, Tab) @ A1
        A0 ------------------------>  clr (C)     @ A1 }
```

Three of the goals are satisfied by action A0, so now we look at the goal "on (A, B) @ Ainf". We now choose action A2 to be "move (A, Tab, B)", but we notice one of the effects is ~clr(B). Since A0 provides clr(B) as a precondition to A1, there is an ordering requirement. We can either place A2 as an action before A0 (which is not valid according to our model), or we can place the action between A1 and Ainf. The resulting plan is:

```
    A = { A1 - move (B, Tab, C)         O = { A0<A1 A0<A2 A1<Ainf
          A2 - move (A, Tab, B) }               A2<Ainf   A1<A2 }
```

where the last order relation is the result of the analysis of the causal links. Except for causal requirements, we leave actions unordered.

*********end of notes for 24 Jan 94*********

```
From rao  Wed Jan 26 22:49:30 1994
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11396; Wed, 26 Jan 94 22:49:30 MST
Message-Id: <9401270549.AA11396@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Teasers
Date: Wed, 26 Jan 94 22:49:30 MST
Reply-To: rao@asuvax.asu.edu
```

0. Mark set a rather high standard for note-taking, with his first set of notes. I hope those who follow will exceed it ;-)

1. Here are a couple of things to think about before next class:

[A.] When we talked about variables at the end of the class today, I said that variable constraints also have to be maintained, and their consistency checked just as we did for ordering constraints.

A set of binding constraints of the form (x=y, y=z, z=A, y!=B) etc are said to be inconsistent if and only if we can derive a binding as well as its negation
From the constraits (e.g., x=A and x!=A).

It turns out that the cost of checking the consistency of a set of binding constraints depends on whether you the domain of the varibales under (i.e., the set of objects which can be the possible values of a variable) consideration is infinite or finite. It is polynomial

(O(n^3)) for one case and NP-hard (i.e., all known algorithms take exponential time) for the other. Can you guess (a) which case is which? and (b) why does the difference come about?

[B] When we talked about causal links today, we said that a causal link is said to be violated if and only if a step can intervene its producer and consumer and DELETE the condition being supported.

Can you think of any advantages of weakening the definition to say that a causal link is thereatened if a step can intervene and either DELETE or ADD the condition?

Can you think of any advantages/disadvantages of strengthening the definition and say that a causal link is violated if a step v intervenes and DELETES the condition, and no step that necessarily comes after v and before the consumer adds the condition back?

[C] Can you think of an example planning problem where you can clearly see that the order in which the goals are worked on drastically changes the search-space size?

Please feel free to email your thoughts to plan-class

Rao

```
From rus@seine.eas.asu.edu  Mon Jan 31 14:15:12 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
Status: RO
Return-Path: <rus@seine.eas.asu.edu>
Received:
From seine.eas.asu.edu (enws293.EAS.ASU.EDU) by parikalpik.eas.asu.edu (4.1/SMI-
4.1)
        id AA17035; Mon, 31 Jan 94 14:15:12 MST
Received: by seine.eas.asu.edu (4.1/SMI-4.1)
        id AA05885; Mon, 31 Jan 94 14:24:19 MST
Message-Id: <9401312124.AA05885@seine.eas.asu.edu>
Content-Type: X-sun-attachment
From: rus@seine.eas.asu.edu (Rus Ioana)
To: plan-class@parikalpik.eas.asu.edu
Subject: Seminar notes
Date: Mon, 31 Jan 94 14:24:19 MST

----------
X-Sun-Data-Type: text
X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 231
```

***************Notes for 26 Jan 1994   Ioana Rus***********

Agenda
- POP
- POP + variables
- POP + Conditions Effects   (*)
- POP + Disjunctive Preconditions   (*)
- UCPOP (Quantified Effects)   (*)

(*) = not covered

Article: "An Introduction to Partial Order Planning", Weld

Last class: why to change search
From plan space to world state space.
One answer: flexibility in plan space planning-keeping partial plans allows
to add an action between any other actions.

A possible partial plan:

```
                           A1--------A3
     START---------- /              \--------END
                         \-----A2------/
```

where START and END are dummy actions

In the final plan

```
        START----A5----A1---A6---A3----A2-----END
```

the actions
From the partial plan and their order is preserved.

REFINING planning = adding constraints

POP algorithm

A plan may be described by a set of actions A and a set of orderings among the actions, O

< A, O>

such as:

```
        A = { A1, A2, A3 }
        O = { START < A1 < A3 < END , START < A2 < END }
```

```
Assume we have the planning problem given by:
        the initial state I = {I1}
        the goal state    G = {P,Q}     where I1, P, Q are predicates
and the following possible actions:
        A1      precond.: (I1)
                effects: adds P
        A2      precond.: ()
                effects: adds Q, deletes I1
        A3      precond.: ()
                effects: adds I1
```

Description of initial state:

< {START, END}, {START < END} >

agenda (set of goals): { P@END, Q@END }

```
Goal = condition that has to be true at a state
Top-level-goal = condition that has to true at the end

Pick a goal
From the agenda and see what action can provide it.
Action A1 can provide goal P to END.
             P
        A1----------> END
At each step:- every precondition of the action added to the plan is
                introduced in the agenda,
             - every goal on which we work is deleted
From the agenda
the agenda becomes:
        { Q@END, I1@A1 }
the partial plan is:
                I1         P
        START -------->A1--------> END

the causal links between the actions must also be represented,
so the description of the planner will contain the links, as well:

        < A, O, L>

where   A = {START, END, A1}
        O = {START < A1 , END}
                  P                  I1
        L = { A1 ----> END , START -------> A1}


Another possible partial plan is:

                             I1            P
        START -------> A3 ---------> A1 -------> END

for the moment cannot choose between the two partial plans, so search will
have to be performed, in the space of plans.

For the other goal at END, Q there is only action A2 that can provide it.

               Q
        A2 ---------> END

The description of the new partial plan will be:

        A = {START, END, A1, A2}
        O = {START < A1 < END, START < A2 , END}
                     I1             P              Q
        L = {START -------> A1, A1 -----> END, A2 -----> END}


Links describe commitments in the plan.

When introduce a new step in the plan, must check if the action introduced
threatens the links (comes between the producer and consumer of the link
and deletes what the producer provides.)

A2 can come between START and A1, but  the link between START and A1 is
threatened by A2, so A2 must be before START, or after A1. In this case we
know that no action can come before START, and A2 must be after A1, but
in general some tests must be performed, so the ramification occurs again.
```

```
A new order will be add to O : A1 < A2

Now the agende is empty, so the plan is complete. It is:

        START ------> A1 -----> A2 -----> END

The termination conditions for the algorithm are :
        - the agenda is empty
        - there are no conflicts (threats).
This final plan can be viewed as the "plan" or a set of possible plans
( any plan that respects these constraints is a possible final plan).

IMPLEMENTATION
To verify and avoid conflicts and establish the order among actions,
the transitive closure of the partial ordering graph has to be built.
                             d
The search performance is O(b ).
        b (the branching factor) is affected by:
          - how establish the choice for the source of a precondition
                b = n + m, where n is the number of steps in the partial plan
                                 m is the number of possible actions
          - how many possibilities to solve conflict resolutions there can be

        d (depth of graph) is affected by:
          - number of conditions in the plan

        P * n + n^3   P=  max. number of preconditions for the actions
                 |
                 |---> conflict resolution, depending on the number of
                        threats

World state space planners have d ~ P * n, but b is much greater
than for planning in plan space.

COMPLETENESS - is ensured by POP
EFFICIENCY - influenced by the order of working on goals in the agenda.


EXTENTIONS OF POP

                POP algorithm with variables
                ----------------------------

If work with variables (instead of working only with constants), an action
could look like:
        move x, y, z    - move x
From y to z

        x cannot be 'table'

and if the preconditions for this action are : cl(x),cl(z), on (x,y)
and its effects: on (x,z), not cl(z), not on (x,y), cl(y)

then z cannot be 'table'.

the action      move-to-table (x,y) will have
        preconditions: cl(x), on (x,y)
        effects: on (x, table), cl(y), not on (x,y)

When parametrizing for allowing variables to be introduced, must be sure to
restrict their domain.
```

When pick an action to be added to the partial plan, so that it will provide one of the goals in the agenda, if variables are used, instead of verifying equalities, unification should be verified.
For example, if the agenda contains   on (a,b)@END, and there is an action move (x,y,z) that has as effect on (x,y), then this action can provide END with the goal on (a,b), if x=a and y=b. (unification). There is no binding for y=> 'least commitment' = not to impose restrictions if don't need them.
The bindings introduce new constraints, so the representation of the plan should include now these bindings.
The plan representation will be given by:

        < A, O, L, B>   where B is the bindings set.

Bindings have to be done at - establishment of an action to be added
                            - conflict resolution.
The verification of conflicts implies also unification, instead of equality.

Example: consider a partial plan

```
                                -----
                               |START|
                                -----
                                 cl(b)
                                  |
                                  |
                               cl(x)
        ----------------                ------------
        |move (x',y',z')|               |move(x,y,z)|
        ----------------                ------------
        on (x',z'), not cl(z')          on (x,z)


                on (a,b) on (b,c)
                        ----
                        |END|
                        -----

        bindings: x=b, z=c, x'=a, z'=b
```

the action move (x,y,z) with x=b and z=c provides on (b,c)@END, but needs as precondition cl(x), that means cl(b). cl(b) is provided by START, but action move (x',y',z') with x'=a and z'=b, which can provide on(a, b)@END deletes cl(z'
),
that is cl(b), so there is a conflict!!!

        ************* end of notes for 26 Jan *********

From rao Tue Feb 1 15:40:59 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
X-VM-v5-Data: ([nil nil nil nil nil nil nil t nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA19223; Tue, 1 Feb 94 15:40:59 MST
Message-Id: <9402012240.AA19223@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Counter examples needed
Date: Tue, 1 Feb 94 15:40:59 MST
Reply-To: rao@asuvax.asu.edu

This is just to remind you that we are on lookout for counter examples for

1. showing that not using confrontation will lead to loss of completeness (when we have operators with condiditional effects)

2. showing that treating disjunctive preconditions by making a nondeterministic choice on the disjunct could lead to losss of completeness for some times (in this case, you should also come up with a reasonable assumption on the planning problem that will rule-out the counter example).

Rao

From plan-cla@enws318.eas.asu.edu Thu Feb 3 17:31:16 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <plan-cla@enws318.eas.asu.edu>
Received:
From enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA22331; Thu, 3 Feb 94 17:31:16 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA27846; Thu, 3 Feb 94 17:29:21 MST
Date: Thu, 3 Feb 94 17:29:21 MST
From: plan-cla@enws318.eas.asu.edu (Subbarao Kambhampati)
Message-Id: <9402040029.AA27846@enws318.eas.asu.edu>
To: plan-class@enws228
Cc: rao@enws318.eas.asu.edu
Subject: Instructions for Using UCPOP

[[THIS MAIL IS FAIRLY LONG. YOU MAY WANT TO PRINT IT AND READ IT]]

Dear Planning class attendees:

A general account for planning seminar folks has now been set up on my research machines.

machine: enws323.eas.asu.edu
Account: plan-cla
Password: (please send me mail individually and I will give you the password)

From rao Thu Feb 3 18:02:29 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA22391; Thu, 3 Feb 94 18:02:29 MST
Message-Id: <9402040102.AA22391@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)

To: plan-class
Subject: UCPOP ASSIGNMENT (*important*)
Date: Thu, 3 Feb 94 18:02:29 MST
Reply-To: rao@asuvax.asu.edu

The following are two simple ucpop assignments, with due dates.

The purpose of these is to give you a chance to see the planner in action and write some domains for it. The assignments are for your learning. They will not be graded.

I would STRONGLY encourage everyone attending the class to do it.

[Obligatory threats:

If you are registered for credit, you *better* do it or else...

If you are not registered, I cannot force you to do it. But if you don't, I will be much less willing to take your questions in the class seriously!]

Since the plan-cla account is a general account, you may want to create a subdirectory with your name and keep your code/dribble files etc in it.

****Assignment 1. (childs play)
Due Date: 9th February (Submission by electronic mail)

Start the ucpop system, familiarize yourself with its interface.

1. Run a couple of pre-defined problems
From a couple of domains
2. Define a couple of hard problems for some of the domains in the domains.lisp file. Run them and see how long UCPOP takes.

3. Write a new ranking metric for UCPOP and load it into the ucpop system
Compare its performance with the other metrics.

Deliverable: An electronic mail (to plan-class list) with annotated,
short and to the point dribble files
From the various sessions.
[REMEMBER NOT TO STORE VERY LARGE DRIBBLE FILES IN THE ACCOUNT. THERE IS SPACE SHORTAGE ON THAT MACHINE]

****Assignment 2. (warming up) [You should start work on assignment 2 simultaneously with assignment 1 so that you have enough time to come up with, and write down the specification of, the domain.]

Due Date: 16th February (Submission by electronic mail)

Write a non-trivial domain of your choice for UCPOP. The domain must not be one of the ones present in the domains.lisp file. (Aravind: You can try encoding a toy version of process planning, for example).

Experiment with problems
From the domain. Try to direct the planner with
the help of any domain-specific ranking function.

Deliverable: You should leave a file with your domain specification in the plan-cla directory (You may want to make a subdirectory under your name and keep the file in there.)

You should send a mail to plan-class summarizing your successes and failures in specifying the domain, and being able to run non-trivial problems in the domain, as well as any nifty ranking functions with which you had good success.

That is it. Get busy. If you have problems interpreting the assignment or getting started on it, please feel free to see me.

Rao
[Feb 3, 1994]

From yqu@enuxsa.eas.asu.edu Sun Feb 6 14:12:43 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <yqu@enuxsa.eas.asu.edu>
Received:
From enuxsa.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA25137; Sun, 6 Feb 94 14:12:43 MST
Received:
From localhost (yqu@localhost) by enuxsa.eas.asu.edu (8.6.4/8.6.4) id OAA24524 for plan-class@enws228.eas.asu.edu; Sun, 6 Feb 1994 14:12:21 -0700
Message-Id: <199402062112.OAA24524@enuxsa.eas.asu.edu>
X-Mailer: ELM [version 2.4 PL23]
Content-Type: text
Content-Length: 13541
From: Yong Qu <yqu@enuxsa.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu
Subject: Jan 31. note
Date: Sun, 6 Feb 1994 14:12:20 -0700 (MST)

********   Class note for January 31, 1994   ******************
Planning seminar

Agenda
    Variables
    Conditional effects
    Disjunctive preconditions
    Universial quantities
    UCPOP

Review:
    In last class, we have learned that compared with POP, UCPOP has some new features: conditional effects, disjunctive precondition and univeral quantities.
    UCPOP is sound and complete. That means if there is a solution to a plan problem, UCPOP can always find that. but UCPOP can't garantee to find the solution in reasonable time: effeciency is always a big problem. And some method are used to improved effeciency.

In Today:

1. Variables

Upto now, all the actions in POP are instantiated actions, like move-A-B-C means move block A From B to C. Then we need a whole bench of actions to represent all the actions of the world.

Now we can use one parameterized action instead of several instantiated actions.

For example, in the blocks world, the action move can be instantiated to several actions, each correspondence to a block. To simplizied that, we can introduced a parameterized action move(x,y,z)

```
   operator move(x,y,z)
   :precondition cl(x)^cl(z)^on(x,y)
   :effect !cl(z)^cl(y)^on(x,z)
```

It's fairly clear that the operator move(x,y,z) is a much more economical descritption than the fully specified move actions it replaces. In addition, the abstract description has enormous software engineering benefits -- needless duplication would likely lead to inconsistent domain definitions if an error in one copy was replaced but other copies were mistakenly left unchanged.

But now each partial plan is a four - tuple:

P=<S,O,B,L>

where S stands for Steps, O for order, B for binding, and L for links.

We should change POP is the following parts of POP:

a. Establishment: introduce an action.

b. Threat detection

ex. in block world , there is one action move(x,y,z)

```
     :precondition clear(x)^clear(z)^on(x,y)
     :effect !on(x,y)^ on(x,z)
  initial state: cl(A) cl(B) cl(C)
  goal state : on(A,B) on(B,C)
```

```
                                ____________
                               |   Start    |
                               --------------
                                       cl(B)
                                         |
                                       cl(x)
                 ______________     _____________
        S2    |move(x',y',z')|     | move(x,y,z) | S1
              ---------------      ---------------
              !cl(z') on(x',z')        on(x,z)
                             |         |
                             |         |
                             |         |
                             |         |
                           on(A,B)on(B,C)
                             _______________
                            |    end        |
                            ________________
```

We use step S1 to resolve on(B,C) and S2 to resolve on(A,B)

Here,

S: start, S1,S2, end

```
O:
        /-------S1-------\
   start                   end
        \-------S2-------/

B: {x=B, z=C, x'=A z'=B}
```

```
         cl(B)
L: start--------->S1
```

here, S1 is to move B From someplace to on top of C.

yet in S2 there is a effect of !cl(z'). and, unfortunately, z' is binded to B, so this might be a threat:

```
  S2    --> !cl(B)
  cl(B) -->  S1     : it's a threat.
```

Here, we give the definition of a threat:

Def. A THREAT is any step which intervene the producer and consumer of a action and necessary delete the condition.

If there is a threat, we must resolve it.

If a new action is a threat to an existed casual link, there are two methods to resolve the threat:

a. Promotion: put the new action in front of the producer, or

b. Demotion : put the new action after the consumer.

In this example, S2 is a threat to start-->S1 casual link, then we can either put S2 in front of start (promotion), or put S2 after S1 (demotion). In this example, we know that it's impossible to put S2 in front of start, and we can cut out this branch, yet in ordinally case, both two cases should be considered.

Problem: how to detect a confliction among casual links?

Solution: If there is a circle in the constraints, then we should stop this branch and cut it off.

We have assumed that each new steps should be between the dummy steps start and end, so to every step Si, we got

start<Si<end

In this example, we have start<S1<end.

in case of promotion, we want to put S2 in front of start, that means we need to add a new constraint S2<start.

Yet it's default that start<S2<end. So there is a circle exist start<S2<start. So we prune this branch.

In UCPOP, new constraints can only be added in, they will never be removed. New steps, add new constraints, are added to resolve the threats existed. so there is one problem: do we need the "white knight", that is, if there is a casual link, for example,

```
   +P
S1---->S2,
```

can we add some steps between S1 and S2, Si..Sj, such that P is deleted and than added again, (Si will delete P and Sj will add P)?

Well, the answer is no. Because we don't have to deal with it. UCPOP is sound and completeness, we can find every solution to a problem, if it exists. The situation above will be explored by another plan in UCPOP, without having to use "white knight". We can simplify our planning if we don't deal with white knight, otherwise, the branching factor will be much larger. (To delete a casual link, we need to add in some more steps, and to resolve the new casual links, more and more steps need to be introduced..., the branching factor is much larger.)

Another problem: how can we know that a solution has been found?

In a ground linearization, if there is nothing in the agenda, and there is not threat, then a unique sequence , that is , a solution, has been found.

Now the way we make a planning is different From taking a plan and check if it's correct. In some sense, me "make" the current partial correct. All the constraints are kept, and we must make sure that all the link constraints are safe. Nothing will come between with the casual link and delete. When all the ground linearization are correct and nothing left on age

```
nda, then a total is got.

ex1. In the following chart, We want to achieve P,Q,R at goal. At the present pa
rtial plan, A2 gives P, A4 gives Q, and no other actions will delete them. so un
der whatever kind of action sequence, P and Q will always be true.
    But we can't garantee that R will be true at this time. Through A2 gives R,
A3 will delete R. similiarly, A4 will give R, yet A1 will delete R. So the plan
is unsafe.
    But if we look deeper into the problem, we'll find that A4 always go after A
3, and A2 always after A1. Addder always goes after deleter. So this is a plan.

                      C,!R          +P,+R
                     ________      _________
                    |   A1  |-----|   A2    |
        ___________ / --------      --------- \  _________
        |          |                           |          |
        ----------- \ C',!R         +Q,+R        ---------
                     ________      _________ /  P,Q,R
                    |  A3    |----|   A4    |
                     --------      ----------


ex2. In block world.
    initial state: on(A,D)^on(d,Tb)^on(C,Tb)^cl(A)^cl(B)^cl(C)
    goal state: on(A,B)^on(B,C)^on(C,Tb)^cl(D)

                                 ____________
                                |   Start    |
                                --------------
                                        cl(B)
                                          |
                                        cl(x)
                ______________     ______________
            S2  |move(x',y',z'|    | move(x,y,z) | S1
                ---------------    ---------------
          !cl(z')cl(y') on(x',z')       on(x,z)
                   |                     |
                   |                     |
                    --------             |
                            |            |
                          cl(D)on(A,B)on(B,C)
                                ________________
                                |    end       |
                                ________________

    We try to resolve on(B,C) using S1, and add the binding
    {x=B,y=C}
    suppose we want to resolve cl(D) first. It can also be resolved by a move ac
tion. We can add a binding here {y'=D}. We needn't make commitment on z' and x'
at this time.
    Under current binding, there isn't a threat now. Through there MIGHT be a th
reat in the furture binding. In case of variables, the old definition of threat
need to be changed. Shall we change threat to a step that possible, instead of n
ecessary, intervere the consumer and producer of a link?
    If we make this change, there will be a lot of other problem arise. We would
 like to keep the old definition, consider both codesignation and non- codesigna
tion binding. So we can promote and demote the new step accordingly. This is a b
etter algorithm.
    In this example, we should add a non-codesignated binding {z'!=D}

    Problem. When variables are introduced, how can we check the conflicts?
```

```
    1. find all the instantiated codesignation and check if it's conflicted by n
oncodesignation constraints.
    We know that the checking time is n cube. It can be done in polynomial time.
 If there are variable bindings, we should bind all the variables to real world
objects, and check that all the bindings are without confliction.
    for example, there is a binding {x=y, y=z, z=B}, then we should bind all the
 variables to {x=B,y=B,z=B}.
    But in this method, we have made a assumption that the domain is a infinite
domain. If the domain is a finite domain, there might be some other constraints
besides codesignation and noncodesignation constraints. And even throught the co
designation check is passed, the plan might still be unsafe.
    for example, in a particular domain, y can only be A and B {y=A or y=B}. Thi
s constraint won't appear. And if we have the following binding.
    {x=A y!=x y!=B}

From here, we can't find any confliction, yet in fact, if y!=B, y must be A, and
 that will conflict with y!=x .
    To solve this problem, we should derive new constraints
From domain-related constraints and find the transitive closure. But then the ch
ecking consistance time won't be polynomial. It become NP hard.
    So, we make the infinite domain assumption to make things simplier. yet we s
hould know that in the real world, almost all the domains are finite domains.
    One final change is still necessary. We can return a plan only if all variab
les have been constrained toa unique (constant) value. It's possible to instanti
ate all the variables in reasonable time when a good search algorithm is used. W
hen a A* or breadth-frith search, we can always find a solution, yet we can't ga
rantee when depth-first search is used.


2. Condition effects
   Let's start with blocks world. We have two actions: move-to and move-to-table
. The ways it's done like this is that the effect is a little different. We can
use only one action instead of two, using condition effects:
   (operator move(x,y,z)
    :precondition cl(x),cl(z),on(x,y)
    :effects on(x,z), !on(x,y), cl(y)
      if block(z) then !cl(z))

    if z is a block, z will not be clear. But it z is a table, clear(z) will sti
ll hold.
    We need to change two parts of the UCPOP to imprement that:
    1. Establishment
    makesure the P is true at S1 to get Q at a effect.

                        P,R,S
                     ___________
                    |if P then Q| S1
                    ------------
                        Q
                        |
                        Q'
                    ____________
                    |           | S2
                    -------------

                         Q
    To make a casual link S1--->S2 , we will make a binding {Q=Q'} (this might b
e a variable binding). And put the preconditions on agenda.
    R@S1,S@S1 true already.
    To make sure Q is a effect, we must add P@S1 in the agenda.
    2. Threat resolution
```

see the following example

```
                       ____________
                      |   Start    |
                      --------------
                              cl(B)
                                |
                              cl(x)
          ______________      _____________
  S2  |move(x',y',z'|     | move(x,y,z) | S1
      ---------------     ---------------
              on(x',z')       on(x,z)
       if block(z')    |       |
         then !cl(z')  |       |
                       |       |
                       |       |
                   on(A,B)on(B,C)
                      _______________
                      |    end       |
                      ________________
```

When deal with this threat, besides promotion and demotion, there is another way of doing thing, that is confractation: we must make explicitly clear !block (z) true in preconditon.

If we don't use confractation, we'll lose completeness. Counter example will be shown next class.

3. Disjunctive precondtions:

see the following example:

There are one door and one window in a room. We can move an object onto the room either through the door or throught the window. We can make two seperate actions:

open-front-door-in &

open-window-in

But now we can make just one action, using disjunctive preconditions, to represent the two actions:

```
    (operator movein
    :precondition (:or (open door) (open window))
    :effect getin)
```

On the other hand, it can be implemented by conditional effect, too.

With the usage of disjunctive precondtions, we can reduce the number of operators. If the number of operators is reduced, we can reduce the numer of committance, so reduce the branch factor.

```
    (operator block-room
     :precondition (:or (open door) (open window))
     :effect in(Block, Room)
        if (open door) then unsafe-room ...
    )
```

We can use both disjunctive precondition and conditional effect in one action. so in this example, if we want to make room unsafe, we must make (open door) true before the action.

How can we deal with disjunction when resloving the problem? Very naturaly, we want to put the disjuctions into search space, that is, for each disjunction, generate a new plan, and push it to the search space.

But now, we see that the branching factor increases again. And one of the main purpose of disjunctive precondition is to reduce the branching factor. so?

There is a trade-off between less committance and no committance. When we use disjunctive precondition, we can push the committance as late as we can. If there is less committance, the chance of error reduced. Yet no pay no game. If there is no committance, the difficult to check if the plan is correct is almost as difficult as to generate a plan. So both extreme is useless in case of effeciency. We need to consider the trade-off.

```
From AZEWS@ACVAX.INRE.ASU.EDU  Sun Feb  6 12:30:34 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <AZEWS@ACVAX.INRE.ASU.EDU>
Received:
From ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA24979; Sun, 6 Feb 94 12:30:34 MST
Received:
From ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF V4.2-13 #2382) id
 <01H8KCZ3PH0G006KTD@ACVAX.INRE.ASU.EDU>; Sun, 6 Feb 1994 12:28:30 MST
Message-Id: <01H8KCZ3Q9YA006KTD@ACVAX.INRE.ASU.EDU>
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT
From: AZEWS@ACVAX.INRE.ASU.EDU
To: plan-class@enws228.eas.asu.edu
Subject: PLANNING NOTES -- 2 FEB 94 -- ED SMITH
Date: Sun, 06 Feb 1994 12:28:30 -0700 (MST)
X-Vms-To: IN%"plan-class@enws228.eas.asu.edu"
```

```
********************************************************************
*********      Notes for 2 Feb 1994         Ed Smith     ***********
********************************************************************
```

Agenda

- Conditional Effects
- Disjunctive Preconditions
- Quantified Effects

Article: "An Introduction to Partial-Order Planning", Weld

```
********************************************************************
HOW TO HANDLE CONDITIONAL EFFECTS
********************************************************************
```

Conditional effects are effects of operators with the form:

```
    if <cond> then <action(s)>
```

They cause trouble because we don't know when they will make a genuine threat or if they can really give support.

In the case of <action(s)> we want to avoid, we must bring ~<cond> before the action and add ~<cond> to the agenda. Aside (B) (at the end of these notes) discusses negated goals.

In the case of <action(s)> we need, we must bring <cond> before the action and add <cond> to the agenda.

```
********************************************************************
```

```
DISJUNCTIVE PRECONDITIONS
*********************************************************************
DISJUNCTIVE PRECONDITIONS are a way of expressing that an action can
be done if any one of a number of things are true. For example, the
house can be entered if the front door is open or if the side window
is open.

DISJUNCTIONS will appear in POPs if any invoked action has a
DISJUNCTIVE EFFECT. In the example of tossing a coin, DISJUNCTIVE
EFFECTS can express uncertainty (incomplete information). We know
that after tossing the coin, we will have either Showing(Heads) or
Showing (Tails).

It is ***NOT*** ok to split a disjunctive precondition into N
separate problems and then pursue each problem.

UCPOP ***CANNOT*** deal with nondeterminism. As in page 3 of Weld, we
will limit ourselves in two ways - namely, we will require
DETERMINISTIC EFFECTS and OMNISCIENCE.

No actions may have disjunctive effects.
From this we get two
constraints:
(1) Complete information -> no disjunctions in the initial state.
(2) Actions must not have nondeterministic effects.
Of these two, (2) is more constraining.

Furthermore, all actions must explicitly state their effects - e.g.,
via the add and delete lists of STRIPS representation.

*********************************************************************
HOW TO HANDLE QUANTIFIED EFFECTS
*********************************************************************
QUANTIFIED EFFECTS are of two flavors: UNIVERSAL and EXISTENTIAL.
Before handling quantifiers, consider why we use them. We use them to
avoid many more rules (as a student pointed out) or to avoid rules
that mention each item explicitly. Rules that mention each item
explicitly are bad because they are sensitive to the number of
objects in the world. Such a scheme would require a "rule generation"
step according to the world-de-jour.

We could try to handle the UNIVERSALLY QUANTIFIED effects problem by
creating another independent rule. Such a rule (in the briefcase
example) might state that for each object in the briefcase, that
object's location is given by the location of the briefcase. This
approach imparts bad modularity
From a software engineering
viewpoint, and the FRAME PROBLEM comes up. In the briefcase example,
we would end up having to refer to the new independent rule each time
we wanted to know the location of an object. SPHEXISHNESS results.

UNIVERSAL QUANTIFICATION requires two modifications to POP: the
ESTABLISHMENT and the THREAT DETECTION schemes must be modified.

The following example shows ESTABLISHMENT with universally quantified
effects. Consider the case where putting the dictionary in the
briefcase, and taking the briefcase to work gets the dictionary to
work:

     +------------------+
     |      Initial     |
     +------------------+
At(Briefcase,Home) & In(Dictionary,Briefcase) & At(Dictionary,Home)

     +------------------+
     |  move ( B, H, O) |
     +------------------+
all x * In(X,Briefcase) -> At(X,Office)

At(Dictionary,Office)
     +------------------+
     |       goal       |
     +------------------+


For all parts of binding with variables that are universally
quantified, bring the unified atom(s) to the precondition of the
operation. These new preconditions then enter the agenda and links
must be formed for them.

     +------------------+
     |      Initial     |
     +------------------+
In(Dictionary,Briefcase) & At(Dictionary,Home) & At(Briefcase,Home)
                 |
                 V
         In(X,Briefcase) with binding {Dictionary/X}
     +------------------+   ( Below, the link for at office bound )
     |  move ( B, H, O) |   ( X to Dictionary, so we added the    )
     +------------------+   ( precondition In(X,Briefcase) above. )
All x * In(X,Briefcase) -> At(X,Office)
                              |
                              V
                       At(Dictionary,Office)
     +------------------+
     |       goal       |
     +------------------+


THREAT DETECTION is modified to include unification and confrontation
of unified universally quantified variables.

     +------------------+
     |        Sn        |
     +------------------+
          Clear(X)
             |
             |               +------------------+
             |               |        S         |
             |               +------------------+
             |           All x * Green(X) -> ~Clear(X)
             |
             V
          Clear(A)
     +------------------+
     |       Sn+1       |
     +------------------+

After seeing that ~Clear(X) will give us trouble when X=A, we add the
precondition ~Green(A):

     +------------------+
```

```
     |        Sn        |
     +------------------+
           Clear(X)
              |              ~Green(A)
              |                +------------------+
              |                |        S         |
              |                +------------------+
              |            All x * Green(X) -> ~Clear(X)
              |
              V         unifying Clear(X) w/ Clear(A) -> {X/A}
           Clear(A)
     +------------------+
     |       Sn+1       |
     +------------------+
```

EXISTENTIAL QUANTIFIERS require no modifications to POP: We have been handling these all along! (The goal states are dummy actions with preconditions.) They are dealt with indirectly. ASIDE (C) (see below) forbids any operation that would introduce new variables. As other links are formed, the unification procedure binds them.

****
ASIDES:
****
(A) There are problems where AVOIDING CONFRONTATION -> ~COMPLETENESS.

(B) Can we deal with negative goals?
Yes - nothing is different. We form links just as before, but note that there is one (more) interesting source of precondition atoms: the initial state. Here, we need the CLOSED WORLD ASSUMPTION:
In the initial state, or null operator’s effect, we have a complete description of the world. If the an atom does not appear in the conjunctive description of the initial state, then we can assume the negation of the atom is true.

(C) When UCPOP finishes, what should we do if we’re left with uninstantiated variables? In such a case, we would not know if we had the description of a real solution or not! We answer this question by constraining the actions to avoid the situation: If each variable that occurs in the effect of an action also occurs in (1) the precondition or (2) the antecedent of a conditional effect, then this condition will not arise. Notice that UCPOP requires actions to "declare" all variables; look at the file domains.lisp.

****
*********   End of Notes for 2 Feb 1994   ***********
****

From rao Sun Feb 6 14:00:45 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA25006; Sun, 6 Feb 94 14:00:45 MST
Message-Id: <9402062100.AA25006@parikalpik.eas.asu.edu>
References: <01H8KCZ3Q9YA006KTD@ACVAX.INRE.ASU.EDU>
From: rao (Subbarao Kambhampati)
To: AZEWS@ACVAX.INRE.ASU.EDU
Cc: plan-class@parikalpik.eas.asu.edu
Subject: PLANNING NOTES -- 2 FEB 94 -- ED SMITH
Date: Sun, 6 Feb 94 14:00:45 MST
In-Reply-To: <01H8KCZ3Q9YA006KTD@ACVAX.INRE.ASU.EDU>
Reply-To: rao@asuvax.asu.edu

Ed-- Thanks for the notes. I believe the notes for the class of 31st Jan (which Yong Qu took) are still not done. Let try to stick to a one-week turn around on these so you write down when the class is fresh in your mind.

thanks
Rao

From yqu@enuxsa.eas.asu.edu Sun Feb 6 14:41:24 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil])
Status: RO
Return-Path: <yqu@enuxsa.eas.asu.edu>
Received:
From enuxsa.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA25339; Sun, 6 Feb 94 14:41:24 MST
Received:
From localhost (yqu@localhost) by enuxsa.eas.asu.edu (8.6.4/8.6.4) id OAA25438 for plan-class@enws228.eas.asu.edu; Sun, 6 Feb 1994 14:41:01 -0700
From: Yong Qu <yqu@enuxsa.eas.asu.edu>
Message-Id: <199402062141.OAA25438@enuxsa.eas.asu.edu>
Subject: Note of Jan 31 class--resend
To: plan-class@parikalpik.eas.asu.edu
Date: Sun, 6 Feb 1994 14:41:00 -0700 (MST)
X-Mailer: ELM [version 2.4 PL23]
Content-Type: text
Content-Length: 13829

I used textedit to input the note and I ignored the carriage return inside one paragraph. Please use this version if you want to print it out.
Sorry for the inconvenience.

Yong Qu

********   Class note for January 31, 1994   ******************
Planning seminar
taken by Yong Qu

Agenda
- Variables
- Conditional effects
- Disjunctive preconditions
- Universal quantities
- UCPOP

Review:
In last class, we have learned that compared with POP, UCPOP has some new

features: conditional effects, disjunctive precondition and univeral quantities.
UCPOP is sound and complete. That means if there is a solution to a plan problem, UCPOP can always find that. but UCPOP can't garantee to find the solution in reasonable time: effeciency is always a big problem. And some method are used to improved effeciency.

In Today:

1. Variables
Upto now, all the actions in POP are instantiated actions, like move-A-B-C means move block A
From B to C. Then we need a whole bench of actions to represent all the actions of the world.
Now we can use one parameterized action instead of several instantiated actions.
For example, in the blocks world, the action move can be instantiated to several actions, each correspondence to a block. To simplizied that, we can introduced a parameterized action move(x,y,z)

```
   operator move(x,y,z)
   :precondition cl(x)^cl(z)^on(x,y)
   :effect !cl(z)^cl(y)^on(x,z)
```

It's fairly clear that the operator move(x,y,z) is a much more economical descritption than the fully specified move actions it replaces. In addition, the abstract description has enormous software engineering benefits -- needless duplication would likely lead to inconsistent domain definitions if an error in one copy was replaced but other copies were mistakenly left unchanged.
But now each partial plan is a four - tuple:
P=<S,O,B,L>
where S stands for Steps, O for order, B for binding, and L for links.
We should change POP is the following parts of POP:
a. Establishment: introduce an action.
b. Threat detection
ex. in block world , there is one action move(x,y,z)

```
     :precondition clear(x)^clear(z)^on(x,y)
     :effect !on(x,y)^ on(x,z)
 initial state: cl(A) cl(B) cl(C)
 goal state : on(A,B) on(B,C)
```

```
                               ____________
                              |   Start    |
                              --------------
                                   cl(B)
                                     |
                                   cl(x)
            ______________       ______________
       S2  |move(x',y',z')|     | move(x,y,z) | S1
            ---------------      ---------------
            !cl(z') on(x',z')        on(x,z)
                            |        |
                            |        |
                            |        |
                            |        |
                          on(A,B)on(B,C)
                            _______________
                           |    end        |
                            _______________
```

We use step S1 to resolve on(B,C) and S2 to resolve on(A,B)
Here,

```
S: start, S1,S2, end

O:
       /-------S1-------\
   start                 end
       \-------S2-------/

B: {x=B, z=C, x'=A z'=B}

          cl(B)
L: start--------->S1
```

here, S1 is to move B
From someplace to on top of C.
yet in S2 there is a effect of !cl(z'). and, unfortunately, z' is binded to B, so this might be a threat:

```
  S2    --> !cl(B)
  cl(B) -->  S1     : it's a threat.
```

Here, we give the definition of a threat:

Def. A THREAT is any step which intervene the producer and consumer of a action and necessary delete the condition.
If there is a threat, we must resolve it.
If a new action is a threat to an existed casual link, there are two methods to resolve the threat:
a. Promotion: put the new action in front of the producer, or
b. Demotion : put the new action after the consumer.

In this example, S2 is a threat to start-->S1 casual link, then we can either put S2 in front of start (promotion), or put S2 after S1 (demotion). In this example, we know that it's impossible to put S2 in front of start, and we can cut out this branch, yet in ordinally case, both two cases should be considered.

Problem: how to detect a confliction among casual links?
Solution: If there is a circle in the constraints, then we should stop this branch and cut it off.
We have assumed that each new steps should be between the dummy steps start and end, so to every step Si, we got
start<Si<end
In this example, we have start<S1<end.
in case of promotion, we want to put S2 in front of start, that means we need to add a new constraint S2<start.
Yet it's default that start<S2<end. So there is a circle exist start<S2<start. So we prune this branch.

In UCPOP, new constraints can only be added in, they will never be removed. New steps, add new constraints, are added to resolve the threats existed. so there is one problem: do we need the "white knight", that is, if there is a casual link, for example,

```
     +P
 S1---->S2,
```

can we add some steps between S1 and S2, Si..Sj, such that P is deleted and than added again, (Si will delete P and Sj will add P)?
Well, the answer is no. Because we don't have to deal with it. UCPOP is sound and completeness, we can find every solution to a problem, if it exists. the situation above will be explored by another plan in UCPOP, without having to use "white knight". We can simplify our planning if we don't deal with white knight, otherwise, the branching factor will be much larger. (To delete a casual link, we need to add in some more steps, and to resolve the new casual links, more and more steps need to be introduced..., the branching factor is much

larger.)

Another problem: how can we know that a solution has been found?
In a ground linearization, if there is nothing in the agenda, and there is not threat, then a unique sequence , that is , a solution, has been found.
Now the way we make a planning is different
From taking a plan and check if
it's correct. In some sense, me "make" the current partial correct. All the constraints are kept, and we must make sure that all the link constraints are safe. Nothing will come between with the casual link and delete. When all the ground linearization are correct and nothing left on agenda, then a total is got.

ex1. In the following chart, We want to achieve P,Q,R at goal. At the present partial plan, A2 gives P, A4 gives Q, and no other actions will delete them. so under whatever kind of action sequence, P and Q will always be true.
But we can't garantee that R will be true at this time. Through A2 gives R, A3 will delete R. similiarly, A4 will give R, yet A1 will delete R. So the plan is unsafe.
But if we look deeper into the problem, we'll find that A4 always go after A3, and A2 always after A1. Addder always goes after deleter. So this is a plan.

```
                C,!R          +P,+R
               ________      _________
              |   A1  |-----|   A2    |
 ___________ / --------      --------- \  _________
|           |                          |  ---------|
 ----------- \ C',!R          +Q,+R       ---------
               ________      _________ /  P,Q,R
              |  A3    |----|   A4    |
               --------      ----------
```

ex2. In block world.
initial state: on(A,D)^on(d,Tb)^on(C,Tb)^cl(A)^cl(B)^cl(C)
goal state: on(A,B)^on(B,C)^on(C,Tb)^cl(D)

```
                                 ____________
                                |   Start    |
                                 -------------
                                        cl(B)
                                          |
                                        cl(x)
              _______________    ______________
       S2   |move(x',y',z'|    | move(x,y,z) | S1
             ---------------    ---------------
      !cl(z')cl(y') on(x',z')       on(x,z)
                 |                     |
                 |                     |
                  --------             |
                           |           |
                       cl(D)on(A,B)on(B,C)
                           _______________
                          |    end        |
                           ________________
```

We try to resolve on(B,C) using S1, and add the binding
{x=B,y=C}
suppose we want to resolve cl(D) first. It can also be resolved by a move action. We can add a binding here {y'=D}. We needn't make commitment on z' and x' at this time.
Under current binding, there isn't a threat now. Through there MIGHT be a threat in the furture binding. In case of variables, the old definition of threat need to be changed. Shall we change threat to a step that possible, instead of necessary, intervere the consumer and producer of a link?
If we make this change, there will be a lot of other problem arise. We would like to keep the old definition, consider both codesignation and non-codesignation binding. So we can promote and demote the new step accordingly. This is a better algorithm.
In this example, we should add a non-codesignated binding {z'!=D}

Problem. When variables are introduced, how can we check the conflicts?
1. find all the instantiated codesignation and check if it's conflicted by noncodesignation constraints.
We know that the checking time is n cube. It can be done in polynomial time. If there are variable bindings, we should bind all the variables to real world objects, and check that all the bindings are without confliction.
for example, there is a binding {x=y, y=z, z=B}, then we should bind all the variables to {x=B,y=B,z=B}.
But in this method, we have made a assumption that the domain is a infinite domain. If the domain is a finite domain, there might be some other constraints besides codesignation and noncodesignation constraints. And even throught the codesignation check is passed, the plan might still be unsafe.
for example, in a particular domain, y can only be A and B {y=A or y=B}. This constraint won't appear. And if we have the following binding.
{x=A y!=x y!=B}

From here, we can't find any confliction, yet in fact, if y!=B, y must be A, and that will conflict with y!=x .
To solve this problem, we should derive new constraints
From domain-related
constraints and find the transitive closure. But then the checking consistance time won't be polynomial. It become NP hard.
So, we make the infinite domain assumption to make things simplier. yet we should know that in the real world, almost all the domains are finite domains.
One final change is still necessary. We can return a plan only if all variables have been constrained toa unique (constant) value. It's possible to instantiate all the variables in reasonable time when a good search algorithm is used. When a A* or breadth-frith search, we can always find a solution, yet we can't garantee when depth-first search is used.

2. Condition effects
Let's start with blocks world. We have two actions: move-to and move-to-table. The ways it's done like this is that the effect is a little different. We can use only one action instead of two, using condition effects:

```
(operator move(x,y,z)
  :precondition cl(x),cl(z),on(x,y)
  :effects on(x,z), !on(x,y), cl(y)
    if block(z) then !cl(z))
```

if z is a block, z will not be clear. But it z is a table, clear(z) will still hold.
We need to change two parts of the UCPOP to imprement that:
1. Establishment
makesure the P is true at S1 to get Q at a effect.

```
            P,R,S
         ___________
        |if P then Q| S1
         ------------
```

```
                       Q
                       |
                       Q'
                   ____________
                   |           | S2
                   -------------

                        Q
```

To make a casual link S1--->S2 , we will make a binding {Q=Q'} (this might be a variable binding). And put the preconditions on agenda.
R@S1,S@S1 true already.
To make sure Q is a effect, we must add P@S1 in the agenda.
2. Threat resolution
see the following example

```
                           ____________
                           |   Start    |
                           --------------
                                   cl(B)
                                     |
                                   cl(x)
               ______________    _______________
         S2  |move(x',y',z')|     | move(x,y,z) | S1
             ---------------     ---------------
                     on(x',z')       on(x,z)
             if block(z')    |       |
               then !cl(z')  |       |
                             |       |
                             |       |
                          on(A,B)on(B,C)
                            _______________
                           |    end       |
                           ________________
```

When deal with this threat, besides promotion and demotion, there is another way of doing thing, that is confractation: we must make explicitly clear !block(z) true in preconditon.
If we don't use confractation, we'll lose completeness. Counter example will be shown next class.

3. Disjunctive precondtions:
see the following example:
There are one door and one window in a room. We can move an object onto the room either through the door or throught the window. We can make two seperate actions:
open-front-door-in &
open-window-in
But now we can make just one action, using disjunctive preconditions, to represent the two actions:

```
    (operator movein
    :precondition (:or (open door) (open window))
    :effect getin)
```

On the other hand, it can be implemented by conditional effect, too.
With the usage of disjunctive precondtions, we can reduce the number of operators. If the number of operators is reduced, we can reduce the numer of committance, so reduce the branch factor.

```
    (operator block-room
     :precondition (:or (open door) (open window))
     :effect in(Block, Room)
        if (open door) then unsafe-room ...
    )
```

We can use both disjunctive precondition and conditional effect in one action. so in this example, if we want to make room unsafe, we must make (open door) true before the action.
How can we deal with disjunction when resloving the problem? Very naturaly, we want to put the disjuctions into search space, that is, for each disjunction, generate a new plan, and push it to the search space.
But now, we see that the branching factor increases again. And one of the main purpose of disjunctive precondition is to reduce the branching factor. so?
There is a trade-off between less committance and no committance. When we use disjunctive precondition, we can push the committance as late as we can. If there is less committance, the chance of error reduced. Yet no pay no game. If there is no committance, the difficult to check if the plan is correct is almost as difficult as to generate a plan. So both extreme is useless in case of effecieny. We need to consider the trade-off.

From rao Tue Feb 8 08:38:23 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA27396; Tue, 8 Feb 94 08:38:23 MST
Message-Id: <9402081538.AA27396@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Maintenance goals and goals on intermediate states
Date: Tue, 8 Feb 94 08:38:23 MST
Reply-To: rao@asuvax.asu.edu

The following contains a preamble, and two simple exercises. I will ask one of you to present the solutions to the exercises in the next class.

One of the things that we ignored after the first class is the fact that the objective of planning in general is to exhibit behavioral patterns. So, in general, a planning goal could be any constraint on the behavior (e.g.: achieve (not(Clear C)) while keeping On(A,B) true; achieve On(A,B) by first putting A on something else, and then transfering it to B, Achiece On(C,A) by using at most 3 instances of Puton operator etc. etc.).

In discussing UCPOP however, the only type of goals that we had explicitly concentrated on were the so-called goals of attainment (achieve On(C,A); achieve (not(On(B,A))).

This *DOES NOT* however mean that UCPOP cannot handle other types of behavioral constraints including maintenance goals and intermediate goals. In this mail, we will look at HOW we can handle them within UCPOP.

The key insight that will help us handle these types of goals is the following:

In the current UCPOP/POP algorithm, we start the planner off with a dummy partial plan which contains the *start* and *end* steps, with *start* preceding *end* and with all the top level goals expressed as the preconditions of the *end* step (and the initial state expressed as effects of *start*). Note that we only initialize the steps, orderings and agenda fields of the partial plan, leaving the links and bindings fields empty.

In general, there is no reason why we can't start with a richer dummy plan, that has more steps (either dummy or non-dummy) than *start* and *end*, and we we can't initialize not only the orderings, but also other fields such as links.

Given this insight, it should be a simple matter to deal with maintenance goals and intermediate goals.

By next class, think of how you can solve the following two problems withing POP/UCPOP alogorithm:

Problem 1. On(A,B), On(B,Table), Clear(C) , On(C,Table) ,
Clear(D), On(D, Table) in the initial state.

Goal: We want to achieve (not(Clear(C)) while MAINTAINING On(A,B)

Problem 2. In the initial state, the brief-case is at home. The Paycheck is at home.

Goal: Take the pay check to the office and bring it back to home
(This is kinda like doing a roundtrip
From city to city),
(Note that you can't do problem 2 with our current goals of attainment, since by the end of plan the pay check is where it was at the beginning. So, there is no state change from initial to final state!)

Get thinking...;-)

Rao
[Feb 8, 1994]

From rao Mon Feb 14 22:19:10 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
[nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA06978; Mon, 14 Feb 94 22:19:10 MST
Message-Id: <9402150519.AA06978@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Secondary Preconditions based theory of planning (forward notes)
Date: Mon, 14 Feb 94 22:19:10 MST
Reply-To: rao@asuvax.asu.edu

In today's class, I said that Chapman's TWEAK truth criterion only dealt with patially ordered plans with variablized effects. There was no treatment of conditional effects and universally quantified effects etc.

We already know that UCPOP does use conditional effects, universally quantified effects etc. This brings up the question:

"Who/what is the theory of planning with expressive actions?"

The theory of planning with expressive effects is due to Pednault. Specifically, Pednault stated a general theory of planning with "regressive operators", and showed that a large sub-class of situational calculus can be handled in terms of these operators. The plus point is that for this subclass, the frame axioms can be written automatically.

THe following are the relevant sectiosn
From last years's notes (the
completes volume of which has been distributed to you on the first day). Please look at it before coming to next class, since we will be discussing this stuff at some length next class.

rao
[Feb 14, 1994]

Notes for the 9 March Planning Seminar

Note Taker: Greg Elder

1. Actions as state changes:

   a. Actions cause a change to the world; i.e., a transition between states

   b. An action can therefore be represented as a set of tuples <s, t>, where s is the current state and t is the next state. An action is executeable at state s if and only if there is a state t for which <s, t> is a set of state transitions for that action.

2. Regression

   a. Regression operators are used to reason about plans. Using regression operators, one can determine if a desired condition wil be true after executing a sequence of actions.

   b. A regression operator a-1 for action a is a function mapping formulas to formulas with the property that for each formula rho and every pair of states <s, t> which are elements of a, if a-1(rho) is true in s, then rho is true in t.

   c. Given a plan, regression can be used to determine if the plan is correct.

   d. Progression used to project effects of actions into the future; allows you to conclude precondition of next action

   e. A special case of the regression operator is called the Tidy regression operator. Not only should orginal regression definition be satisfied but the reverse as well--necessary and sufficient conditions.

If s |= a-1(rho) then t |= rho
If t |= rho then s |= a-1(rho)

If actions don't have non-deterministic effects, then you can use Tidy regression operators.

f. If you have necessary and sufficient regression operators, and regression formulas are satisfied for a plan, then the plan is correct under all conditions.

g. Use regression not just for checking correctness of a plan, but to generate other plans. Use regression as a basis for doing planning.

3. Causality Theorem

a. Condition rho is true at a point p during execution if and only if one of the following holds:

(1) An action a is executed prior to point p such that a makes rho true and rho remains true until at least point p.

(2) Rho is true in the initial state and remains true until at least point p.

b. Sigma(rho a) is a causation precondition.

Pi(rho a) is known as a preservation precondition

c. Causation and preservation preconditions can be defined in terms of regression operators.

d. Causality Theorem can be expressed using causation and preservation preconditions.

A condition rho will be true at point p during the execution of a plan if and only if one of the following holds:

(1) An action a is executed prior to point p such that

(a) Sigma(rho a) is true immediately before executing a.

(b) Pi(rho a) is true immediately before the execution of each action b between a and point p.

(2) Rho is true in the initial state and Pi(rho a) is true immediately before every action a priot to p.

From gary@enws320.eas.asu.edu Thu Mar 11 20:02:51 1993
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
[nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <gary@enws320.eas.asu.edu>
Received:
From enws320.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA18261; Thu, 11 Mar 93 20:02:51 MST
Received: by enws320.eas.asu.edu (4.1/SMI-4.1)
id AA01005; Thu, 11 Mar 93 19:56:55 MST
Date: Thu, 11 Mar 93 19:56:55 MST
From: gary@enws320.eas.asu.edu (Kevin Gary)
Message-Id: <9303120256.AA01005@enws320.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

Planning Class Notes for Wednesday March 10

by Kevin Gary

Doing Planning Using Causality Theorems

Objectives:
===========

- Going beyond STRIPS:
  1> Complex goals - including quantifiers, disjunction, and negation
  2> Actions with Context-dependent effects

- generalize plan refinement to nonlinear planning

Theoretical Basis for Linear Planning

Causality Theorem (
From Pednault - see class 3/9)
- gives sufficiency conditions

Diagram: Nondeterministic Planning Obtained
From the Causality Theorem:

```
            (exists)new a         a < p
               /            \      /
              /              \    /
          or              and  ----  a achieves phi
         /  \             /    \
        /    \           /      \
   and    (exists) old a      (forall)b, a < b < p, b preserves phi
   /  \
  /    \---  (forall)b < p, b preserves phi
 /
/
phi is true in the initial state
```

How does a Planner assert that an action achieves/preserves a desired goal?

- The action must be executed in the appropriate context.

- The _context_ can be defined by secondary preconditions that are introduced as additional preconditions to the actions in order for it to produce the desired effects.

2 types of secondary preconditions: (note: @ used for "phi")

___ a

```
\    =  _Causation_ precondition for action a to achieve @
/__
    @

----- a
 | |   = _Preservation_ precondition to preserve @
 | |
     @
```

From class on 3/9 we looked at computing these as follows:

```
___ a                             ----- a
\    = alpha(a, R)                 | |    = (not)delta(a, R)
/__                                | | R
    R
```

- These are relational (atomic) formulae
- We have no way of dealing with non-atomic formulae now.

- We have seen causation and preservation preconditions before:

```
Causation:    codesignation constraints
Preservation: non-codesignation constraints
```

- Looking at TWEAK with ADL's glasses, these constraints are considered separate
From treating them as subgoals like we'd normally do with sigma and pi above.

Theoretical Basis for Secondary Preconditions:

```
   @ is provably true after executing a1,...,an if and only if:

                                             ----- ak
   1> @ is provably true initially and  | |       is provably true just prior
                                              | | @

      to executing each action ak for 1<= k <= n.

                    ___ ak
   2> For some k, \        is provably true just prior to executing action ak,
                    /__ @
            ----- ai
      and  | |      is provably true just prior to executing each action
            | | @    ai for k < i <= n.

   So now the new diagram is:

            (exists)new a         a < p
            /            \       /
           /              \     /
       or               and  ----  a achieves sigma(a, phi)
      /  \             /     \
     /    \           /       \
   and    (exists) old a      (forall)b, a < b < p, achieve  pi(b, phi)
   /  \
  /    \---  (forall)b < p, achieve  pi(b, phi)
 /
/
phi is true in the initial state
```

- We then went through Pednault's examples (Pednault, 88)

  - The Briefcase Problem (p. 364) assumes conjunctive, unquantified (atomic) formulae. One path through the nondeterministic space is shown. Sigma and Pi are defined as follows:

```
___ a                              ----- a
\       = alpha(a, R)               | |      = (not) delta(a, R)
/__                                 | |
    R()                                 R()

___ a                              ----- a
\       = delta(a, R)               | |      = (not) alpha(a, R)
/__                                 | |
    (not)R()                            (not)R()
```

  - The Bomb in the Toilet Problem (p. 3) has an initial incomplete state - don't know if package A or B really has the bomb. So, In(bomb, A) v In(bomb, B) is true in the initial state, but we can't conclude which of {In(bomb, A), In(bomb, B)} is in the "actual" initial state. Sigma and Pi are defined in terms of regression operators $^{-1}_{a}$

```
From rao  Wed Feb 16 14:16:29 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
	[nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
	id AA09057; Wed, 16 Feb 94 14:16:29 MST
Message-Id: <9402162116.AA09057@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: papers on reserve
Date: Wed, 16 Feb 94 14:16:29 MST
Reply-To: rao@asuvax.asu.edu
```

I am putting a ring-binder with several papers in the AI lab (the same place where the Readings in Planning book is placed).

I will be periodically adding papers to this ring binder. The papers will either be discussed in the class directly, or may be assigned as supplementary information for something that was discussed in the class.

Currently, the binder contains several papers on Pednault's ADL based theory of planning (we will discuss this briefly in today's [Feb 16, 1994] class).

The paper "Synthesizing plans that contain actions with context dependent effects" provides the planning theory for total ordering plan-space planning.

The paper "Generalizing nonlinear planning to handler complex goals and actions with context dependent effects" extends the previous paper to allow for partially ordered plans in the search space.

The paper "UCPOP: A sound, complete and Partial order planner for ADL" is the technical paper on UCPOP that explains how UCPOP derives its theory
From ADL. Those of you who like a good soundness/completeness proof SHOULD look at this paper to see how the soundness and completeness of UCPOP algorithm is proved.

The paepr "ADL: Exploring the middle ground between STRIPS and Situation Calculus" is written
From a Knowledge representation
point of view and explains how ADL is related to situational calculus (those
From Intro to AI remember that we ran away
From sit-calc
because of the frame problem; ADL turns out to be the largest known subset of situational calculus for which frame axioms can be generated automatically).

Happy reading. I will be more than happy to discuss/answer any question you may have when reading these papers.

Rao
[Feb 16, 1994]

```
From rao  Thu Feb 17 10:37:18 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA09988; Thu, 17 Feb 94 10:37:18 MST
Message-Id: <9402171737.AA09988@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: More on ADL vs. UCPOP
Date: Thu, 17 Feb 94 10:37:18 MST
Reply-To: rao@asuvax.asu.edu
```

Here is a simplified version of the bomb-in-the-toilet problem that ADL based planning theory can solve but UCPOP cannot.

Consider the problem and think about why UCPOP cannot solve it.

Problem

Initial state: P V Q

Final state: R

Actions: (two actions are available)

```
    A1
     precond: nil
     effects: If P then R

    A2
     precond: nil
     effects: If Q then R
```

We know that the following plan

A1 --> A2

will solve this problem (i.e., if you execute A1 and then A2, you are guaranteed to have R true in the resulting situation).

Qn 1. Explain how the secondary preconditions based theory of planning can be used to (a) check that this plan is correct and (b) make this plan.

Qn 2. What are the reasons UCPOP algorithm is not able to deal with this problem?

(I hope that you will have answers for this by next class -- feel free to send your answers to the class list).

The following is a hint that you should look at after you tried the problem for a while:

```
Hint: Note that in the class we said that causation preconditions are
that part of the regression which deal with the situation where the
formular phi is not already true. This differentiation between
causation preconditions and regression makes sense only in situations
where we can tell whether the formular phi is true or false. If we
can't answer this question, then causation preconditions are the same
as regression. Now convince yourself that (a) in ADL, the only time
when you can't tell whether a formula phi is true or false in a given
situation is when the initial state has incomplete information (hint:
actions cannot have incomplete effects. So the only incomplete
information must the stuff that is getting propagated
From initial
state). (b) given this, show to yourself that if you use regression as
the causation precondition, you can both show the plan above to be
correct, and generate it yourself. (c) now think about why this is
hard to do for UCPOP.

Rao

From ATAPK@ACVAX.INRE.ASU.EDU  Thu Feb 17 14:38:23 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <ATAPK@ACVAX.INRE.ASU.EDU>
Received:
From ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11245; Thu, 17 Feb 94 14:38:23 MST
Received:
From ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF V4.2-13 #2382) id
 <01H8ZUMH8XZ400DUDC@ACVAX.INRE.ASU.EDU>; Thu, 17 Feb 1994 14:35:05 MST
Date: Thu, 17 Feb 1994 14:35:00 -0700 (MST)
From: ATAPK@ACVAX.INRE.ASU.EDU
Subject: PLANNING CLASS NOTES - FEB. 7th
To: PLAN-CLASS@PARIKALPIK.eas.asu.edu
Message-Id: <01H8ZUMH9QWY00DUDC@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"PLAN-CLASS@PARIKALPIK.EAS.ASU.EDU"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT

******************************************************************
CSE 591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE
NOTES
From THE SESSION ON FEBRUARY 7th
Notes taken by: Dimitri Karadimce
******************************************************************

Agenda:

   1. We look at an illustrative example of application of
      the UCPOP algorithm.  In particular, we illustrate
      the handling of universally quantified variables and
      conditional effects.  Along with the explanation of
      individual steps, we also provide some additional
      insights about various features of UCPOP.

   2. We further clarify specific aspects of the UCPOP,
      by discussing specific features and details of UCPOP.

   3. We take a look at UCPOP algorithm
From another perspective:
      we try to understand the rationale behind its individual
      steps, and we consider alternative ways of dealing with
      some particular problems of partial order planning.

******************************************************************
1. AN EXAMPLE OF UCPOP APPLICATION IN A PLANNING PROBLEM

We look at an example of application of the UCPOP algorithm,
featuring universally quantified variables and conditional
effects.  In due course, we also discuss some interesting
features of UCPOP.

We are using the specification of UCPOP algorithm as given in
the paper: "An Introduction to Partial Order Planning" by D.S.Weld.

1.1. The Planning Problem

A paycheck is in a briefcase, and that briefcase is at home.
The goal is:  the briefcase should be at the office, and the
paycheck at home.

Formally, we have the following problem description:

   a. Let P designate the particular paycheck we are interested in
      Let B designate the particular briefcase we are interested in
      Let H designate the particular location we call "home"
      Let O designate the particular location we call "the office"
      Let single lowercase letters designate variables.

   b. Initial state:

             object(P)
             briefcase(B)
             in(P,B)
             at(B,H)
             at(P,H)

          Note: although at(P,H) can be inferred
From in(P,B) & at(B,H)
                using obvious rules of inference, we are still speficying
                it explicitly, to satisfy the "complete information"
                criterion for the initial state.

   c. Final state:

             at(B,O)
             at(P,H)

   d. Available Actions:

                                                 briefcase(b)
                                                 at(b,l1)
      precondition:   in(x,b)                    l1 != l2
                    +----------------+     +--------------------+
      action:       | take-out(x,b) |     |   move(b,l1,l2)    |
                    +----------------+     +--------------------+
      effect:         ~in(x,b)                   at(b,l2)
                                                 ~at(b,l1)
```

```
                                    (forall x such-that object(x))
                                         when  in(x,b)
                                         then  at(x,l2) & ~at(x,l1)
```

1.2. Generating a Plan Using the UCPOP Algorithm

At the beginning we have the following partial plan, based on
the initial and the final state, and UCPOP conventions for their
representation:

```
                +-------------------------+
                |         *start*         |
                +-------------------------+
    object(P) & briefcase(B) & in(P,B) & at(B,H) & at(P,H)



                     at(B,O) & at(P,H)
                +-------------------------+
                |          *end*          |
                +-------------------------+
```

AGENDA: { at(B,O) @ *end*, at(P,H) @ *end* }

UCPOP now picks a conjunct
From the agenda. Note that it doesn’t
matter (
From the completeness point of view) which conjunct is chosen
-- if a solution to the planning problem exists, UCPOP is guaranteed
to find it regardless of the order of picking the conjuncts

From the agenda).

Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])

Let UCPOP pick the goal conjunct ’at(P,H)’
From the agenda.

UCPOP now needs to establish ’at(P,H)’. One choice is to
use the ’at(P,H)’ effect of the *start* step. The only other
choice is to instantiate the ’move(b,l1,l2)’ action and use
its effects.
Let UCPOP nondeterministically decide to use the ’at(P,H)’ effect
of the existing *start* step:

```
                +-------------------------+
                |         *start*         |
                +-------------------------+
    object(P)  &  briefcase(B)  &  in(P,B)  &  at(B,H)  &  at(P,H)
                                                              |
                                                              | (1)
                                  +---------------------------+
                                  |
                                  |
                     at(B,O) & at(P,H)
                +-------------------------+
                |          *end*          |
                +-------------------------+
```

AGENDA: { at(B,O) @ *end* }

Note that no new subgoals were introduced in the agenda, and the
conjunct ’at(P,H)’ was deleted
From the agenda since it is now
supported ("established") by a causal link. This type of
establishment, where a goal is supported without introducing
a new step, is refered to as "simple establishment".

Digression: recall how we "establish" goals in FORWARD PLANNING
in the space of WORLD STATES.
For example, assume we have an initial state
on(A,Table) & on(B,Table) & on(C,Table),
and we want to make a plan stacking C on A on B:
on(C,A), on(A,B), on(B,Table).
One reasonable step may be ’puton(A,Table,B)’,
which will ESTABLISH on(A,B).
We KNOW that once this step has been included in the
plan, IT CAN ALWAYS BE EXECUTED in any refinement
of the plan, since all steps are added AFTER the
last added step. In other words, once established
steps can NOT be threatened in planning in the space
of WORLD STATES.

However, when planning in the space of PARTIAL PLANS,
we don’t have a guarantee that the execution of
the step will not be threatened by subsequently
added steps (since steps can be added anywhere
in the plan). In order to guarantee the execution
of the step, we must create a causal link that will
establish the goal conjunct, and make sure that
the link is never threatened.

UCPOP now takes the only remaining conjunct ’at(B,O)’
From the agenda.

UCPOP must instantiate a ’move(b,l1,l2)’ action, because it is
the only way to support the ’at(B,O)’ goal conjunct.
We refer to this instantiation of the action as ’S1’.
We immediately put the bindings b=B, l2=O., in order to be able
to support the goal conjunct ’at(B,O)’.

```
                +-------------------------+
                |         *start*         |
                +-------------------------+
    object(P)  &  briefcase(B)  &  in(P,B)  &  at(B,H)  &  at(P,H)
                                                              |
                                                              | (1)
                                                              |
             briefcase(B), at(B,l1)                           |
            +-------------------------+                       |
         S1 |      move(B,l1,O)       |                       |
            +-------------------------+                       |
  at(B,O) & ~at(B,l1) & (forall x such-that object(x))        |
     |                        when in(x,B)                    |
     | (2)                    then at(x,O) & ~at(x,l1)        |
     |                                                        |
     |                                                        |
```

```
+---------------+         +---------------------------+
                |         |
                |         |
           at(B,O) & at(P,H)
        +-------------------------+
        |         *end*           |
        +-------------------------+


    AGENDA: { briefcase(B) @ S1, at(B,l1) @ S1}
    BINDINGS: { b=B, l2=O, l1 != O }
    ORDERINGS: { *start* < S1 < *end* }
```

Note how the conditional effect of the step S1 is specified as
an effect of S1.

Also note that at this moment we need NOT bind 'l1' in the step S1.
One might be tempted to IMMEDIATELY bind l1 to H (which will later
turn out to be the right binding). However, in general it is
a mistake to do any bindings prematurely, if they are not necessary
to support a link generated at the current iteration.

Let us check against any threatened links at this moment.

(a) The newly introduced link (2) can not be threatened, since
    no steps can be ordered between S1 and *end* (the only other
    step is *start*, and it is guaranteed to be before S1).

(b) The new step S1 looks like it can threaten the link (1),
    which supports 'at(P,H)' and goes
From *start* to *end*..
    The step S1 can be ordered between *start* and *end*,
    and it has a potential of making '~at(P,H)' due to its
    universally quantified conditional effect ~at(x,l1).
    Compute MGU(at(P,H), at(x,l1), CURRENT BINDINGS) = { x=P, l1=H }.
    However, using the definition of a threat in the case of
    universally quantified effects, we decide that the link
    is not threatened since in the MGU binding list there is
    a binding l1=H, and l1 is NOT universally quantified.
    Intuitively speaking, S1 does not threaten link (1)
    at this moment since l1 is still not bound. When l1 becomes
    bound, if l1=H, then S1 will definitely be a threat, and
    if l1 != H, then S1 will not threaten (1).

Since no threats have been detected, the UCPOP will proceed
with the next iteration.

```
     Digression: at this iteration we added goals to the agenda.
                 Here are all cases when the agenda increases:
                   1. at the beginning: all goal conjuncts
                   2. when adding a step: the preconditions of the step
                   3. when adding a step: (if necessary) the conditional
                        preconditions of the needed effect
                   4. when protecting a link: (for conditional effects,
                           if necessary) negated conditional preconditions
                                      of the threatening effect.
```

Let UCPOP pick 'briefcase(B)'
From the agenda. It can be
established simply by using the effect 'briefcase(B)' of the
start step.

Then UCPOP picks the only remaining goal conjunct 'at(B,l1)'.

Again, two principal choices are available: one is to simply
establish 'at(B,l1)'
From the effect 'at(B,H)' of the *start*
step; the other is to generate a new step by instantiating
the action 'move' again: 'move(B,l3,l1)' would have an effect
'at(B,l1)' which can be used to establish the current goal
'at(B,l1)'.

UCPOP must consider all possibilities, otherwise it could
fail to generate a plan. In other words, UCPOP must keep
track of all possible options, in order to backtrack and use
them, if necessary. Note, however, that UCPOP is free to
use some heuristics to determine the choice which looks
most promising to start with.

Let UCPOP nondeterministically decide to support 'at(B,l1)'
directly
From the effect 'at(B,H)' of the *start* step.
This forces the binding l1=H.
The step S1 now is as following:

```
                 briefcase(B), at(B,H)
                +-------------------------+
            S1 |      move(B,H,O)         |
                +-------------------------+
     at(B,O) & ~at(B,H) & (forall x such-that object(x))
                                when in(x,B)
                                then at(x,O) & ~at(x,H)
```

However, the conditional effect '~at(x,H)' now threatens the link
(1), which establishes 'at(P,H)' (see the last partial plan).
UCPOP must protect the link (1). However, neither promotion
nor demotion can be used, since the step S1 can not be ordered
before *start* or after *end*. The only choice is to use
CONFRONTATION in order to disallow the conditional effect
'~at(P,H)'. For the confrontation purposes, it is sufficient
to add '~in(P,B)' as a precondition of S1.

Note that it is NOT necessary to confront the conditional effect
for ALL objects x, since we are only concerned to disallow
'~at(P,H)', and we are NOT disallowing the effect '~at(x,H)'
for any x != P.

```
                +-------------------------+
                |         *start*         |
                +-------------------------+
    object(P)  &  briefcase(B)  &  in(P,B)  &  at(B,H)  &  at(P,H)
                      |                         |            |
                      | (3)                     | (4)        | (1)
                      |                    +----+            |
                      |                    |                 |
                  briefcase(B), ~in(P,B), at(B,H)            |
                +-------------------------+                  |
            S1 |      move(B,H,O)         |                  |
                +-------------------------+                  |
     at(B,O) & ~at(B,H) & (forall x such-that object(x))     |
        |                       when in(x,B)                 |
```

```
        | (2)                     then at(x,O) & ~at(x,H)      |
        |                                                      |
        |                                                      |
        +---------------+         +----------------------------+
                        |         |
                        |         |
                    at(B,O) & at(P,H)
                +--------------------------+
                |          *end*           |
                +--------------------------+


    AGENDA: { ~in(P,B) @ S1}
    BINDINGS: { b=B, l2=O, l1 != O, l1=H }
    ORDERINGS: { *start* < S1 < *end* }
```

No other links are threatened, so we can go on with the
establishment step.

The only way to establish '~in(P,B)' is to instantiate the
'take-out' action.
We refer to this instantiation of the action as 'S2'.
We immediately put the bindings x11=P, b11=B, in order to be able
to support the goal conjunct '~in(P,B)'.
Note that we are using new variable names, different
From the
variable names used in previous instantiations of any actions.

```
                +--------------------------+
                |         *start*          |
                +--------------------------+
    object(P)  &  briefcase(B)  &  in(P,B)  &  at(B,H)  &  at(P,H)
                       |                           |            |
                       | (3)                       | (4)        | (1)
                       |                      +----+            |
                       |                      |                 |
                       |          in(P,B)     |                 |
                       |   +---------------+  |                 |
                       | S2| take-out(P,B) |  |                 |
                       |   +---------------+  |                 |
                       |         ~in(P,B)     |                 |
                       |            |         |                 |
                       |            | (5)     |                 |
                       |            |         |                 |
                  briefcase(B), ~in(P,B), at(B,H)               |
                +--------------------------+                    |
             S1 |      move(B,H,O)         |                    |
                +--------------------------+                    |
     at(B,O) & ~at(B,H) & (forall x such-that object(x))        |
        |                         when in(x,B)                  |
        | (2)                     then at(x,O) & ~at(x,H)       |
        |                                                       |
        |                                                       |
        +---------------+         +-----------------------------+
                        |         |
                        |         |
                    at(B,O) & at(P,H)
                +--------------------------+
                |          *end*           |
                +--------------------------+
```

```
    AGENDA: { in(P,B) @ S2}
    BINDINGS: { b=B, l2=O, l1 != O, l1=H, b11=B, x11=P }
    ORDERINGS: { *start* < S1 < *end*, *start* < S2 < *end*, S2 < S1  }
```

The new step S2 does not threaten any links; the new link (5) is not
threatened by any other steps.

The last step establishes the link (6) supporting the 'in(P,B)' goal:

```
                +--------------------------+
                |         *start*          |
                +--------------------------+
    object(P)  &  briefcase(B)  &  in(P,B)  &  at(B,H)  &  at(P,H)
                       |              |             |            |
                       | (3)          | (6)         | (4)        | (1)
                       |              |        +----+            |
                       |              |        |                 |
                       |          in(P,B)      |                 |
                       |   +---------------+   |                 |
                       | S2| take-out(P,B) |   |                 |
                       |   +---------------+   |                 |
                       |         ~in(P,B)      |                 |
                       |            |          |                 |
                       |            | (5)      |                 |
                       |            |          |                 |
                  briefcase(B), ~in(P,B), at(B,H)                |
                +--------------------------+                     |
             S1 |      move(B,H,O)         |                     |
                +--------------------------+                     |
     at(B,O) & ~at(B,H) & (forall x such-that object(x))         |
        |                         when in(x,B)                   |
        | (2)                     then at(x,O) & ~at(x,H)        |
        |                                                        |
        |                                                        |
        +---------------+         +------------------------------+
                        |         |
                        |         |
                    at(B,O) & at(P,H)
                +--------------------------+
                |          *end*           |
                +--------------------------+


    AGENDA: { }
    BINDINGS: { b=B, l2=O, l1 != O, l1=H, b11=B, x11=P }
    ORDERINGS: { *start* < S1 < *end*, *start* < S2 < *end*, S2 < S1  }
```

The new link (6) is not threatened.

The agenda is empty, the binding constaints are consistent,
there are no threatened links, and there is a ground linearization
of the ordering constaints: *start* --> S2 -> S1 --> *end*.

Therefore, the planning process successfully terminates, giving
the plan: take-out(P,B) --> move(B,H,O).

****

2. OTHER COMMENTS ABOUT SPECIFIC FEATURES AND DETAILS OF UCPOP

2.1. Note how unification has to be done for the purposes of

```
    the UCPOP algotithm:

    MGU (P1, P2, CURRENT BINDINGS)

    The unification of P1 and P2 is done in the usual way,
    with the following two additions:

    (a) The list of CURRENT BINDINGS must be respected ;
    (b) The CURRENT BINDINGS may contain NON-CODESIGNATION
        constraints, such as l1 != O, which also must be
        respected in the process of finding the most general
        unifier.


2.2. In the example we presented above, we illustrated the
    handling of the possible threatening effects of
    UNIVERSAL QUANTIFICATION.  However, the UNIVERSALLY
    QUANTIFIED EFFECTS can be used to support goals as well.

    For example, let us slightly extend the above example,
    requiring that the object D (originally in the briefcase)
    be moved to the office.

    We need not add any extra steps in the plan.
    We can support the new goal 'at(D,O)' by the conditional
    effect of step S1:  (forall x such-that object(x))
                          when in(x,B)
                          then at(x,O) & ~at(x,H)

    We only need to use the effect instance for x=D.
    We then need to add the condition 'in(D,B)' of the
    utilized effect to the agenda.  This condition, in turn,
    is directly supportable by the initial state, which
    contains 'in(D,B)'.

    Note how WITHIN A SINGLE UNIVERSALLY QUANTIFIED CONDITIONAL
    EFFECT, one instance of the effect threatened a link and had to
    be confronted, and stil in the same plan, another instance of
    the same effect was used to support another link.

    The complete plan is as follows:

               +-------------------------+
               |         *start*         |
               +-------------------------+
   object(P) & briefcase(B) & in(P,B) & at(B,H) & in(D,B) & at(D,H) & at(P,H)
                     |             |        |         |                  |
                     | (3)         | (6)    | (4)     | (8)              |
                     |             |        |         |      +-----------+
                     |             |        |         |      |
                     |          in(P,B)     |         |      | (1)
                     |   +---------------+  |         |      |
                     | S2| take-out(P,B) |  |         |      |
                     |   +---------------+  |         |      |
                     |         ~in(P,B)     |         |      |
                     |           |          |         |      |
                     |           | (5)      |         |      |
                     |           |          |         |      |
                 briefcase(B), ~in(P,B), at(B,H), in(D,B)    |
               +-------------------------+                   |
            S1 |      move(B,H,O)        |                   |
```

```
               +-------------------------+                   |
   at(B,O) & ~at(B,H) & at(D,O) & ~at(D,H) &                 |
      |                  |   (forall x such-that object(x))  |
      |                  |      when in(x,B)                 |
      | (2)              |      then at(x,O) & ~at(x,H)      |
      |                  |                                   |
      |                  | (7)                               |
      +--------+         |         +-------------------------+
               |         |         |
               |         |         |
            at(B,O) & at(D,O) & at(P,H)
              +-------------------------+
              |         *end*           |
              +-------------------------+


2.3. Let us consider planning problems where the GOAL is
     universally quantified.
     Refering again to our original example, we may require
     that ALL objects be moved to the office:

         (forall x such-that object(x))   at(x,O)

     Assuming a finite domain of objects, we can "expand" the
     universal quantifier, and try to generate a plan that
     will achieve:    at(obj1,O) & at(obj2,O) & ...& at(obj-n,O).
     Since this is a finite ground conjunction, which is no different

From the UCPOP examples we have considered so far, UCPOP will
     be able to generate the plan.


2.4. The handling of negated goals in UCPOP is not different from
     the handling of non-negated goals.  We have already illustrated
     the handling of negated subgoals that were introduced in
     the agenda by the confrontation.  The treatment of real
     negated goals is no different in any way.


2.5. Consider the following planning problem:

     All files in the directory of a UNIX machine should be deleted.
     Two UNIX commands (ACTIONS) are at disposal:
        rm *      (deletes all files in the directory)
                  (*** PLEASE BE CAREFUL WHEN USING THIS COMMAND ***)
        rm file   (deletes a given file

     Obviously, there are two basic approaches (plans):  the first
     is to use 'rm *' and have a one-step plan;  the other is to
     delete the files one-by-one, using the 'rm single-file' action
     (other plans are also feasible, such as using 'rm single-file'
     several times, and then using 'rm *', but they are not as interesting).

     The question is: given a good formal specification of the actions,
     is UCPOP going to generate the two interesting plans?
     The answer is positive :

     Formal specification (D=directory, f-i = i-th file in D):
        initial state:  in(f1,D),  in(f2,D), ...,  in(f-n,D)
        final state:    ~in(f1,D), ~in(f2,D), ..., ~in(f-n,D)
        action1:   rm f-i
```

```
                 precondition: in(f-i,D)
                 effect:       ~in(f-i,D)
   action2:   rm *
                 precondition: none
                 effect:       (forall f-i such-that file(f-i))
                                   when in(f-i,D)
                                   then ~in(f-i,D)
```

Plan1: UCPOP will instantiate a separate 'action1' for each goal !in(f-i,D). Each of those steps will have the precondition in(f-i,D) which will be supported directly
From the initial
state. Therefore, the planning will successfully terminate.

Plan2: UCPOP will instantiate 'action2' once (it will have a clue to look at 'action2' since individual instances of its (conditional) effect unify with the goals in the agenda). For each file f-i it will use the corresponding instance of the universally quantified effect '~in(f-i,D)', which unifies with the corresponding goal '~in(f-i,D)'. In order to satisfy the precondition of each used instance of the universally quantified effect, UCPOP will add goals of the form in(f-i,D) to the agenda. But these goals are directly supported by the initial state, and the planning will successfully terminate.

2.6. Let us look again at the example presented in 2.5
From a
different perspective. The crutial assumption that allowed for successfull planning was that we KNEW IN ADVANCE all files in the directory. However, what if the planner was given the ability to use the full command set of UNIX ? In particular, there are commands that can CREATE new files. Nothing in UCPOP can then prevent the algorithm
From trying
to use such commands (actions) to support intermediate goals of the form in(f-i,D).

Let us consider the following scenario: UCPOP expanded the universally quantified goal in the very first step. In the process of planning, it may use an action that generates a new file. Since the non-existence of that NEW file is nowhere required as a goal that must be achieved, the planning may successfully terminate (according to UCPOP), but there may still be files in the directory. Although none of these files was among the original files in the directory, the outcome is still unsatisfactory, since we wanted to get rid of ALL files and end up with an empty directory.

The example shows that the the assumption of having STATIC UNIVERSE of objects is not realistic for some domains.

An open research question is how to deal with such non-static universes WITHIN the basic framework if UCPOP, in a CLEAN way.

Note that the ONLY change in UCPOP which is sufficient to handle non-static universes is TO CHANGE THE DEFINITION OF A THREATENED LINK, and appropriately handle such threats while generating the plan. Refering to the above example, the definition of a threat must somehow recognize the creation of a new file as a threat to the goal "all files in the directory must be deleted".

****

3. CHALLENGING VARIOUS ASPECTS OF UCPOP

We now take a look at UCPOP algorithm
From another perspective.
We want to understand the rationale behind its individual steps, and we try to consider alternative ways of dealing with some particular problems of partial order planning.

Here we will raise some doubts that will serve as a motivation for the discussion on the topics scheduled for the next session.

3.1. Why do we need causal links ?
Why do we utilize the links in this particular way ?

- why wouldn't we deal with MAINTENANCE GOALS ?
- why wouldn't we allow deletion of a subgoal, and then its addition, as opposed to disallowing the plan (as UCPOP does) ? It seems that allowing such steps may help, and (as long as such steps don't violate a maintenance goal) we don't mind such deletions.

3.2. Why wait until all preconditions are worked on ?

- why work on each goal separately ?
- it seems that very often many subgoals would come as side effects of establishment of other goals. Then why should we work on such subgoals, when anyway they would be achieved, without special steps devoted to them ?

3.3. Why wait until ALL ground linearizations are solutions ?

- recall that all the time we have a PARTIALLY ORDERED PLAN, which in general has many ground linearizations. However, we are perfectly happy with ONLY ONE plan (totally ordered sequence of steps). Then why wait until ALL ground linearizations become solutions ? Why wouldn't we stop as soon we have a single ground linearization ?

3.4. Why do we need to treat the threats as we do in UCPOP ?

- as soon as an establishment is done, why do we need to resolve all threats AT THAT MOMENT ?

- why wouldn't we wait, hoping that some of the threats would be resolved as side-effects of other steps ?

- why wouldn't we wait until any threat becomes a REAL THREAT, not just a temporary threat ?

We know that it is always better to defer the branching of the search to the very end. By resolving non-essential threats, we are effectively prematurely introducing branching in the search space, thus decreasing the performance of the planning in general.

There are various answers to the raised questions, addressing some of the doubts and problems mentioned in the questions. UCPOP may be altered to accomodate each of them, thus giving rise to a whole family of related algorithms.

In the following session we will discuss THE MOST GENERAL ALGORITHM based on the ideas of PARTIAL ORDER PLANNING, and then we will survey the various particularizations of that general algorithm, one of which will be the (now) familiar UCPOP.

****
END OF NOTES
From THE SESSION ON FEBRUARY 7th
****

From rao Mon Feb 21 10:39:44 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA13775; Mon, 21 Feb 94 10:39:44 MST
Message-Id: <9402211739.AA13775@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Sanity Check Test #1 (Due March 4th)
Date: Mon, 21 Feb 94 10:39:44 MST
Reply-To: rao@asuvax.asu.edu

Sanity Check #1 (Due 28th Feb)

The following is the "Midterm Sanity Check". I would like you to work on this and return your solutions (hand-written or electronic-- your choice) to me by 28th February (You can use the respite provided by the absence of class on 23rd).

This is not exactly an examination. You are not going to be graded on it. I do however want to make sure that I am making sense to you, and plan to read the answers carefully and comment on them. So, you may want to treat it with the same sincerity/seriousness you will an exam. In particular, consider it an Open-Book Take-Home exam.

Rao
[Feb 21, 1994]

ps: If you need clarifications specific questions, please send me e-mail.

Part A. Direct questions

The answers to the following have been discussed directly in the class in one form or another. They thus test whether the discussion in the class is making sense to you.

I. Here is a single "puton" operator for a version of blocks world that includes the "on" predicate and "above" predicate.

```
(define (operator puton)
      :parameters (?x ?y ?d)
      :precondition (and (neq ?x ?y) (neq ?x table) (neq ?d ?y)
                         (on ?x ?d)
                         (or (eq ?x Table)
                             (forall (block ?b) (not (on ?b ?x))))
                         (or (eq ?y Table)
                             (forall (block ?b) (not (on ?b ?y)))))
      :effect
      (and (on ?x ?y) (not (on ?x ?d))
           (forall (?c)
                   (when (or (eq ?y ?c) (above ?y ?c))
                     (above ?x ?c)))
           (forall (?e)
                   (when (and (above ?x ?e) (neq ?y ?e) (not (above ?y ?e)))
                     (not (above ?x ?e)))))))
```

Qn 1. What happened to the "clear" predicate that we typically use in the blocks world. Specifically how would I encode the problem where A is on top of B in the initial state, and I want B to be clear in the goal state?

Qn 2. Explain in words what the "above" predicate stands for.

Qn 3. hand-generate the complete annotated trace of how UCPOP will solve the problem (On A B) & (Above A C) starting
From the initial
state containing On(A table), On(B table), and On(C, table).

* Why is it that the POP algorithm does not ever post the causation and preservation preconditions of an action during planning? Does this mean that the secondary precondition based theory of planning is not applicable to POP?

II. In the very beginning, we said that the general aim of planning is to "synthesize desirable behaviors", and that the goals of planning should be seen as "constraints on behaviors". Explain what we mean formally by this definition? (what _are_ behaviors?) Explain how the usual blocksworld planning is covered by this definition. Given examples of   also 3-4 other types of useful behavioral constraints, and example problems where they could be useful.

III. Explain briefly why the type of reasoning done by TWEAK truth criterioan is not necessary for classical planning. Specifically, which is the assumption of classical planning this makes TWEAK MTC type reasoning unnecessary?

IV. Explain the pros and cons of defining a threat in terms of necessary codesignation (as against possible codesignation)

V. Why is it that checking the consistency of a set of binding constratins (codesignation and non-codesignation) is tractable if we assume infinite domain variables, but becomes intractable for finite domain variable assumption? Which of these assumptions is more reasonable?

VI. Suppose we know that a set of variables have finite domains, but suppose our planning algorithm ignores this fact and checks the consistency by acting as if they have infinite domains. What effect does this have on the soundness and completeness of the planning algorithm?

VII. Why exactly is it that you do not need to backtrack on the order in which the planning goals on the agenda are addressed in the case of plan-space planning?

VIII. Explain in what sense UCPOP/POP algorithms use closed-world assumption? In what sense does it use static-universe assumption?

Give an example of a problem domain where these assumptions don't hold. Explain what sorts of failures will UCPOP have in such domains.

IX. UCPOP attempts to implement a planner for ADL. Pednault's theory of ADL planning allows for incomplete initial states. Explain the parts of UCPOP algorithm that are less general than is required

X. What will be the effect of not using conditional effects in modeling a domain (does it effect completeness? soundness? efficiency? domain-size? operator size?)

What about universal quantification?

XI. Consider the following types of effects: (a) Forall (x) (not(P x))
(b) (not (Forall (x) (P x)))
Can an operator in UCPOP/ADL representation have effects of type "a"? How about effects of type "b"? Explain.

XII. Explain _when_ the ucpop's treatment of disjunctive preconditions will not be enough.

PART B: Thinking questions:
(The answers to the following have not been directly discussed in the class, but can be provided with a little thinking. They thus test whether you are able to apply what you heard in the class).

I. Suppose you are told that three events E1, E2 and E3 are going to occur during the execution of your plan (i.e. between *start* and *end*). All these events are beyond your control and cannot be prevented. You are also told that E1 is going to occur Either before E2 or before E3. Can this precedence relation between the three events be represented in our usual partial ordering representation? If so how? If not, why not? Explain.

II. Here is a blocksworld planning problem with a twist which shows how UCPOP can be creatively extended to deal with unplanned for actions. Suppose in addition to your move(x,y,z) action, you have an event of type Blow-up. This event occurs whenever block A is not on top of block B. Its effect is to shake the table such that all the block stacks are destroyed (and all the blocks fall down to the table).

Model this event using (not(On A B)) as the trigger condition, and the stack-destroying as its effect (you will need to use universally quantified effects).

Suppose your initial state contains A on top of B, and C and D on table. Your goal state is to get (On C D). Explain how you will solve this problem with UCPOP (extending UCPOP in minimal ways).

III. We have talked about the fact that we can start UCPOP/POP with partially specified dummy plans (which contain actions other than *start* and *end*, as well as constraints other than the top level goals).

Can we start state-space planners also (consider both the forward search or the backward search version) with partially specified plans?

In particular, consider (1) the possibility of solving the round-trip problem using state-space planners. (2) the blocks world blow-up problem described in the previous question.

Compare and contrast the plan-space and state-space planners in this regard.

IV. Ability to start with partially filled plans provides for a rudimentary ability to reuse previous plans, and extend them to solve new problems. Suppose you have just found a plan for putting On(A,B) and On(C,D), starting with an initial situation where everthing is on table. Suppose your plan is move(A,T,B) -> move(C,T,B), where T is the Table.)

Suppose you have the new problem where the initial state remains same, but the goal state is On(A,B) & On(B,C) & On(C,D). Suppose further that you want to start your planner off with the plan for the old problem.

Compare and contrast the way state space planners and plan-space planners will reuse the old plan.

V. We observed in the class that UCPOP/ADL representation shortcuts the frame (ramification) problem by ensuring that each action specifies all the predicates which will be made true, either directly or indirectly, by the action. We also said that this is not allow for good software engineering/modularity (since sometimes we would like to keep the invariant laws of the domain, such as the fact that any block which has another block on top of it, is not clear, separate

From the actions).

Explain why it is bad software engineering to not keep such laws separate. How exactly does UCPOP deal with them?

Suppose you are forced to use domain axioms separate
From the action
representation. Speculate on the type of extensions that will be

needed to the various components of UCPOP algorithm.

------End---------

From yqu@enws318.eas.asu.edu Mon Feb 21 16:16:42 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
	[nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <yqu@enws318.eas.asu.edu>
Received:
From enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
	id AA14578; Mon, 21 Feb 94 16:16:42 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
	id AA07660; Mon, 21 Feb 94 16:14:25 MST
Date: Mon, 21 Feb 94 16:14:25 MST
From: yqu@enws318.eas.asu.edu (Subbarao Kambhampati)
Message-Id: <9402212314.AA07660@enws318.eas.asu.edu>
To: plan-class@enws228.eas.asu.edu
Subject: PLAN CLASS NOTE: Feb 14, 1994
Content-Type: X-sun-attachment

X-Sun-Data-Type: text
X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 230

****
Class note for Feb 14 94
CSE 591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE
Taken by: Yong Qu
****

Agenda : Goal Establishment
-TWEAK
-ADL

Now we have extend the representation of a partial to a five-tuple:
P : <T,O,B,ST,L> where:
T: Steps
O: Ordering
B: Binding
ST: Symbol Table
L: auxillary constraints

Here the auxillary constraints can be comprehended as the casual link.
We introduce the ST (Symbol table) to deal with the case that two instances of one same action appear in the partial plan. That is, more than one steps corresponse to one same action, we need a mapping
From steps to
actual actions.
For example, the current partial plan is <T,O,B,ST,L> , where

```
                (S1)+P--|        +Q
                  A1----+------->A2 (S3)
     ___________/       |-------P  \  _______
    |  start   |                     | End  |
    ------------\  +P--|         +R/  -------
                  A1---+--------A2    Q,R
                  (S2) |--------P (S4)
```

T: {S1,S2,S3,S4,Start, End}
ST:{S1-->A1, S2-->A1, S3-->A2, S4-->A2}
L: {   P   P   Q   R   }
      S1--->S3, S2-->S4 , S3--->End , S4--->End

A ground linearization is a fully instantiated total ordering of the steps of partial plan that satisfy all the Ordering and Bindings (Constraints).
On top of that, a ground linearization is a safe ground linearization if an only if it also satisfies all the auxiliary constraints.
As we have defined, a ground operator sequence is a solution to a planning problem, we say that each safe ground linearization corresponse to a ground linearization .
If a safe ground linearization is got, we can add some more steps to the plan without violeting the existing constraints and auxillary constraints, we can still get more solutions to the planning problem. (Through they aren't minimal solution)
Here we define a algorithm of refinement plan:

Algorithm Refinement Plan-1

0. Termination: If a solution can be picked up From P, return it.
1. Goal Selection: select an open goal G of some steps S of P
2. Goal Establishment: (split partial plan into candidate set, make the goal true refinemently)
   refine P into different subplans, each corresponding to a different way of establishment G@S.
3. Consistency check: if the partial plan is inconsistent, prune it.

Refinement Plan-1 is complete if step 2 is complete.
The problem comes to: how to select a goal
From the open goals?

Here we present the Theory of Establishment Refinement:
approach A. Modal True Criteria (MTC) given by Chapman. This is a necessary and sufficient modal truth criterion.
For example, we want to make L true at A2.
  0) First, we check if L is already true at A2?
     if yes, we are done.
     1) if not, find out the action that can satisfy it (make it true).
But , how can we check that L is already true at A2?
Tool1. necessary & sufficient condition to check if a goal is true.
Tool2. use a set of sufficient condition to ensure that goal is true.

There is a straight forward method to check if P is necessary true : just find all the ground linearization and check if they are true. We can check if it's necessary true, necessary false, or possible true (possible false).
But it's clear that this method isn't very useful. We want ways to check if P is necessary true without exhausting list all the ground linearization.

Criteria 1. C is true at S, if there exist a step S' , it's necessary that S'<S (appear before S), S' gives C, and there is no step S" go between S' and S and delete C.
This is a sufficient criteria. There exist some cases that doesn't satisfy criteria. (the problem of white knight).

    Criteria 2. (a necessary and sufficient one).
        for every step S" in current plan which can come between S’ and S,
(S’< S and S’ gives C), and S" delete C, there is another step Sw necessary befo
re S and necessary after S’ such that it make C true.

We can invert criteria 2 to get a algorithm to make something necessary true: that’s what we do in establishment:

If a step S" needs to come before S,

```
         / New step S’          / S’ gives C
        or            such that
       / \ existing S’          \ S’<S
      /
    and
    / \                                    / S"<S’ (promotion)
   /    S’ gives C, S’<S                 so or
  /     make sure not step S" s.t. S’<S"<S     \ S<S" (demotion)
and
  \                    / new Sw           / Sw gives C
    if S" delete C, add               s.t.
                       \ existing Sw       \ (N) (S"<Sw<S)
```

Note: (N) means necessary. I’ll use this notation further on. (I can’t draw the little block using ascii character.)

This criteria is sound and complete. That is, all the ground linearization can be achieved in this partial plan if it’s already a solution. But some solution can’t be returned by UCPOP (ucpop can’t deal with the white knight, its goal establishment is sufficient but not necessary).

We should notice that the goal of planning is to make something true but not checking if something is true. So, sufficient is enough.

By the way, we might don’t need to check true before picking a goal.

What we were talking was planning with variables. It’s chapman who first give the necessary and sufficient condition for establishment of variable version. On the base of the instantiated criteria, we add some more constraints in case of variables:

    criteria 1. ...
        if there exist S, (N) (S’<S) and S’ have an effect P, s.t. (N) (P~C)
(P necessary binds to C).

criteria 2 can be represented as the following chart:

```
            / New step S’       / S’ gives C
           or           such that
          / \ existing S’       \ S’<S
         /
        /
      and    for every effect !Q of S, add NCD
      / \  / to ensure (N) (Q!=C) (separation)
     /   or
    /     \                                    / S"<S’ (promotion)
   /       S’ gives C, S’<S                 so or
  /        make sure not step S" s.t. S’<S"<S    \ S<S" (demotion)
and
  \                    / new Sw           / Sw gives C
    if S" delete C, add               s.t.
                       \ existing Sw       \ (N) (S"<Sw<S)
```

Note: NCD means "noncodesignation"

Here is a example:

```
              !R(u),Q          +P(x)
               ____            _____
            / [    ]---------[     ]\
   _______ /   ----           -----  \_______
   |start|                      Q     [  end ]
   -------\  !P(z),R         +P(y),W /------
           \  ____            _____ /  P(c),m
              [    ]---------[     ]
               ----           -----
                                R
```

For example, in the plan, for P(c) to be necessary true at end, we should:
(x~C or if z~C, we set y~C), or
(y~C of if u~C, we set x~C)
here we use the inverse of criteria 2, change
From "give" to "make", make
it not happen by add the negative of the precondition. So in the case of variables, we add one more branch: separation.

```
    For example, !Q : !on(x,y)
                 C :  on(A,B)
    To separate:  x!=A or y!=B
```

To be sufficient and necessary, we need to add explicitly noncodesignation to add one more branch. But also we can ignore this branch totally, make the planning not necessary but sufficient. That’s because the goal of planning is to garantee the completeness of result. And a sufficient goal establishment is enough, as long as all the goals can be found. Similiarly, we don’t introduce white knight in UCPOP to add the branching factor and add in redundancy. We don’t need to be sure that a particular goal must be true at a particular time phase. (We do need necessary and sufficient at some time, which we will discuss later.) At this time, sufficient is enough. The reason for search is jus t to find a ground operation sequence. To do that, we have better keep it as a partial plan.

Also , we should remember that when in search, we just want to find a goal, (exist), we are not searching for the optimal goal.

Now let’s discuss the complexity of these algorithms.

It’s clear that ground linearization checking is NP complete (suppose n steps, we should totally order these steps, and the sequence is n!).

Using criteria 2. (necessary & sufficient)

a. No variable

That’s polynomial time , to be more precisely, n cube. Stick to the assumption of transitive closure, to check a goal, we can find the the last action that add this condition, then check if all the deletor of this condition don’t fall between them. It require n cube time.

b. with variables

Compare with the case without variable, we need one more work: checking the bindings of the variables.

Now the problem goes back to the finite and infinite domain. If the domain is infinite, then the complexity is polynomial, yet according to transitive closure assumption, the variable of the polynomial is the number of variables instead of number of steps.

If it’s a finite domain, the problem become NP hard. That’s because if variables have finite domain, the noncodesignation might imply some new codesignation, which doesn't appear explicitly. To check that the problem become NP hard.

Yet what we discuss up to know are actions without conditional effects. If there exists actions with conditional effect, the establishment become NP hard. We can image that (some kinds of informal proof):

Suppose we want to check if a particular goal C is true at a point, we check all the actions that add this goal. Suppose there is one step S with conditional effect that add this goal C. To check if this goal C after S, we must make sure first that if the precondition Q of S is true or not when S is executed. Here again the checking of goal Q will repeat the checking procedure of goal C. Through this kind of recursion doesn't necessarily means the problem is NP complete, we can find some method to prove it.

So, checking the true criteria will be NP hard, make the problem intractable, and the planning become very hard. This is why Chapman give up partial planning.

But, it isn't a must that we should use a necessary and sufficient goal establishment algorithm to pick up a not necessary true goal. Also, we can expand searching space to reduce the searching time.

Those problem will be discuss in next class.

X-Sun-Data-Type: default
X-Sun-Data-Description: default
X-Sun-Data-Name: note2
X-Sun-Content-Lines: 230

****

Class note for Feb 14 94

CSE 591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE

Taken by: Yong Qu

****

Agenda : Goal Establishment
- -TWEAK
- -ADL

Now we have extend the representation of a partial to a five-tuple:

P : <T,O,B,ST,L> where:
- T: Steps
- O: Ordering
- B: Binding
- ST: Symbol Table
- L: auxillary constraints

Here the auxillary constraints can be comprehended as the casual link.

We introduce the ST (Symbol table) to deal with the case that two instances of one same action appear in the partial plan. That is, more than one steps corresponse to one same action, we need a mapping From steps to actual actions.

For example, the current partial plan is <T,O,B,ST,L> , where

```
               (S1)+P--|        +Q
                 A1----+------->A2 (S3)

  ___________/       |-------P  \  _______
 |  start   |                     | End  |
 ------------\  +P--|         +R/  -------
               A1---+--------A2    Q,R
               (S2) |--------P (S4)
```

```
T: {S1,S2,S3,S4,Start, End}
ST:{S1-->A1, S2-->A1, S3-->A2, S4-->A2}
L: {     P          P          Q           R      }
      S1--->S3,  S2-->S4 , S3--->End , S4--->End
```

A ground linearization is a fully instantiated total ordering of the steps of partial plan that satisfy all the Ordering and Bindings (Constraints).

On top of that, a ground linearization is a safe ground linearization if an only if it also satisfies all the auxiliary constraints.

As we have defined, a ground operator sequence is a solution to a planning problem, we say that each safe ground linearization corresponse to a ground linearization .

If a safe ground linearization is got, we can add some more steps to the plan without violeting the existing constraints and auxillary constraints, we can still get more solutions to the planning problem. (Through they aren't minimal solution)

Here we define a algorithm of refinement plan:

Algorithm Refinement Plan-1

0. Termination: If a solution can be picked up From P, return it.
1. Goal Selection: select an open goal G of some steps S of P
2. Goal Establishment: (split partial plan into candidate set, make the goal true refinemently)
   refine P into different subplans, each corresponding to a different way of establishment G@S.
3. Consistency check: if the partial plan is inconsistent, prune it.

Refinement Plan-1 is complete if step 2 is complete.

The problem comes to: how to select a goal From the open goals?

Here we present the Theory of Establishment Refinement:

approach A. Modal True Criteria (MTC) given by Chapman. This is a necessary and sufficient modal truth criterion.

For example, we want to make L true at A2.

0) First, we check if L is already true at A2?
   if yes, we are done.
   1) if not, find out the action that can satisfy it (make it true).

But , how can we check that L is already true at A2?

Tool1. necessary & sufficient condition to check if a goal is true.

Tool2. use a set of sufficient condition to ensure that goal is true.

There is a straight forward method to check if P is necessary true : just find all the ground linearization and check if they are true. We can check if it's necessary true, necessary false, or possible true (possible false).

But it's clear that this method isn't very useful. We want ways to check if P is necessary true without exhausting list all the ground linearization.

Criteria 1. C is true at S, if there exist a step S' , it's necessary that S'<S (appear before S), S' gives C, and there is no step S" go between S' and S and delete C.

This is a sufficient criteria. There exist some cases that doesn't satisfy criteria. (the problem of white knight).

Criteria 2. (a necessary and sufficient one).
for every step S" in current plan which can come between S' and S, (S'< S and S' gives C), and S" delete C, there is another step Sw necessary before S and necessary after S' such that it make C true.

We can invert criteria 2 to get a algorithm to make something necessary true: that's what we do in establishment:
If a step S" needs to come before S,

```
         / New step S'          / S' gives C
        or             such that
       / \ existing S'          \ S'<S
      /
    and
    / \                                    / S"<S' (promotion)
   /    S' gives C, S'<S                  so or
 /      make sure not step S" s.t. S'<S"<S     \ S<S" (demotion)
and
  \                     / new Sw              / Sw gives C
    if S" delete C, add                 s.t.
                        \ existing Sw         \ (N) (S"<Sw<S)
```

Note: (N) means necessary. I'll use this notation further on. (I can't draw the little block using ascii character.)

This criteria is sound and complete. That is, all the ground linearization can be achieved in this partial plan if it's already a solution. But some solution can't be returned by UCPOP (ucpop can't deal with the white knight, its goal establishment is sufficient but not necessary).
We should notice that the goal of planning is to make something true but not checking if something is true. So, sufficient is enough.
By the way, we might don't need to check true before picking a goal.

What we were talking was planning with variables. It's chapman who first give the necessary and sufficient condition for establishment of variable version. On the base of the instantiated criteria, we add some more constraints in case of variables:

criteria 1. ...
if there exist S, (N) (S'<S) and S' have an effect P, s.t. (N) (P~C) (P necessary binds to C).

criteria 2 can be represented as the following chart:

```
            / New step S'        / S' gives C
           or          such that
          / \ existing S'        \ S'<S
         /
        /
      and    for every effect !Q of S, add NCD
      / \  / to ensure (N) (Q!=C) (separation)
     /   or
    /      \                                      / S"<S' (promotion)
   /        S' gives C, S'<S                    so or
/           make sure not step S" s.t. S'<S"<S    \ S<S" (demotion)
and
  \                     / new Sw              / Sw gives C
    if S" delete C, add                 s.t.
                        \ existing Sw         \ (N) (S"<Sw<S)
```

Note: NCD means "noncodesignation"

Here is a example:

```
          !R(u),Q         +P(x)
           ____           _____
        / [    ]---------[     ]\
_______ /   ----            -----  \_______
|start|                      Q     [  end ]
-------\  !P(z),R         +P(y),W /------
        \  ____           _____/  P(c),m
          [    ]---------[     ]
           ----           -----
                            R
```

For example, in the plan, for P(c) to be necessary true at end, we should:
(x~C or if z~C, we set y~C), or
(y~C of if u~C, we set x~C)
here we use the inverse of criteria 2, change
From "give" to "make", make it not happen by add the negative of the precondition. So in the case of variables, we add one more branch: separation.
For example, !Q : !on(x,y)
C : on(A,B)
To separate: x!=A or y!=B
To be sufficient and necessary, we need to add explicitly noncodesignation to add one more branch. But also we can ignore this branch totally, make the planning not necessary but sufficient. That's because the goal of planning is to garantee the completeness of result. And a sufficient goal establishment is enough, as long as all the goals can be found. Similiarly, we don't introduce white knight in UCPOP to add the branching factor and add in redundancy. We don't need to be sure that a particular goal must be true at a particular time phase. (We do need necessary and sufficient at some time, which we will discuss later.) At this time, sufficient is enough. The reason for search is just to find a ground operation sequence. To do that, we have better keep it as a partial plan.
Also , we should remember that when in search, we just want to find a goal, (exist), we are not searching for the optimal goal.

Now let's discuss the complexity of these algorithms.
It's clear that ground linearization checking is NP complete (suppose n steps, we should totally order these steps, and the sequence is n!).
Using criteria 2. (necessary & sufficient)
a. No variable
That's polynomial time , to be more precisely, n cube. Stick to the assumption of transitive closure, to check a goal, we can find the the last action that add this condition, then check if all the deletor of this condition don't fall between them. It require n cube time.
b. with variables
Compare with the case without variable, we need one more work: checking the bindings of the variables.
Now the problem goes back to the finite and infinite domain. If the domain is infinite, then the complexity is polynomial, yet according to transitive closure assumption, the variable of the polynomial is the number of variables instead of number of steps.

If it's a finite domain, the problem become NP hard. That's because if variables have finite domain, the noncodesignation might imply some new codesignation, which doesn't appear explicitly. To check that the problem become NP hard.

Yet what we discuss up to know are actions without conditional effects. If there exists actions with conditional effect, the establishment become NP hard. We can image that (some kinds of informal proof):

Suppose we want to check if a particular goal C is true at a point, we check all the actions that add this goal. Suppose there is one step S with conditional effect that add this goal C. To check if this goal C after S, we must make sure first that if the precondition Q of S is true or not when S is executed. Here again the checking of goal Q will repeat the checking procedure of goal C. Through this kind of recursion doesn't necessarily means the problem is NP complete, we can find some method to prove it.

So, checking the true criteria will be NP hard, make the problem intractable, and the planning become very hard. This is why Chapman give up partial planning.

But, it isn't a must that we should use a necessary and sufficient goal establishment algorithm to pick up a not necessary true goal. Also, we can expand searching space to reduce the searching time.

Those problem will be discuss in next class.

From ihrig@enws318.eas.asu.edu Wed Feb 23 14:15:35 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil])
Status: RO
Return-Path: <ihrig@enws318.eas.asu.edu>
Received:
From enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA15660; Wed, 23 Feb 94 14:15:35 MST
Received:
From enws322.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA08080; Wed, 23 Feb 94 14:13:10 MST
Date: Wed, 23 Feb 94 14:13:10 MST
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
Message-Id: <9402232113.AA08080@enws318.eas.asu.edu>
To: plan-class@enws228

****

Class note for Feb 16 94

CSE 591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE

Taken by: Laurie H. Ihrig

(notetakers comments in brackets)

****

Agenda : Goal Establishment
- -TWEAK
- -ADL

As discussed previously:

Algorithm Refinement Plan-1

0. Termination: If a solution can be found in P, return it.
1. Goal Selection: select an open goal G of some step S of P
2. Goal Establishment:
   generate refinements of P, each corresponding to a different
   way of establishing G@S.
3. Consistency check: if the partial plan is inconsistent, prune it.

Historically, there have been two methods of establishment:

Theories of Establishment Refinement:
1. Invert a necessary and sufficient Modal Truth Criterion (MTC)
   -
From Chapman
- -includes white knights
- -includes separation

- -this method provides completeness in terms of partially ordered plans
- -but we don't need completeness in terms of PO plans, only in terms of ground operator sequences
- -therefore we can used second method below:

2. Add sufficient constraints to make a given condition true
   -
From Pednault

There are some instances where you would require 1, for example if we want to deal with events as well as actions. Consider that events are actions over which the planning agent has no control (for example, actions performed by the agent's spouse)

Suppose that part of the problem specification includes a set of such events and that these events are partially ordered. Establishing a goal requires knowing if a certain condition will be true at a certain point in time. Suppose we have the following ordering for events E1 to E4:

```
                          (unrested)
                ____       _____                     E3 = sleep
             / [ E1 ]---------[ E2  ]\               E2 = party
    ______ /    ____       _____  \______            E4 = exam
   [start]                         [ end ]
    ______                          ______
          \                        /
           \  ____       _____    /
             [ E3 ]---------[  E4 ]
              ____          _____
                           (rested)
```

Suppose that E4 is an exam, and you want to know if you will be rested before the exam. But E2 is a party which will cause you to become unrested.

The problem then is that you only have a partial order for events. Moreover, there is no way you can establish an ordering since these events are out of your control.
ie. If you could order events, you could always demote the party, but since you can't demote the party (since this is under the control of the spouse) planning will only be complete if you know the necessary and sufficient conditions, ie use method 1, MTC.

This is the temporal projection problem, which is discussed in
Dean 'Temporal Data Base Management' in Readings in Planning.

In classical planning, it is assumed that all events are under the control of the agent. The projection problem does not occur if you make this assumption, or if you deal only with scheduled events, ie. totally ordered events. It is also possible to deal with scheduled events that have trigger conditions, for example, if C then E. In this way the planning agent has some control over the effects of an event.

The complexity of the projection problem is discussed in
Dean and Boddy Reasoning About Partially Ordered Events in AIJ
(I can't find it in the readings. But the reference is:

Thomas Dean and Mark Boddy
Reasoning about Partially Ordered Events
Artificial Intelligence 36(3):375-399, 1988)

-The Projection Problem:
Given a set of partially ordered events, will condition C be true.

-This problem is intractable, but there is a sound but incomplete projection algorithm.

Action Description Language (ADL)
____

The theory on which the proof of completeness for UCPOP is based comes
From Pednault (a Canadian!) UCPOP is an implementation based on Pednault's ADL/Secondary Precondition Theory. The technical paper on UCPOP has a proof of completeness and soundness. UCPOP implements only a subset of ADL.

Question?
Suppose that you have a totally ordered plan:

```
  ____      _____       ____        _____
 [ A1 ]---------[ A2 ]---------- [ A3 ]---------  [ A4 ]
  ____      _____       ____        _____
                                     (P)
```

What will be required so that P will be true?

Assume actions are state transformation functions, ie. A={<s,t>}
Can you model actions with conditional effects under this assumption? Yes. For example, action A where:

```
    If R then S
    If Q then P
  A
```

If state s is such that s entails R then t entails S
ie. s |= R -> t |= S, and
s |= Q -> t |= P

What about actions with nondeterministic effects?
-these are not allowed in UCPOP

eg. tossing a coin has effect H V T

Actions as state transformation functions cannot deal with nondeterministic effects, since actions are functions they can only map a state into a single state.

Therefore, assuming all actions are state transformation functions,
P is true at A if
1. There is some action A' before A which causes P to be true, and
2. Every action A" between A' and A preserves P.

Suppose actions have conditional effects, eg if Q then P. Then it is also required to make Q true before P. You must add the causation preconditions as one of the secondary preconditions.

Therefore P is true at A if
1. There is some action A' before A such that

```
          ___  P
         \          is true immediately before A'
         /___ A'
```

That is, the causation conditions are true.

2. For very action A" between A' and A

```
         ______ P
           ||          is true immediately before A"
           ||   A"
```

That is, the preservation conditions are true.

For example, if there is an action A" which has the effect: If R then not P

then

```
         ______ P
           ||        = not R
           ||   A"
```

Therefore we can resolve a threat by promotion or demotion of the action, but also by adding a secondary precondition, eg. not R.

To make P true at A:

```
            / add step A' \            /  bindings so A' gives P
            or             \           and
          / \ existing A' __add        \  ordering A'< A
         /                             
        /                              \
       /                                  ___ P
      /                                  \
     /                                   /        to agenda
    /                                     ___
   /                                          A'
and
```

```
   \
    \                                            / A"<A' (promotion)
    for all A" s. t. possibly  A'<A"<A add   or
                                             \ A <A"  (demotion)

                                             \
                                                ______ P
                                                  ||       to agenda
                                                  ||   A"
```

Assume actions are state transformation functions, ie. A={<s,t>},
what are the weakest conditions that must be true before an action
such that P will be true in the resulting state?

```
  -1
A    (P) is called the regression of P over A.  It is a formula
```

which if true in a state s, implies that P is true in state t.

By definition:

```
       -1
s |= A    (P)     iff      t  |=  P
```

Example:
If action is:

```
            If R then not P
            If Q then P
           A
```

then

```
       -1
      A  (P)   = Q V ( P ^ not R)
```

This is the weakest condition that needs to be true before A such that
P will be true after A.

In the above example:

```
  ___  P
 \
 /         = Q
  ___
       A


 ______ P
   ||      = not R
   ||   A


1.
  ___  P
 \                  -1
 /            |=   A (P)
  ___
       A
```

```
2.
                          ___  P
           -1             \
 not P ^    A  (P)  |=    /___
                               A'

3.
      ______ P          -1
        ||       |=  A (P)
P ^     ||   A

4.
      -1
P ^  A (P)   |=    ______ P
                     ||
                     ||   A



              (I think you also need    V      ___  P    )
                                               \
                                               /__  A
```

Pednault shows that ADL is a subset of situation calculus. It is the
largest subset for which you don't have to use frame axioms. ADL
provides a way of getting regression of formulas
by rules like:

```
              -1                              -1       -1
             A (P ^ Q)   is equivalent to   A  (P)  ^ A (Q)
```

This theory is used in UCPOP in goal establishment,
but UCPOP is not full ADL. It has to make an additional assumption
that the initial world state is complete,
eg. it can't do the bomb in the toilet problem.


```
From rus@seine.eas.asu.edu  Sun Feb 27 22:16:02 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <rus@seine.eas.asu.edu>
Received:
From seine.eas.asu.edu (enws293.EAS.ASU.EDU) by parikalpik.eas.asu.edu (4.1/SMI-
4.1)
        id AA04186; Sun, 27 Feb 94 22:16:02 MST
Received: by seine.eas.asu.edu (4.1/SMI-4.1)
        id AA20981; Sun, 27 Feb 94 22:24:55 MST
Date: Sun, 27 Feb 94 22:24:55 MST
From: rus@seine.eas.asu.edu (Rus Ioana)
Message-Id: <9402280524.AA20981@seine.eas.asu.edu>
To: plan-class@enws228.eas.asu.edu
Subject: planning class notes Feb 20
Content-Type: X-sun-attachment

----------
X-Sun-Data-Type: text
```

X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 242

NOTES FOR 20 FEBRUARY CLASS

Ioana Rus

Agenda:

1. ADL vs. UCPOP
2. Case study: TWEAK
3. Algorithm Refine-Plan-2

1. ADL vs UCPOP

Here is a simplified version of the bomb-in-the-toilet problem that ADL based planning theory can solve but UCPOP cannot.

Problem

Initial state: P V Q
Final state: R
Actions: (two actions are available)

```
    A1
     precond: nil
     effects: If P then R

    A2
     precond: nil
     effects: If Q then R
```

We know that the following plan

  A1 --> A2

will solve this problem (i.e., if you execute A1 and then A2, you are guaranteed to have R true in the resulting situation).

Qn 1. Explain how the secondary preconditions based theory of planning can be used to (a) check that this plan is correct and (b) make this plan.

Qn 2. What are the reasons UCPOP algorithm is not able to deal with this problem?

Answers:

```
             if P        if Q
            then R      then R
P V Q --------A1--------A2---------End
                                  R

 -1
A2 (R)=QVR

 -1        -1         -1
A1 (QVR)=A1 (Q) V A1 (R)= QVPVR

      -1  -1
if I |= A1 (A2 (phi)), then phi is true after the execution of the plan.
      |
      |
(that means initial state entails...)
```

Let us see what are conditions for a plan A1, A2, ..., An to make phi true, after it is executed.

EXECUTABILITY:

```
I |= PI
       A1

       -1
I |= A1 (PI   )
           A2

.....

       -1  -1  -1         -1
I |= A1 (A2 (A3 (....(An-1 (PI  ))..)
                              An
```

The regression of phi over action A = the weakest condition required before A, such that phi is true after A.

```
       -1         -1  -1
I |= A1 (...(An-1 (An (phi))..)
```

Can convert proof of correctness of a plan into a theorem proving.

The second precondition theory - for phi to be true at an action A, there must be some action A’, A’<A that causes phi and every action A’<A’’<A must preserve phi.
Unless you know what action makes phi true, cannot distinguish differences between causation precondition and preservation.
Regression can be used to prove that this plan is correct, and also to generate it.

```
      A2----------A1---------End
PVQVR           PVR         R



---A1
\       -1
/      A1 (R)= PVR
---R


---A2
\       -1          -1       -1
/      A2 (PVR) = A2 (P) V A2 (R) = PVQVR
---PVR
```

For ADL actions cannot have non-deterministic effects.
The initial state gives PVQVR; as PVQ |= PVQVR, we have R at the final state.

In UCPOP cannot do that, because:

- it is to conservative:

```
PVQ----A1----A2-----R
            Q
```

If A2 gives R to the final state, Q has to be true before A2.

UCPOP can be changed, such that causation precodition=regression, but it still

- splits a disjunctive precondition,

so, THE REAL DIFFERENCE between ADL and UCPOP is that UCPOP assumes completely specified initial states.

When have completely specified initial state, can:
- differentiate causation From regression (and can make it a formula simpler then regression);
- work on a disjunction by working on the disjunct parts separately.

For any action A, $A^{-1}(F)$ - where F is a formula-can be distributed over the components of the formula.

For actions with nondeterministic effects (like action A1=tossing a coin: the effects will be H(ead) V T(ail). We want to know what are the weakest conditions before A1, such that T will be true after A1.
We can look regression as Weakest SUFFICIENT conditions (no necessary)

$A1^{-1}(T)=T$
$A1^{-1}(H)=H$
$A1^{-1}(TVH)=True$ which is different From TVH

Adopting this definition of regression, all its distributive properties will be no longer valid.

If the initial state is completely specified, then causation and preservation are subformula of the regression.

Establishment in UCPOP is adding:
- ordering
- bindings
- preconditions
- links.

As a solution constructor, can take the following:
1. take a ground linearization of the plan;
2. check if satisfies ADL correctness principles forward execution, and if there is a plan, the termination will be successful.

There may be branches in the search space in which establishment is do and undo, but in the whole search space there is at least one branch which will lead to a solution. So, having a complete search algorithm, will have completness (for plans).

Case Study - TWEAK (Chapman)

- Theory of establishment is different From ADL.
- It cares for efficiency.
- TWEAK can work on the same open condition arbitrary many times, within the same branch.
- TWEAK is not systematic.

For seeing TWEAK 's redundancy and doing and undoing establishments, see figure 4, in 'Planning as Refinement Search...', by S. Kambhampati.

Completness: - if there is a plan, it will be found. If no plan exists, the algorithm may loop forever.

BOOKKEEPING strategies - 3 ideas:

Idea 1 - GOAL PROTECTION - to work only once on each condition, and take it out From the agenda. When agenda is empty and the solution is not a candidate one, then can abandon that branch.

```
Goal        / whenever a goal already established becomes necessarily
protection /  untrue, quit that part!   or
           \
            \ when work on a goal, remove it
From the agenda (doing
                  this, the efficiency can be smaller)
```

Idea 2 - PROTECT THE INTERVAL

```
    g
A1-----A2    (A1 gives g to A2)
```

if an action A3 (A1<A3<A2) deletes g, that path can be quit and backtrack.
Protection of intervals can be seen like goals maintenance (protect causal links)

Idea 3 - CONTRIBUTOR PROTECTION -for having systematicity

```
    g
A1------A2
|       |        an action A1 gives g to A2
|--A3---|         - if A3 comes between A1 and A2 and deletes
|       |           g, it is not allowed
|       |         - if A4 can come between A1 and A2 and also
|   g   |           adds g, that path will be quit
|--A4---|
                        because
                the contributor that gives goal g is protect!
```

That leads to systematicity: if P1 and P2 are plans in different branches, then no ground operator sequence S will be consistent with both P1 and P2.

Algorithm Refine-Plan-2

0. Termination
1. Goal selection
2. Goal establishment
   2.1. Bookkeeping - add auxiliary constraints (causal links) to remember establishment decisions
   This is a new step!
3. Consistency checking

X-Sun-Data-Type: default
X-Sun-Data-Description: default
X-Sun-Data-Name: r.pla
X-Sun-Content-Lines: 242

NOTES FOR 20 FEBRUARY CLASS

Ioana Rus

Agenda:

1. ADL vs. UCPOP
2. Case study: TWEAK
3. Algorithm Refine-Plan-2

1. ADL vs UCPOP

Here is a simplified version of the bomb-in-the-toilet problem that ADL based planning theory can solve but UCPOP cannot.

Problem

Initial state: P V Q
Final state: R
Actions: (two actions are available)

```
    A1
     precond: nil
     effects: If P then R

    A2
     precond: nil
     effects: If Q then R
```

We know that the following plan

A1 --> A2

will solve this problem (i.e., if you execute A1 and then A2, you are guaranteed to have R true in the resulting situation).

Qn 1. Explain how the secondary preconditions based theory of planning can be used to (a) check that this plan is correct and (b) make this plan.

Qn 2. What are the reasons UCPOP algorithm is not able to deal with this problem?

Answers:

```
                if P       if Q
               then R     then R
P V Q --------A1--------A2---------End
                                  R
```

$A2^{-1}(R)=QVR$

$A1^{-1}(QVR)=A1^{-1}(Q) \ V \ A1^{-1}(R)= QVPVR$

if $I \models A1^{-1}(A2^{-1}(phi))$, then phi is true after the execution of the plan.
(that means initial state entails...)

Let us see what are conditions for a plan A1, A2, ..., An to make phi true, after it is executed.

EXECUTABILITY:

$I \models PI_{A1}$

$I \models A1^{-1}(PI_{A2})$

.....

$I \models A1^{-1}(A2^{-1}(A3^{-1}(....(An-1^{-1}(PI_{An}))..)$

The regression of phi over action A = the weakest condition required before A, such that phi is true after A.

$I \models A1^{-1}(...(An-1^{-1}(An^{-1}(phi))..)$

Can convert proof of correctness of a plan into a theorem proving.

The second precondition theory - for phi to be true at an action A, there must be some action A', A'<A that causes phi and every action A'<A''<A must preserve phi.
Unless you know what action makes phi true, cannot distinguish differences between causation precondition and preservation.
Regression can be used to prove that this plan is correct, and also to generate it.

```
      A2----------A1---------End
PVQVR           PVR         R
```

```
---A1
\       -1
/      A1 (R)= PVR
---R
```

```
---A2
\       -1          -1       -1
/      A2 (PVR) = A2 (P) V A2 (R) = PVQVR
---PVR
```

For ADL actions cannot have non-deterministic effects.
The initial state gives PVQVR; as PVQ |= PVQVR, we have R at the final state.

In UCPOP cannot do that, because:

- it is to conservative:

```
PVQ----A1----A2-----R
            Q
```

If A2 gives R to the final state, Q has to be true before A2.

UCPOP can be changed, such that causation precodition=regression, but it still

- splits a disjunctive precondition,

so, THE REAL DIFFERENCE between ADL and UCPOP is that UCPOP assumes completely specified initial states.

When have completely specified initial state, can:
- differentiate causation From regression (and can make it a formula simpler then regression);
- work on a disjunction by working on the disjunct parts separately.

For any action A, $A^{-1}(F)$ - where F is a formula-can be distributed over the components of the formula.

For actions with nondeterministic effects (like action A1=tossing a coin: the effects will be H(ead) V T(ail). We want to know what are the weakest conditions before A1, such that T will be true after A1.
We can look regression as Weakest SUFFICIENT conditions (no necessary)

$A1^{-1}(T)=T$

$A1^{-1}(H)=H$

$A1^{-1}(TVH)$=True which is different From TVH

Adopting this definition of regression, all its distributive properties will be no longer valid.

If the initial state is completely specified, then causation and preservation are subformula of the regression.

Establishment in UCPOP is adding:
- ordering
- bindings
- preconditions
- links.

As a solution constructor, can take the following:

1. take a ground linearization of the plan;
2. check if satisfies ADL correctness principles forward execution, and if there is a plan, the termination will be successful.

There may be branches in the search space in which establishment is do and undo, but in the whole search space there is at least one branch which will lead to a solution. So, having a complete search algorithm, will have completness (for plans).

Case Study - TWEAK (Chapman)

- Theory of establishment is different From ADL.
- It cares for efficiency.
- TWEAK can work on the same open condition arbitrary many times, within the same branch.
- TWEAK is not systematic.

For seeing TWEAK 's redundancy and doing and undoing establishments, see figure 4, in 'Planning as Refinement Search...', by S. Kambhampati.

Completness: - if there is a plan, it will be found. If no plan exists, the algorithm may loop forever.

BOOKKEEPING strategies - 3 ideas:

Idea 1 - GOAL PROTECTION - to work only once on each condition, and take it out From the agenda. When agenda is empty and the solution is not a candidate one, then can abandon that branch.

```
Goal        / whenever a goal already established becomes necessarily
protection /  untrue, quit that part!   or
           \
            \ when work on a goal, remove it
```
From the agenda (doing this, the efficiency can be smaller)

Idea 2 - PROTECT THE INTERVAL

```
   g
A1-----A2    (A1 gives g to A2)
```

if an action A3 (A1<A3<A2) deletes g, that path can be quit and backtrack.
Protection of intervals can be seen like goals maintenance (protect causal links)

Idea 3 - CONTRIBUTOR PROTECTION -for having systematicity

```
    g
A1------A2
|       |        an action A1 gives g to A2
|--A3---|         - if A3 comes between A1 and A2 and deletes
|       |           g, it is not allowed
|       |         - if A4 can come between A1 and A2 and also
|   g   |           adds g, that path will be quit
|--A4---|
                        because
```

the contributor that gives goal g is protect!

That leads to systematicity: if P1 and P2 are plans in different branches, then no ground operator sequence S will be consistent with both P1 and P2.

```
        Algorithm Refine-Plan-2
        -----------------------

0. Termination
1. Goal selection
2. Goal establishment
        2.1. Bookkeeping - add auxiliary constraints (causal links)
                           to remember establishment decisions
        This is a new step!
3. Consistency checking
```

```
From AZEWS@ACVAX.INRE.ASU.EDU  Mon Feb 28 20:37:13 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <AZEWS@ACVAX.INRE.ASU.EDU>
Received:
From ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA05309; Mon, 28 Feb 94 20:37:13 MST
Received:
From ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF V4.2-13 #2382) id
 <01H9FKDSRUY800J887@ACVAX.INRE.ASU.EDU>; Mon, 28 Feb 1994 20:34:03 MST
Date: Mon, 28 Feb 1994 20:34:02 -0700 (MST)
From: AZEWS@ACVAX.INRE.ASU.EDU
Subject: class notes of monday, feb 28 --- in a jiffy !
To: plan-class@parikalpik.eas.asu.edu
Message-Id: <01H9FKDSXR4Y00J887@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"plan-class@parikalpik.eas.asu.edu"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT
```

```
********************************************************************
*********     Notes for 28 Feb 1994         Ed Smith     ***********
********************************************************************
```

Administrative Notes:
* Have questions about the sanity check for next time (Wed).
* Make-up class period: Friday, 4 Mar ERC 393 15:00 to 16:40.
* For Wednesday: Read: NONLIN, FORBIN, NOAH, SIPE papers

```
********************************************************************
LAST TIME:
********************************************************************
```

We talked about bookkeeping. Bookkeeping supports the objective of REFINEMENT SEARCH, wherein search is defined as the process of splitting a node into a number of nonempty nodes in such a way that the child nodes are mutually uncommon; i.e., they share no children.

Book keeping methods we saw last time are:

1 Agenda (weak ;-don't backtrack till agenda empty)

2 Protection

3 Contributor Protection (strongest protection)

Last time we also saw the equation:

Fd = K * Rd / kd

where:
Fd == # of fringe nodes @ level d of search tree
K == total # of ground operator sequences
Rd == average redundancy of the dth level of the search tree
kd == average size of the nodes (candidate sets)

Bookkeeping strategies reduce Rd; attacking Rd sooner rather than later is better, all other things being equal.

For Contributor Protection, Rd == 1 (:-))

Doing a consistency check ---> no empty nodes will get added to the search tree.

```
********************************************************************
IF WE ASSUME THE COST OF CONTRIBUTOR PROTECTION IS SMALL:
********************************************************************
```

Impact on unsolvable problems:
  With Protection:
  * overall search space WE EXPLORE will be smaller since Rd = 1.

  Without Protection:
  * the search space explored will be larger.

Impact on solvable problems:
  * if density of solutions in search space is high, no difference.
  * if density of solutions in search space is low, we approach the unsolvable problem situation.

```
********************************************************************
so, PROTECTION STRATEGIES ARE LIKE INSURANCE:
(log-linear performance plots of time vs. complexity bore this out)
********************************************************************
```

They will help us in situations without solutions, and in situations with low solution densities. Protection strategies cost something to implement, and may even steer the search AWAY
From SOME
solutions. Consider a protection scheme P:

At each node, we have:
    P = { Pe , Pp }
where:
    Pe are the establishments
    Pp are the (imposed) protection requirements

```
  S  <--- the start node.
  |
  |  ( in this search tree through the search space, I have )
  O  ( not shown the branching paths; just a single path to )
  |  ( a non-unique goal and one other sub-branch.          )
  |
  O
  |\
  | \  <--- Start of the next sub-branch P will pursue.
```

```
  O <--- Backtracking forced by P, since Pp violated.
  |
  |
  O  <--- But without violating { Pe }, a "dumber" search gets here
  |       without any hitch !
  |
  Gn  <--- a goal we CANNOT FIND with protection scheme P - the
           dumber search can find this goal and (presumably)
           terminate sooner !!!
```

****
WEAKER STRATEGIES MAY AVOID PROBLEMS
****
(like the missed goal above), but Rd will be higher, since the search will be (to some degree) not completely systematic. Weaker strategies are ok for problems with a not-too-small goal density, wherein a goal would probably be encountered before a lot of backtracking through a (somewhat larger) search tree would be done.

****
ANOTHER PROTECTION STRATEGY IS CALLED MULTICONTRIBUTOR PROTECTION:
****
We have considered an establishment to go
From a single provider
action to a consumer action. This can be generalized by allowing the provider to be a (partially ordered) SET of actions.

We used to write:

```
           c
    (S') ------>(S)   where S' and S are actions
```

But now we will write:

```
              c
    {S'''} ------>(S)   where S is an action, and S""" is a SET.
```

* We can avoid backtracking by changing S''' in the event that another provider of c is introduced after S'

* This scheme makes sure S receives c, so unlike the AGENDA method, S cannot wind up deprived of c.

****
WHY DOES UCPOP BOTHER TO DO CONFLICT RESOLUTION, ANYWAY?
****
UCPOP makes the search tree bigger by inflating the search space when a possible conflict is identified and the possible conflicting action is split (promoted / demoted) resulting in two more nodes. The resulting nodes can be checked for consistency by checking only orderings and bindings - causal links needn't be checked in UCPOP!

****
*********   End of Notes for 28 Feb 1994   ***********
****

From rao Thu Mar 3 18:14:16 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA08898; Thu, 3 Mar 94 18:14:16 MST
Message-Id: <9403040114.AA08898@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Pre-ordering strategies-- a clearer story
Date: Thu, 3 Mar 94 18:14:16 MST
Reply-To: rao@asuvax.asu.edu

Here is a more complete story on the pre-ordering strategies.

Pre-ordering strategies use a localized interaction test to check if a pair of (unordered) steps should be ordered with respect to each other. They will refine the plan until no pair of interacting steps are left unordered.

The "interacts" routine uses the local properties of the steps to decide whether they interact

Here are a variety of ways of defining this routine, in increasing order of eagerness to order steps.

1. PI:

   s1 interacts with s2 if s1 has a precondition c and s2 has an effect that negates c

   Using PI will ensure that the partial plans in the search queue will either be necessarily safe or necessarily unsafe with respect to every protection interval based causal link (of the kind used in POP and UCPOP)

   Thus consistency check (with respect to protection intervals) can be done by simply checking the safety of a single linearization

2. CPI:

   s1 interacts with s2 if s1 has a precondition c and s2 has an effect that either negates c OR ADDS c

   Using CPI will ensure that the partial plans in the search queue will either be necessarily safe or necessarily unsafe with respect to every contributor protection based causal link (of the kind used in SNLP and UCPOP)

3. UA:
   s1 interacts with s2 if s1 has a precondition c and s2 has an effects that eitehr negates c or adds c

   OR
   s1 has an effect c and s2 has an effect that negates c

   Using UA will not only ensure that the partial plans in the search queue will be necessarily safe or necessarily unsafe, BUT WILL ALSO ENSURE that the partial plans are unambiguous.

   When a plan is unambiguous, every condition is either necessarily

```
    true or necessarily untrue (irrespective of whether or not there
    is a causal link supporting it already).

    Thus if the plans are unambiguous, we can check necessary truth of
    a condition in polynomial time and only work on those conditions
    that are not necessarily true (i.e., are necessarily FALSE).

    In the literature, unambiguity was first introduced to show a
    systematicity like property. But, it turns out that unambiguity
    has nothing to do with systematicity. Systematicity depends only
    on whether or not the planner uses causal links based on
    contributor protection.

4. TO:
    s1 always interacts with s2

    This is the most conservative of interaction definitions, and it
    will wind up ordering every pair of unordered steps, thus making each
    partial plan in the search queue a totally ordered one.

    Thus, it not only gives all the properties of PI, CPI and UA, but also
    makes the planning totally ordered plan-space planning.

It should be easy to see that the branching factor introduced by PI
will be less than that introduced by CP, which will be less than that
introduced by UA, which in turn will be less than that introduced by
TO. As the branching factor increases, so does the search space size
(and the impact of the chemistry between b_t and b_e factors on
performance).

Suppose we want to compare the pre-ordering strategies with conflict
resolution strategies.

The "interacts" routine can be made a three-place function, with the
third argument being the plan (which is ignored in the four
interaction definitions above). Then we can also cover conflict
resolution using the same notion. Specifically, conflict resolution
strategies consider a pair of steps to interact only when they are
together taking part in the violation of an auxiliary constraint
(causal link).

0.1 PCR
   s1 and s2 interact if there is a link L:s1--c-->s3 and
   s2 has an effect that negates c

0.2 CCR
   s1 and s2 interact if there is a link L:s1--c-->s3 and
   s2 has an effect that negates c or adds c


Note that PCR and CCR are less eager than PI and CI respectively to
order steps (in otherwords, if two steps interact according to PCR,
then they interact according to PI too; verify this!). In particular,
PCR and CCR wait until there is a causal link involving s1 and s2
before considering them to be interacting.

Given this generalization, it is easy to compare the pre-ordering
strategies with conflict resolution strategies:

1. Preordering strategies may order a pair of steps that will never be
   ordered with respect to the conflict resolution strategies (e.g.
   ITO strategy will order every pair of steps even if they will never
   take part in an actual conflict)

2. More importantly, preordering strategies will order a pair of steps
   earlier than they will be ordered by the conflict resolution
   strategies. This means that the preordering strategies have a
   larger branching factor at the top of the search tree, increasing
   the search space size.

   Moreover, since we have no way of knowing which of the orderings
   will lead to solutions, it is better to avoid premature orderings
   (especially those introduced only to aid tractability of
   refinement).


Based on this, the general hypothesis is that if you _have_ to do
tractability refinements, do the conflict resolution based
tractability refinements, rather than pre-ordering refinement based
ones, since the former are more sensitive to the role played by the
steps in the plan, while the latter base their analysis just on the
steps.

(about the only utility of pre-ordering based refinements is that some
of them, eg UA and TO can make necessary truth computation polynomial,
which can be used in goal selection and termination steps. But even
this may be somewhat of a weak claim. Specifically, it is not
necessary to check "necessary truth" of a plan to terminate. Since we
only need one correct linearization, we can just check a random
linearization of the plan.

This leaves us only with the possible advantages in goal selection
The idea here is to work on goals that are not necessarily true
(giving rise to some sort of a demand-driven establishment-- establish
only those things that are not yet necessarily true. If this leaves
the other conditions still necessarily true, there is a good chance
that you will terminate early).

At first glance it may _look_ as if we need the necessary truth check
to do this (and necessary truth check is intractable for anything
other than ground tweak rep).

However, this is misleading. In particular, since what we are after is
really a heuristic, we don't have to use a sound and complete truth
criterion to check necessary truth. Any polynomial approximation to
necessary truth/falsity (such as those described in Dean et. al.'s
paper) will do just fine. (The key point here is that we are not using
the truth criterion to actually terminate -- but just to pick _an_
ordering).


Hope this makes sense...

Rao
[Mar  3, 1994]



From rao  Sat Mar  5 15:05:18 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil ni
l nil])
```

```
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA00600; Sat, 5 Mar 94 15:05:18 MST
Message-Id: <9403052205.AA00600@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Pre-ordering strategies (an EVEN clearer story)
Date: Sat, 5 Mar 94 15:05:18 MST
Reply-To: rao@asuvax.asu.edu

The mail I sent earlier on pre-ordering strategies is erroneous. This
modified version is correct (or so I believe right now)...


Here is a more complete story on the pre-ordering strategies.

Pre-ordering strategies use a localized interaction test to check if a
pair of (unordered) steps should be ordered with respect to each
other. They will refine the plan until no pair of interacting steps
are left unordered.

The "interacts" routine uses the local properties of the steps to
decide whether they interact

Here are a variety of ways of defining this routine, in increasing
order of eagerness to order steps.



1. UA (unambiguous truth ordering)

   c1. s1 interacts with s2 if s1 has a precondition C and
      s2 has an effects that either negates C or adds C

    OR
   c2. s1 has an effect C and s2 has an effect that negates C

    Using UA will not only ensure that the partial plans in the search
    queue will be necessarily safe or necessarily unsafe with respect
    to protection intervals, but will also ensure that the partial
    plans are unambiguous.

    When a plan is unambiguous, every condition  is either necessarily
    true or necessarily untrue (irrespective of whether or not there
    is a causal link supporting it already).

    Thus if the plans are unambiguous, we can check necessary truth of
    a condition in polynomial time and only work on those conditions
    that are not necessarily true (i.e., are necessarily FALSE).

    In the literature, unambiguity was first introduced to show a
    systematicity like property. But, it turns out that unambiguity
    has nothing to do with systematicity. Systematicity depends only
    on whether or not the planner uses causal links based on
    contributor protection.

[ As Yong Qu pointed out in the class, if we are only interested in the
necessary safety/unsafety with respect to protection intervals, and
will only be interested in truth and falsity of the preconditions of
the steps of the plan, than c2 above can be changed as follows:

  c2': s1 has an effect C, such that C is the precondition of some
         action in the domain, or one of the top level goals of the problem
       and s2 has and effect that negates C
]


2. UCA (unambiguous contributor ordering)

   c1. s1 iteracts with s2 if s1 has a precondition C
       and s2 has an effect that either negates or adds C


   c2. s1 has an effect C and s2 has an effect that either negates
        or ADDS C

Note that the difference between UA and UCA is in the clause c2 --UCA
orders two steps not only when the condition added by one is deleted
by another, but also when the condition added by one is also added by
another.

It is easy to see that UCA is strictly stronger than UA in that it
orders every pair of steps that UA orders, and more.

With UCA, the plan will have the property of unambiguous
contributors-- that is every true condition will have a unique
contributor step in all linearizations of the plan. Because of this,
in a plan that is interaction free with respect to UCA, every
contributor protection constraint is either necessarily safe or
necessarily unsafe.

4. TO:
    s1 always interacts with s2

    This is the most conservative of interaction definitions, and it
    will wind up ordering every pair of unordered steps, thus making each
    partial plan in the search queue a totally ordered one.

    Thus, it not only gives all the properties of PI, CPI and UA, but also
    makes the planning totally ordered plan-space planning.

It should be easy to see that the branching factor introduced by PI
will be less than that introduced by CP, which will be less than that
introduced by UA, which in turn will be less than that introduced by
TO. As the branching factor increases, so does the search space size
(and the impact of the chemistry between b_t and b_e factors on
performance).

Suppose we want to compare the pre-ordering strategies with conflict
resolution strategies.

The corresponding "interacts" routine checks the interactions between
a step and an auxiliary constraint (which is typically made up of a
range and a condition protected in that range). Then we can also cover
conflict resolution using the same notion. Specifically, conflict
resolution strategies consider a pair of steps to interact only when
they are together taking part in the violation of an auxiliary
constraint (causal link).

[And
```

```
0.1 PCR
   s1 and s2 interact if there is a link L:s1--c-->s3 and
   s2 has an effect that negates c

0.2 CCR
   s1 and s2 interact if there is a link L:s1--c-->s3 and
   s2 has an effect that negates c or adds c
```

Note that PCR and CCR are less eager than UA and UCA respectively to order steps (in otherwords, if two steps interact according to PCR, then they interact according to UA too; verify this!). In particular, PCR and CCR wait until there is a causal link involving s1 and s2 before considering them to be interacting.

Given this generalization, it is easy to compare the pre-ordering strategies with conflict resolution strategies:

1. Preordering strategies may order a pair of steps that will never be ordered with respect to the conflict resolution strategies (e.g. ITO strategy will order every pair of steps even if they will never take part in an actual conflict)

2. More importantly, preordering strategies will order a pair of steps earlier than they will be ordered by the conflict resolution strategies. This means that the preordering strategies have a larger branching factor at the top of the search tree, increasing the search space size.

   Moreover, since we have no way of knowing which of the orderings will lead to solutions, it is better to avoid premature orderings (especially those introduced only to aid tractability of refinement).

Based on this, the general hypothesis is that if you _have_ to do tractability refinements, do the conflict resolution based tractability refinements, rather than pre-ordering refinement based ones, since the former are more sensitive to the role played by the steps in the plan, while the latter base their analysis just on the steps.

(about the only utility of pre-ordering based refinements is that some of them, eg UA and TO can make necessary truth computation polynomial, which can be used in goal selection and termination steps. But even this may be somewhat of a weak claim. Specifically, it is not necessary to check "necessary truth" of a plan to terminate. Since we only need one correct linearization, we can just check a random linearization of the plan.

This leaves us only with the possible advantages in goal selection The idea here is to work on goals that are not necessarily true (giving rise to some sort of a demand-driven establishment-- establish only those things that are not yet necessarily true. If this leaves the other conditions still necessarily true, there is a good chance that you will terminate early).

At first glance it may _look_ as if we need the necessary truth check to do this (and necessary truth check is intractable for anything other than ground tweak rep).

However, this is misleading. In particular, since what we are after is really a heuristic, we don't have to use a sound and complete truth criterion to check necessary truth. Any polynomial approximation to necessary truth/falsity (such as those described in Dean et. al.'s paper) will do just fine. (The key point here is that we are not using the truth criterion to actually terminate -- but just to pick _an_ ordering).

Hope this makes sense...

Rao
[Mar 5, 1994]

From rao Mon Mar 21 14:01:35 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil])
X-VM-v5-Data: ([nil nil t nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil])
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil])
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA10939; Mon, 21 Mar 94 14:01:35 MST
Message-Id: <9403212101.AA10939@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: NONLIN (HTN Planner) available in plan-cla account
Date: Mon, 21 Mar 94 14:01:35 MST
Reply-To: rao@asuvax.asu.edu

Folks--

I have now set up a task-reduction planner (Tate's Nonlin) within the plan-cla account. To run this, you just type run-nonlin at the command line (it calls cmulisp indirectly).

The manual for using nonlin is available in postscript form in the file nonlin-DOC.ps (you can print it out using the command lzpr -P nonlin-DOC.ps. I may be distributing copies of the manual by todays or wednesday's class).

[[Now onwards, to run ucpop, you need to type run-ucpop (which will initialize some files and then start ucpop)]]

Rao
[Mar 21, 1994]

From rao Mon Mar 21 14:04:01 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil nil])

```
Status: RO
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA10963; Mon, 21 Mar 94 14:04:01 MST
Message-Id: <9403212104.AA10963@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: Nonlin assignment
Date: Mon, 21 Mar 94 14:04:01 MST
Reply-To: rao@asuvax.asu.edu
```

All of you are supposed to be working on encoding and experimenting with a non-trivial planning domain using UCPOP.

The NONLIN assignment involves converting your UCPOP domain into a nonlin form (introducing non-primitive tasks as necessary), and running it on nonlin.

Please submit the domain specification and trace of runs of sample problems
From that domain.

Rao
[Mar 21, 1994]

```
From yqu@enws318.eas.asu.edu  Sat Apr  2 12:42:29 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <yqu@enws318.eas.asu.edu>
Received:
From enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA26590; Sat, 2 Apr 94 12:42:29 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA13568; Sat, 2 Apr 94 12:39:24 MST
Date: Sat, 2 Apr 94 12:39:24 MST
From: yqu@enws318.eas.asu.edu (Yong Qu)
Message-Id: <9404021939.AA13568@enws318.eas.asu.edu>
To: plan-class@enws228.eas.asu.edu
Subject: note of March 9, 1994
Content-Type: X-sun-attachment

----------
X-Sun-Data-Type: text
X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 0

----------
X-Sun-Data-Type: default
X-Sun-Data-Description: default
X-Sun-Data-Name: note.3.9
X-Sun-Content-Lines: 137
```

****

CLASS NOTE FOR MARCH 9TH, 1994

Planning Seminar
Taken by: Yong Qu

****

Agenda :

```
HTN planning  -- ending
              -- merging (consistency)
              -- condition typing
```

As we know, the expression in POP are regular expressions, while there are not non-terminals. Yet in the HTN planner, the expression are CFG, with the non terminals as high level goals. To constranit CFG to regular expression, we should add one more constraint: the expression should be whether left linear (all the non-terminal appear on the leftmost position) or right linear (all the non-terminals appear on the rightmost position).

Under this constraint, we can convert the HTN problem to the equivalent of UCPOP problem. Under the same reason, HTN is much more expressiveness:

1. it’s user controlable.
2. there are set of regular problems that can be done in HTN yet can’t be done in POP.

Following are some examples of the problems that can’t be done in UCPOP:

1. intermediate goals

first problem is the round trip problem. We want to go to one place and do something and return.

in HTN, we can define a high level action Do-round-trip(x,y), and the problem can be represented as:

START---> Do-round-trip(Phx,SFO) ---> end

in UCPOP, things are not so easy.
We can define a action call Goto(x,y) go From x to y. And the problem become:

START --> GOTO(x,y) --> GOTO(y,x) --> END

but here we make an assumption that the round-trip problem use the same traffic on both way. It would be quite difficult to plan to travel on different tools.

UCPOP start with the P0 and go on planning.
We need to add something more than just {START, END}, we add in:

```
   S1                S2               S3
  _______          _______           _____
| START |         | Dummy1|         | End |
  -------          -------           -----
(at Phx)           (at SFO)         (at Phx)
```

so the starting agenda become:
(at Phx)@S3, (at SFO)@S2,
we can try a two-step plan.
But the probelm raise immediately: how about the other goals to be solved? where should we put it? the first half or the second half?

try to solve that, we might define a new action GOTO(x,y,t), where t is the transportation.

Start --> GOTO(x,y,t)   --> D1 --> GOTO(x’,y’,t’) --> End

then we can add a binding B(t=t') to ensure that we use the same transportation.
but the binding isn't enought, we must add the link to ensure is (that is, not that the transportation are same)

as a summary, we can see that:
UCPOP   is precondition achievement planning
HTN   is plan merging algorithm

since we can have high level steps at each level, we can do harder problems HTN than in UCPOP. Also, another thing can be done in HTN is the context free language. This is beyond UCPOP.

We can compare HTN & UCPOP with the most-difficult problems, those undeciable (in any finite domain), we find that some problems in HTN can't be expressed in UCPOP.

Another things is that maintenance is more difficult in UCPOP. In HTN, new constraints can be added comparely easier by just add some more schema, yet in UCPOP it's difficult to do so.

There are some technique details on how to reduce the high level steps to low level ones. Suppose there are already some casual link exist, linking the high level action to some other goal state. Now we'll decompose this high level step, where should we put the links ? we can either put them on the first substep or the last substep.

```
         ___  C1  ____    C2    _____
        | S |--->|  T |  -----> |End |
         ---      ----           ----
```

then we decompose T into T1 & T2
we can put the casual links of precondition to the first substep, and put all the effect links to the final step. That is, all the preconditions are used by the first action, while all the effects are generated by the last action.
This implement is easy, but it will be incomplete: There might be some cases that the unresolvable casual links on the high level might become resolvable in the lower level.
for example, see the following inconsistent partial plan:

```
             !C2,Q
         C1   ___    Q
       ----> | T1|------
      |       --        |
     _|_                |     __
    |  |                ---->|  |
    |  |                ---->|  |
     --      !C1,R      |     --
      |   C2  ___    R  |
       ----->| T2|------
              ---
```

but if we reduce the steps this way:
T1   : (C1)T1' --> (!C2)T1"(+Q)
T2   : (C2)T2' --> (!C1)T2"(+R)
If we reduce this way, the plan is safe in the lower level.
We can deal with this scernrio in two ways:
first, we don't remove the inconsistent plan
From the search queue, but the searching space increase.
Another way in that we can add some constraint to keep it.

Another issue:
there are different kinds of precondition represent in HTN.
one is called "filter condition", which ned to be true at the beginning, yet it can't be achieved.
Notice that we can't use filter condition to filter establishment yet we can just postpone it, hoping that the condition might become true by some other magic. :-) so we don't allow to prune, we just rank the order.
This is an advantage towards UCPOP: we can't put these kinds of constraints on UCPOP. :-(

From yqu@enws318.eas.asu.edu Sat Apr 2 13:36:18 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <yqu@enws318.eas.asu.edu>
Received:
From enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA26626; Sat, 2 Apr 94 13:36:18 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA13603; Sat, 2 Apr 94 13:33:13 MST
Date: Sat, 2 Apr 94 13:33:13 MST
From: yqu@enws318.eas.asu.edu (Yong Qu)
Message-Id: <9404022033.AA13603@enws318.eas.asu.edu>
To: plan-class@enws228.eas.asu.edu
Subject: class note for march 31, 1994
Content-Type: X-sun-attachment

X-Sun-Data-Type: text
X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 0

X-Sun-Data-Type: default
X-Sun-Data-Description: default
X-Sun-Data-Name: note.3.31
X-Sun-Content-Lines: 39

****
CLASS NOTE FOR MARCH 30, 1994
PLANNING SEMINAR
NOTE TAKER: YONG QU
****

Agenda:
Review of execution model
SCR --and incremental universal plans
replanning /interleaving execution
Next class:
Scheduling & time dependent planning

In a reaction execution we just take plan as advisor of the executor, find out which aciton we shouldn't use. While in replanning, we try to

```
detect the current world state and add the new goals to replan.
    In interleaving execution, we interleave execution into planning, add
information into the plan during the planning.

    Suppose we are using the UCPOP-like execution planning algorithm.
The partial plan is a tuple as <S,O,L,ST>, two states <I,G>. During the
execution, we find that we are in the current world state W, and try to
find that where we can go on the current partial plan without having to
do the replanning.
    The condition should be that we try to find the minimal condition
which make the plan goes on
From an "executable" position.
    We define s1//s2 as (S1!< S2) ^ (S1 !>S2)
    The condition is the casuation of S (Exec(P,W)
    to any s belongs to Exec(P,W) (the execuatble actions in current
plan under W state), s can be executed
From W state if:
                           C
    condition 1.  Any l:S1--->S2 ,S2 belongs to S, s.t. C belongs to W

                          C
    condition 2. Any l:S1-->S2 (S1<S<S2) S belongs to S, s.t. C belogns W.


    condition 3. to be continue
```

From ATAPK@ACVAX.INRE.ASU.EDU Mon Apr 11 16:27:39 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <ATAPK@ACVAX.INRE.ASU.EDU>
Received:
From ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA08332; Mon, 11 Apr 94 16:27:39 MST
Received:
From ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF V4.2-13 #2382) id
 <01HB1ZUP4RA8008EFX@ACVAX.INRE.ASU.EDU>; Mon, 11 Apr 1994 16:23:20 MST
Date: Mon, 11 Apr 1994 16:23:20 -0700 (MST)
From: ATAPK@ACVAX.INRE.ASU.EDU
Subject: PLANNING CLASS NOTES - March 7th
To: PLAN-CLASS@PARIKALPIK.eas.asu.edu
Message-Id: <01HB1ZUP5K82008EFX@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"PLAN-CLASS@PARIKALPIK.EAS.ASU.EDU"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT

****
CSE 591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE
NOTES
From THE SESSION ON MARCH 7th
Notes taken by: Dimitri Karadimce
****

Agenda:

1. We review the basic ideas underlying Hierarchical Task Network (HTN) planners.

2. We look at a simple example of HTN planning.

3. We further discuss some details of task reduction in HTN.

4. We identify and discuss some of the technical difficulties that may be encountered during task reductions.

The following papers on hierarchical planners are assigned as additional reading:

i. "Generating Project Networks" by Austin Tate
    (featuring the NONLIN planner).

ii. "The Nonlinear Nature of Plans" by Earl D. Sacerdoti
    (featuring the NOAH planner).

Note that the papers were written in the 70-ties, and use some terminology that differs
From the UCPOP-like terminology.
For example, "criticizing the plan" in these papers is close to the UCPOP notion of "resolving the threats", etc.

****
1. INTRODUCTION TO HTN PLANNING REVIEWED...

Hierarchical Task Network (HTN) planners deal with two types of tasks (actions):

i. primitive tasks
ii. non-primitive tasks

The HTN planner may start with a single-task plan, which is very high-level (non-primitive), and will recursively try to REDUCE all non-primitive tasks of the plan to primitive and/or (lower-level) non-primitive tasks, until only non-primitive tasks are left in the plan.

The TASK REDUCTION is the basic REFINEMENT step used in HTN planners. Recall that in UCPOP the basic refinement step is to support (establish) a precondition of a step (task) in the plan.

Note that the task reduction, used in HTN, is more general plan refinement than the establishment used in UCPOP. Actually, the task reduction scheme includes the precondition establishment.

Both HTN and UCPOP plans can be represented by the tuple P = (A, O, B, ST, L), where A=actions in the plan, O=orderings, B=bindings, ST=symbol table, L=causal (establishment) links. However, in UCPOP only primitive actions are allowed in A, while in HTN both primitive and non-primitive actions are allowed. In principle, it is the only conceptual difference between UCPOP and HTN.

2. A SIMPLE EXAMPLE OF HTN PLANNING

Consider the action "take flight 1234". It seems that this action is quite low-level, and will probably be considered as a primitive task. Its precondition are "be at the airport of departure of the flight 1234" and "have ticket for flight 1234". The effect is

"you will be at the destination of the flight 1234".
Note that the action is totally instantiated.

The same HTN planner in the same domain may have the action "fly
From X to Y". Its preconditions are "be at X" and "have ticket for flight
From X to Y". The effect is "you will be at Y".
This action is an action template, since it contains variables.
Action templates are perfectly legal in HTN planners, as they are in UCPOP planners, too.

Even more general action may exist in the same HTN planner in the same domain, such as "go
From X to Y". It has similar preconditions and effects as the above action. However, it does not specify the means of travel. It can be viewed as the general action "go
From X to Y by M", where M is the means of travel.

The action "go
From X to Y", may be reduced to the following plan:

```
                 +--------------------+
             +---| go-to-airport in X |---+
+-------+    |   +--------------------+   |  +-------------+  +-----+
| start |----|                            |--| take-flight |--| end |
+-------+    |   +--------------------+   |  |
From X to Y |  +-----+
             +---| get ticket for     |---+  +-------------+
                 | flight
From X to Y |
                 +--------------------+
```

Note that the means of travel is already set to "flight".
The existence of the action "go-to-airport in X" ensures that there is an airport in city X (such condition will probably be directly supported by the initial state).

Note how the action "go-to-airport in X" provides the precondition "be-at-airport in X" for the action "take-flight
From X to Y".

This plan may be further reduced, depending on whether the actions (such as "get ticket...") are primitive or not.

During the refinement of the plan, it may turn out that no furher refinements are possible, and the plan is not valid (for example, it may still contain non-primitive actions, or some preconditions of the actions may not be supported). In the above example, it may be that the action "get ticket for flight
From X to Y" has a precondition "have 300 $ in the pocket", which is not satisfied in the start state, and there are no actions that can increase the amount of money in the pocket.

The above reduction of the action "go
From X to Y" illustrates a typical "recipe" for task reduction used in HTN planning. There may be several recipes known to the planner for reduction of each task. For example, another possible reduction of the action "go
From X to Y" may involve traveling by bus. In such reduction, actions such as "go-to-bus-station in X" and "take-bus from X to Y" may appear.

The planner may backtrack when a given reduction does not lead to a valid plan. The planner is guaranteed to find a valid plan (if one exists) if it considers all alternatives, using some complete search strategy. However, backtracking should be avoided as much as possible, since it reduces the efficiency. Moreover, one of the big motivations for HTN planning is improved efficiency, since there is a reason to hope that efficiency is inherent to the hierarchical approach. Therefore, relying heavily on backtracking will affect one of the key advantages of HTN planning: efficiency.

By choosing to do HTN planning, one hopes that the refinement of two different tasks in the higher-level plan usually will not ADD new constraints between the two groups of subtasks generated by refining these two "parent" tasks.

Note that when doing HTN planning, we are still doing binding, have constraints, auxiliary constraints, etc., similarly to the familiar UCPOP planning.

Another simple example of action refinement in HTN planning may be the refinement of the single action "make-hole-by-drilling" to the lower-level actions: (1) "position the hole", (2) "make primary hole", and (3) "fine-drill the hole".

3. SELECTED DETAILS OF TASK REDUCTION

Any HTN algorithm performs the planning in refinements cycles.
Each cycle includes two basic steps: (1) reducing the steps (tasks), and (2) taking care of the constraints.

In order to support an open precondition C of a step in the plan, HTN has two options: (1) to use the effect C of an existing step, or (2) to instantiate a new step providing the effect C. The latter option creates the so-called "precondition achievement tasks".
There may be several candidate tasks that can provide the effect C.

During the task reduction process, usually there are more different ways to reduce a given task. Some of the reductions may include NO-OPERATION (NOOP) subtasks. NOOP tasks are useful when one of the choices for task reduction is to reduce to no subtasks. The NOOP tasks are also sometimes needed to keep track (to enforce) the auxiliary constraints (causal links) that are can not be assigned to the other subtasks generated by the reduction.

When introducing tasks to support preconditions of existing tasks, causal links are added to the plan structure. For example, if the existing task t1 needed the precondition C, a new task s' may be instantiated having an effect C:

```
  s'                        t1
+-------------+  C -> C  +----------------------------+
| achieve (C) |----------| do-whatever-supposed-to-do |
+-------------+          +----------------------------+
```

Once causal links are allowed in the plan, then they can be threatened, so there should be some form of conflict resolution, similarly to UCPOP.

Consider the following example:

```
            s1                         s2
          +-------------+  C -> C  +-----------------------------+
          | achieve (C) |----------| do-whatever-supposed-to-do |
          +-------------+          +-----------------------------+

                 s3
               +-----------------+
               |  do-something  |  ~C
               +-----------------+

                                           C
The task s3 threatens the causal link  s1 ---> s2.  It doesn't
```

matter whether s3 is a primitive task or not.   So some threats may be spotted and dealt with on a higher level, with non-primitive tasks. Actually, it is desirable to detect the threats on a higher level, where they can be dealt with more efficiently.

Note that after each task reduction it is necessary to check the interdependencies of all generated subtasks to all other tasks currently in the plan.

The consistency check should be applied after each reduction (refinement) cycle, and if the plan does not satisfy the constraints, it should be abandoned.

Digression: Learning is one of the attractive aspects of HTN planners. HTN planners seem to be better suited to include learning capabilities.

4. DIFFICULTIES IN TASK REDUCTIONS

Task reduction is the basic refinement step in HTN planning. There are some technical problems associated to the reduction and checking of constraints and threats. Some natural questions immediately arise: (1) how to do reductions, and (2) how to merge the reductions into the existing plan. The necessity to address these questions is what makes HTN planners different From UCPOP- -like planners.

Note that task reductions may ADD new dependencies between two non-primitive tasks, that on a higher level were not visible. This possibility adds to the complexity of doing HTN planning, and generally reduces the efficiency of the planning process. While doing conflict resolution, one may apply exactly the same algorithms as used in UCPOP. However, it will usually be less efficient, since no advantage will be taken of the hierarchical structure of tasks in HTN, where it is usually the case that no new interactions will be introduced by task reductions.

Another design question for HTN planners is whether to keep reducing the selected task until only non-primitive subtasks are left, or to take one task and reduce it once, then go to some other task, reduce it once, etc., and then recursively reducing the subtasks in the same manner.   The former approach is similar to the LIFO data structure, while the latter corresponds to FIFO data structure. Both strategies are complete, although it seems that FIFO approach would be more efficient, since it is more likely to take advantage of the hierarchical nature of HTN planners.

Let us return to the MERGING-OF-REDUCTIONS problem that HTN planners must address. Two questions arise: (1) what happens to causal links between higher-level actions when they are reduced to subtasks, and (2) what happens to the partial orderings. Both questions address the same type of problem: how to split the obligations of the parents among their children.

In order to further illustrate the problems that HTN planners face, consider a simple single-task plan, where the task requires the preconditions X and Y, and generates U & V (the goal). One possible reduction of the task may lead to the following plan:

```
              X  +----+ U  ~Y
             +---| T1 |----------+
+-------+    |   +----+          |  +-----+
| start |----|                   |--| end |
+-------+    |   +----+          |  +-----+
             +---| T2 |----------+
              Y  +----+ V  ~X
```

It is clear that the refined plan can NOT be executed. So, starting with an executable high-level plan, the HTN planner may end up with an illegal plan, after performing some reductions. In such a case, the HTN planner will have to backtrack.

The opposite case is also possible, although less likely to appear in realistic planning problems. Namely, a given plan may be illegal (not executable) on a higher level, but after performing some reductions it may turn out to be a viable plan! We are to come (for homework) with an example illustrating this case. The example should be very simple, assuming a plan with a single start and single end points, and only two steps.

Let us consider an HTN planning problem in the blocks world. Let the initial state be:

```
  +---+
  | B |
  +---+   +---+
  | A |   | C |
  +---+   +---+
-----------------
```

Let the initial plan contain a single non-primitive action:

```
             +------------------+
+-------+    | achieve on(A,B) |  +-----+
| start |----|       & on(B,C) |--| end |
+-------+    |                  |  +-----+
             +------------------+
```

Suppose that the library of reductions suggests that the above non-primitive task may be reduced to two subtasks, leading to the following plan:

```
                +------------------
            +---| achieve on(A,B) |---+
            |   +-----------------+   |
+-------+   |                         |  +-----+
| start |---+                         +--| end |
+-------+   |                         |  +-----+
            |   +-----------------+   |
            +---| achieve on(B,C) |---+
                +-----------------+
```

Note that for any given non-primitive task there may be several possible reductions stored in the library. Usually, HTN planners rely heavily on appropriate domain information to guide the selection of reductions, and to guide the planning in general.

Let us use the FIFO goal selection schema.
We assume that the library indicates that the non-primitive task 'achieve on(X,Y)' can be decomposed to the primitive task 'puton(X,Y)', having the preconditions clear(X) & clear(Y), and the effect on(X,Y), ~clear(Y).

After the reduction of both tasks, the following plan is obtained:

```
                  +~~~~~~~~~+
               +--| clear A |--+
               |  ~~~~~~~~~~~  |    +-------------+
            +--|  +.........+  |----|  puton(A,B) |---+
            |  +--| clear B |--+    +-------------+   |
+-------+   |     +.........+                         |  +-----+
| start |---+                                         +--| end |
+-------+   |     +.........+                         |  +-----+
            |  +--| clear B |--+    +-------------+   |
            +--|  +.........+  |----|  puton(B,C) |---+
               |  +.........+  |    +-------------+
               +--| clear C |--+
                  +.........+
```

Note: dashed boxes denote primitive actions, boxes with tildes (~) denote non-primitive actions, and dotted boxes denote 'phantom' actions. 'Phantom' actions are designed to keep track of preconditions that are already supported by the initial state. They may later become real actions, if their effect is deleted by some action that comes necessarily between the initial state and the phantom action.

In general, action-library schemas used in HTN planners give the user better control on the operation of the planner, when compared to UCPOP planners. The redundancy is built in the knowledge accumulated in the library. The redundancy here means that there are more possible ways to reduce a given non-primitive task, each requiring (possibly) separate set of preconditions and involving different set of subtasks.
Note that the above observation can not be well-illustrated in the blocks world domain, since in it usually there is only one primitive task that can achieve a given effect (e.g. on(A.B)).
In domains where there is redundancy, the user may guide the planning process by assigning priorities to the separate choices for task reduction, or provide some other domain-dependent heuristics.

Note that the notion of completeness is changed in HTN, compared to UCPOP. In UCPOP the "standard" definition of completeness is used (such as "if the problem CAN be solved, it WILL be solved"). The completeness of HTN planning depends on the information stored in the library of available reductions, and it may well be that the library lacks certain recipes for task reductions that would lead to solution. Therefore, although the planning problem is solvable with the primitive actions that can be used, HTN may not find the plan, since the library may not provide sufficient guidance. As an example, the library may lack the "connect" the high-level goal (task) 'have money' with the low-level task 'steal money', although the latter may be available as a primitive action, but is probably only used in refinements of other tasks.

It is not clear whether the above observation on the incompleteness of HTN in the strong (UCPOP-like) sense should be considered as a weakness or strength of HTN planners. While we are sure losing the completeness, do we really care about such weird solutions of the planning problem, e.g. involving stealing money?
It may well be that the designer of the library was aware of such solutions, but knowingly ruled them out by not including the appropriate recipes. So the property of HTN planners to give more control over the execution of the planning process and on the acceptable plans is visible in the context of completeness, too.

Note also that given a set of actions, UCPOP planners will consider ALL possible plans involving these actions (the '*' closure of the set of actions), while HTN planners may only look at some limited subset of all available plans. For example, given a set of three actions { A1, A2, A3 }, and due to the recipes stored in the library, an HTN planner may consider only the subset of all possible plans defined by the following context-free grammar:

```
T' ->  T1, T2
T' ->  T2, T1
T1 ->  A1, T1
T1 ->  A1
T2 ->  A2, T2
T2 ->  A2.
```

Note that the reductions defined above are recursive. Recursion in the definition of the reductions is perfectly legal in HTN planning, and is one of its advantages. It may be useful when planning some iterative tasks, such as unloading a truck.
For example, the non-primitive task "unload the truck" may have two recipes: (1) "remove a bin" and then "unload the truck" (actually, unload the rest of the truckload), or (2) do nothing. Note that this scheme resembles typical recursion found (for example) in Prolog. It also allows HTN planners to deal with repetitive tasks, and still obey the static universe assumption. Note that context-free grammars, on the other hand, usually generate infinite-length sequences of actions.

Let us return to the blocks world problem discussed above.
Recall that we included 'phantom' tasks for conditions that were already true in the initial state, and were used as preconditions of the instantiated steps. The phantom tasks are needed to protect

```
such preconditions
From threats.  The usage of phantom tasks for
such purpose is called "phantomization".

Digression: Note that in general when we further refine the plan, there will
            usually be more choices for plan refinement (task reduction).
            All such choices must be considered, and they should be put
            on the search queue.  Otherwise, the completeness will be
            compromised.  Recall that in HTN planners the completeness is
            meaningful only with respect to the grammar (the library of
            possible reductions), and not on the actual possibility to
            generate the plan based on the available primitive actions.

Back to the blocks world example, we note that the action 'puton(A,B)'
has the effect ~clear(B), and it threatens the causal link going

From  clear(B) to  puton(B,C).  Usually, there will be several
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
possibilities to resolve the threat.  All should be considered,
by saving them in the search queue.  Threats to phantom steps should
also be resolved, in the same way.

Let us assume that the planner decided to resolve the threat by
ordering the step puton(B,C) BEFORE the step puton(A,B), using the
ordering constraints.

The last subgoal that remained to be resolved is 'clear(A)'.
The library may indicate several ways to achieve it, such as by taking
whatever is on A and putting it on the table or on another block.
Note that even here the user (the designer of the library) may
exercise good control over the selection of the refinement.  For
example, the rules in the library may demand that the block
in the above case must be put to the table (so NONE of the other
possibilities, such as putting the block on the other blocks,
will be considered).  Note that in UCPOP it is very hard to
exercise comparable control over the planning process. It may
require a change of the definition of the actions, which is
highly undesirable and cumbersome.

Let us assume that the HTN planner decided to achieve 'clear(A)'
by using the effect 'clear(A)' of the exixting step 'puton(B,C)'
(simple establishment).  Since all tasks are now primitive, all
preconditions have been supported, and there are no threats, the
planning process successfully terminates.   The final plan will be:

  puton(B,C)  --->  puton(A,B) .

Note that the final plan may contain phantom actions as well, if
they are not threatened.

*******************************************************************
END OF NOTES
From THE SESSION ON MARCH 7th
*******************************************************************


From rus@enws293.eas.asu.edu  Fri Apr 29 22:37:20 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <rus@enws293.eas.asu.edu>
Received:
From enws293.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA04496; Fri, 29 Apr 94 22:37:20 MST
Received:
From severn.eas.asu.edu (enws295.eas.asu.edu [129.219.25.19]) by enws293.eas.asu
.edu (8.6.4/8.6.4) with SMTP id WAA22154 for <plan-class@parikalpik.eas.asu.edu>
; Fri, 29 Apr 1994 22:46:04 -0700
Date: Fri, 29 Apr 1994 22:46:04 -0700
From: Rus Ioana <rus@enws293.eas.asu.edu>
Message-Id: <199404300546.WAA22154@enws293.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu
Subject: NOTES for the April 22nd PRESENTATION
Content-Type: X-sun-attachment

----------
X-Sun-Data-Type: text
X-Sun-Data-Description: text
X-Sun-Data-Name: text
X-Sun-Content-Lines: 275

-------------------------------------------------------------------
                CSE 591                 SPRING 1994

        notes for the class on April 22, taken by Ioana Rus
-------------------------------------------------------------------


        Planning as Search Involving Quasi-Independent Subgoals
        -------------------------------------------------------



1. Overview

        - A theoretical analysis of conjunctive goals problem
        - Planning as search using subgolas, macro-operators and abstraction
          as knowledge sources
        - Characterizing subgoals interaction for planning
        - Conclusions


2. A theoretical analysis of conjunctive goals problem

        When final goal is a conjunction of subgoals, the plan depends
on the goals interaction.
Goal protection means to protect a goal already achieved. But this
is not always possible.

For example, the Sussman anomaly:

                                ----
                                | A|
        ----                    ----
        | C|                    | B|
        ----  ----              ----
        | A|  | B|              | C|
        ----  ----              ----
```

```
        initial state           goal state
```

or a robot planning problem: the robot (X) is in a room, in position 1; it has to go in the other room, near a sink ([] position 4), thru a door (positions 2-3). In the final state, the door must be closed.

```
        -----------------------          -----------------------
        |          |          |          |          |          |
        |          |          |          |          |          |
        |         2-3         |          |         2-3         |
        |X         |          |          |          |        X |
        |1         |       4[]|          |1         |       4[]|
        -----------------------          -----------------------

           initial state                            goal state
```

For these problems, goal protection cannot be satisfied.
In [1], they provide a method for analyzing the linearity of a problem, using graphs.
A nonlinear problem is one for which the subgoals cannot be solved in some linear order without having to take into account the way the subgoals interfere with each other.
If planning is done as a search in the world state space, and this space is finite, then it can be represented as a graph:

```
        1O-----2O-------3O------4O      where 1,2,3,4 are positions
               |        |               and O/C means door open/closed
        1C-----2C       3C------4C
       initial                  goal
        state                   state

                State space
```

The goal is composed by two subgoals: position 4 and door closed (C)
The graph can be decomposed into subgraphs, corresponding to the subgoals:

```
        1C----2C     3C----4C                  4O
                                               4C

        subgraph 'door closed'       subgraph 'in position 4'
```

If assume that the operators can be reversed, then these subgraphs are undirected. They have connected components for the subgoals. To achieve a goal means entering in any of these connected components. To protect a goal, is to confine to that connected component; if it doesn't contain the final goal state, then protecting the subgoals prevents solving the problem.

Using this method, by analysing the graphs, one can say whether the problem is or is not nonlinear: having a connected component that doesn't contain a final goal state is a necessary condition for nonlinearity .

(That doesn't mean that there cannot be any algorithm able to avoid entering that component)

3. Planning as search using subgoals, macro-operators and abstraction as knowledge sources

Planning as search can be done either by brute-force search, or using some knowledge sources, such as heuristics, subgoals interaction, macro-operators, and abstraction. All of these are used in order to reduce the complexity.
For brute-force search, the space is $O(b^d)$ for breadth first search, or O(d) for depth first search, and the time is $O(b^d)$, where b=branching factor, and d=depth of the tree.

Heuristics reduce the branching factor.
If the problem can be decomposed, this tends to devide the exponent.
In [2], they develop a subgoals interaction hierarchy, as follows:

1) INDEPENDENT subgoals

Definition: each operator can change the distance to a single subgoal
Property: by concatenating the optimal solutions for the subgoals, can get the optimal solution for the global goal.
Complexity reduction: both the base and the exponent, by the number of subgoals.

2) SERIALIZABLE subgoals

Definition: there is an ordering among the subgoals, such that the subgoals can be always solved sequentially without ever violating a previously solved subgoal in the order.

Advantages: reduces the branching factor (knowing that the goals are serializable, and that a subgoal was achieved, the paths with the subgoal not yet reached will be pruned)

Disadvantages:- proving serializabilty is as difficult as proving that a problem is solvable
From the initial state.
- protecting goals may increase the length of the solution

3) NON-SERIALIZABLE subgoals

Definition: previously achieved goals must be violated for making further progress towards the main goal, regardless of the solution order.

Complexity: - is not reduced

Advantages: - in general, solving subgoals, even non-serializable, reduces the distance to the goal.

4) PATHOLOGICAL subgoals

Definition: solving the subgoals doesn't decrease the distance to the main goal.
Example: sets of subgoals for Rubik's cube

5) BLOCK-SERIALIZABLE subgoals

Definition: serializable sequences of multiple subgoals

Serializability can be considered function of the sets of subgoals. Can abstract, grouping subgoals into serializable sets. The abstraction can be at one level, or at multiple levels.

MACRO-OPERATORS are sequences of primitive operators.

Defining macro-operators involves lerning. It’s usefull when
- the same problem must be solved many times
- many similar problems to solve-> the cost of learning will be amortized over all the problems instances to be solved.

For non-serializable subgoals, can define macros that leave previously achieved goals intact, even though they may violate them temporary.

The goals become serializable w.r.t. the set of macros. An exponential number of problem instances can be solved without any search, using only a linear amount of knowldege expressed as macros.

In [3], the hierarchy of subgoal interaction is extended:

6) TRIVIALLY SERIALIZABLE subgoals

Definition: each subgoal can be solved sequentially in any order, without ever violating past progress

7) LABORIOUSLY SERIALIZABLE subgoals

Definition: there exists 1/n orders in which the subgoals cannot be solved without possibly violating past progress

In [3], they claim that the partial order planners have the advantage that if all planners perform well only when confronted with trivially serializable subgoals, more domain are serializable for partial order planners than for the total order planners.

To support this, they analyze the performance of three different planners for an artificial domain, created for the analysis.

The three planners are:
- TOPI - plan-space algorithm, isomorphic to a regression search in the state space
- TOCL - total order planner
- POCL - partial order planner

The domain is called D1S1*, and it uses a STRIPS representation:

```
(def-step: action Ai
           precond {Ii}
           add {Gi}
           delete {Ii+1,I*})
(def-step: action A2*
           precond {P*}
           add {G*}
           delete {I1})
(def-step: action A1*
           precond {I*}
           add {P*}
           delete {})
```

```
initial state                             goal state
------                                    ------
| In |----An----------------------------|      |
|    |  ...                             | Gn |
| ...|                                  | ...|
| I2 |--------------------A2------------| G2 |
| I1 |------------------------A1--------| G1 |
| I* |---A1*---------------------A2*----| G* |
------                                    ------
                  Time
                -------------->
```

The subgoals specified by this domain are:
- non-serializable for TOPI,
- trivially serializable for POCL
- laboriously serializable for TOCL

Their empirical results shown that both total order planners should find problems in this domain intractable, while the partial order planner should have no difficulty.

CONCLUSIONS

The serializability of a problem depends on:
- the subgoals
- the initial state
- the operators
- the problem solver

Protection and serializability are defined differently for the two planning approaches : state space and plan space, but still attempted to be compared.

References:

1. David Joslin and John Roach, "A Theoretical Analysis of Conjunctive_Goal Problems", AI 41 (1989/90) 97-106
2. Richard E. Korf, "Planning as Search: A Quantitative Approach", AI 33 (1987) 65-88
3. Anthony Barrett and Daniel Weld, "Characterizing Subgoal Interaction for Planning", IJCAI 1993

X-Sun-Data-Type: default
X-Sun-Data-Description: default

X-Sun-Data-Name: notes_pres
X-Sun-Content-Lines: 275

```
-------------------------------------------------------------------
                CSE 591                 SPRING 1994

        notes for the class on April 22, taken by Ioana Rus
-------------------------------------------------------------------
```

Planning as Search Involving Quasi-Independent Subgoals

1. Overview

- A theoretical analysis of conjunctive goals problem
- Planning as search using subgolas, macro-operators and abstraction as knowledge sources
- Characterizing subgoals interaction for planning
- Conclusions

2. A theoretical analysis of conjunctive goals problem

When final goal is a conjunction of subgoals, the plan depends on the goals interaction.
Goal protection means to protect a goal already achieved. But this is not always possible.

For example, the Sussman anomaly:

```
                        ----
                        | A|
----                    ----
| C|                    | B|
----  ----              ----
| A|  | B|              | C|
----  ----              ----

initial state           goal state
```

or a robot planning problem: the robot (X) is in a room, in position 1; it has to go in the other room, near a sink ([] position 4), thru a door (positions 2-3). In the final state, the door must be closed.

```
-----------------------          -----------------------
|          |          |          |          |          |
|          |          |          |          |          |
|         2-3         |          |         2-3         |
|X         |          |          |          |        X |
|1         |       4[]|          |1         |       4[]|
-----------------------          -----------------------

   initial state                            goal state
```

For these problems, goal protection cannot be satisfied.
In [1], they provide a method for analyzing the linearity of a problem, using graphs.
A nonlinear problem is one for which the subgoals cannot be solved in some linear order without having to take into account the way the subgoals interfere with each other.
If planning is done as a search in the world state space, and this space is finite, then it can be represented as a graph:

```
  1O-----2O-------3O------4O      where 1,2,3,4 are positions
         |        |               and O/C means door open/closed
  1C-----2C       3C------4C
 initial                goal
  state                 state

          State space
```

The goal is composed by two subgoals: position 4 and door closed (C)
The graph can be decomposed into subgraphs, corresponding to the subgoals:

```
  1C----2C     3C----4C                   4O
                                          4C

  subgraph 'door closed'       subgraph 'in position 4'
```

If assume that the operators can be reversed, then these subgraphs are undirected. They have connected components for the subgoals. To achieve a goal means entering in any of these connected components. To protect a goal, is to confine to that connected component; if it doesn't contain the final goal state, then protecting the subgoals prevents solving the problem.

Using this method, by analysing the graphs, one can say whether the problem is or is not nonlinear: having a connected component that doesn't contain a final goal state is a necessary condition for nonlinearity .

(That doesn't mean that there cannot be any algorithm able to avoid entering that component)

3. Planning as search using subgoals, macro-operators and abstraction as knowledge sources

Planning as search can be done either by brute-force search, or using some knowledge sources, such as heuristics, subgoals interaction, macro-operators, and abstraction. All of these are used in order to reduce the complexity.
For brute-force search, the space is $O(b^d)$ for breadth first search, or O(d) for depth first search, and the time is $O(b^d)$, where b=branching factor, and d=depth of the tree.

Heuristics reduce the branching factor.
If the problem can be decomposed, this tends to devide the exponent.
In [2], they develop a subgoals interaction hierarchy, as follows:

1) INDEPENDENT subgoals

Definition: each operator can change the distance to a single subgoal
Property: by concatenating the optimal solutions for the subgoals, can get the optimal solution for the global goal.
Complexity reduction: both the base and the exponent, by the number of subgoals.

2) SERIALIZABLE subgoals

Definition: there is an ordering among the subgoals, such that the subgoals can be always solved sequentially without ever violating a previously solved subgoal in the order.

Advantages: reduces the branching factor (knowing that the goals are serializable, and that a subgoal was achieved, the paths with the subgoal not yet reached will be pruned)

Disadvantages:- proving serializabilty is as difficult as proving that a problem is solvable
From the initial state.
- protecting goals may increase the length of the solution

3) NON-SERIALIZABLE subgoals

Definition: previously achieved goals must be violated for making further progress towards the main goal, regardless of the solution order.

Complexity: - is not reduced

Advantages: - in general, solving subgoals, even non-serializable, reduces the distance to the goal.

4) PATHOLOGICAL subgoals

Definition: solving the subgoals doesn't decrease the distance to the main goal.
Example: sets of subgoals for Rubik's cube

5) BLOCK-SERIALIZABLE subgoals

Definition: serializable sequences of multiple subgoals

Serializability can be considered function of the sets of subgoals.
Can abstract, grouping subgoals into serializable sets.
The abstraction can be at one level, or at multiple levels.

MACRO-OPERATORS are sequences of primitive operators.

Defining macro-operators involves lerning. It's usefull when
- the same problem must be solved many times
- many similar problems to solve-> the cost of learning will be amortized over all the problems instances to be solved.

For non-serializable subgoals, can define macros that leave previously achieved goals intact, even though they may violate them temporary.

The goals become serializable w.r.t. the set of macros.
An exponential number of problem instances can be solved without any search, using only a linear amount of knowldege expressed as macros.

In [3], the hierarchy of subgoal interaction is extended:

6) TRIVIALLY SERIALIZABLE subgoals

Definition: each subgoal can be solved sequentially in any order, without ever violating past progress

7) LABORIOUSLY SERIALIZABLE subgoals

Definition: there exists 1/n orders in which the subgoals cannot be solved without possibly violating past progress

In [3], they claim that the partial order planners have the advantage that if all planners perform well only when confronted with trivially serializable subgoals, more domain are serializable for partial order planners than for the total order planners.

To support this, they analyze the performance of three different planners for an artificial domain, created for the analysis.

The three planners are:
- TOPI - plan-space algorithm, isomorphic to a regression search in the state space
- TOCL - total order planner
- POCL - partial order planner

The domain is called D1S1*, and it uses a STRIPS representation:

```
(def-step: action Ai
           precond {Ii}
           add {Gi}
           delete {Ii+1,I*})
(def-step: action A2*
           precond {P*}
           add {G*}
           delete {I1})
(def-step: action A1*
           precond {I*}
           add {P*}
           delete {})
```

```
initial state                                goal state
------                                       ------
| In |----An----------------------------|    |
|    |  ...                             | Gn |
| ...|                                  | ...|
| I2 |---------------------A2------------| G2 |
| I1 |-------------------------A1--------| G1 |
| I* |---A1*---------------------A2*----| G* |
------                                       ------
                   Time
                -------------->
```

The subgoals specified by this domain are:
- non-serializable for TOPI,
- trivially serializable for POCL
- laboriously serializable for TOCL

Their empirical results shown that both total order planners should find problems in this domain intractable, while the partial order planner should have no difficulty.

CONCLUSIONS

The serializability of a problem depends on:
- the subgoals
- the initial state
- the operators
- the problem solver

Protection and serializability are defined differently for the two planning approaches : state space and plan space, but still attempted to be compared.

References:

1. David Joslin and John Roach, "A Theoretical Analysis of Conjunctive_Goal Problems", AI 41 (1989/90) 97-106
2. Richard E. Korf, "Planning as Search: A Quantitative Approach", AI 33 (1987) 65-88
3. Anthony Barrett and Daniel Weld, "Characterizing Subgoal Interaction for Planning", IJCAI 1993


```
From ATAPK@ACVAX.INRE.ASU.EDU  Thu May 12 22:01:52 1994
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
Status: RO
Return-Path: <ATAPK@ACVAX.INRE.ASU.EDU>
Received:
From ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA07203; Thu, 12 May 94 22:01:52 MST
Received:
From ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF V4.2-13 #2382) id
 <01HC9ML7BXQO00IXY8@ACVAX.INRE.ASU.EDU>; Thu, 12 May 1994 21:57:47 MST
Date: Thu, 12 May 1994 21:57:46 -0700 (MST)
From: ATAPK@ACVAX.INRE.ASU.EDU
Subject: PLANNING CLASS NOTES - Appril 25th
To: PLAN-CLASS@PARIKALPIK.eas.asu.edu
Message-Id: <01HC9ML7D9YQ00IXY8@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"PLAN-CLASS@PARIKALPIK.EAS.ASU.EDU"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT
```


****

CSE591: PLANNING & LEARNING METHODS IN ARTIFICIAL INTELLIGENCE

NOTES
From THE SESSION ON APRIL 25th, 1994

Notes taken by: Dimitri Karadimce

Note: Notes based on the presentation given by Dimitri Karadimce
****

AGENDA:

1. We look at the complexity of the planning problem in general and under the well-known restrictions (STRIPS, TWEAK)

2. We briefly overview the known approaches of dealing with complexity of the planning problem

3. We summarize some known results on the complexity of STRIPS propositional planning

4. We introduce the SAS+ planning formalism

5. We summarize the known complexity results for SAS+ planning

6. We conclude by summarizing the results and ideas presented
****

0. INTRODUCTION

The planning seminar presentation on April 25, 1994, looks at the complexity of planning, emphasizing some known tractable subclasses of planning.

After briefly discussing the complexity of the planning problem in general, we consider some of the available options to reduce complexity of domain-dependent and domain-independent planning. Then we concentrate on the complexity results for planning problems representable in the SAS+ formalism, introduced by Backstrom [1]. Tractable variants of problems expressed in SAS+ formalism are introduced and discussed.

1. INTRODUCTION TO COMPLEXITY OF PLANNING

1.1. The General Planning Problem

The general planning problem can be described as follows:

```
  -  Given    D - description of the world (the domain)
              O - set of available operators
              I - description of the initial state
              G - description of the goal state

     (D, O, I, G  can be represented as general sentences in
                  First Order Logic)

  - Find a sequence of actions leading
From the initial to the
    goal state.
```

The general planning problem is UNDECIDABLE. It means that given a PLANNING PROBLEM and a PLANNING ALGORITHM, in general one can NOT tell in advance whether the planning algorithm will HALT.

To clarify the implications of undecidability, consider a run of a planning algorithm on a planning problem, where the algorithm is allowed to take ANY finite time and resources (provided that the limits are set prior to the running of the algorithm). If the planning algorithm fails to generate a plan (or fails to determine that no plan exists) within the predetermined limits, one can NOT tell whether a plan that solves the planning problem exists or not.

Clearly, undecidability is a very undesirable property for the practical use of planning. The expresiveness of the general planning problem needs to be sacrificed in order to avoid undecidability.

1.2. Restricted Planning Problems

Several planning formalisms have been proposed that restrict the expresiveness of the general (unrestricted first-order) planning formalism, trying to deal only with "easier" planning problems. The expresiveness of the domain, the operators, and the initial and goal states (and any combination of these) can be restricted.

The best-known restricted planning formalisms include the following:

a. STRIPS formalism

- Restricts the expresiveness of the operators. Each operator is defined by three lists: precondition, add, and delete list. The operator is applicable in a given state if that state entails all sentences of the PRECONDITION list. If the operator is executed, the sentences of the ADD list are added to the world state, and the sentences of the DELETE list are removed
From the world state.

- In its most general form, STRIPS does not impose any restrictions on the domain.

- Very often, only the PROPOSITIONAL restriction of STRIPS is considered. Under this restriction, neither the domain nor the operators can contain quantified sentences or functions. The states and the operator lists are restricted to sets of propositional atoms. Negated preconditions and goals are allowed.

Problems that can be cast in the PROPOSITIONAL STRIPS formalism are DECIDABLE.

b. TWEAK formalism

- Uses the STRIPS restriction of operators (precondition, add, delete lists) .

- Limits the state and operator descriptions to conjunctions of literals (atoms and negated atoms) .

- Allows partially specified (incomplete) initial states .

Although there has been much controversy about the decidability of problems cast in the TWEAK formalism, it is now known that the planning in TWEAK is DECIDABLE, provided that both (1) functions are not allowed, and (2) there are finitely many constants in the domain.

Later in this presentation we cover the SAS+ planning formalism, which is basically a variant of propositional STRIPS, but uses multi-valued state variables (see 4.). Planning in SAS+ is decidable, and many interesting complexity results have been proven for this formalism (see 5.).

1.3. Inherently Hard Problems

Many problems are inherently hard. Regardless of the formalism in which such problems are formulated, they have the potential of requiring enormous resources (time) for their solution. A simple and somewhat striking example of such problems is the problem of finding an OPTIMAL plan for the BLOCKS WORLD domain. It has been proven [4] that this problem is NP-complete (see 1.4. for overview of NP and other basic complexity classes). (Note that the problem of finding a non-optimal plan in the blocks world is polynomial).

When a PROBLEM is known to be hard, no ALGORITHM can be devised to solve that problem in time less than the inherent complexity of the problem. So, in the worst case, any algorithm solving the problem will take enormous amount of resources (time).

The planning process for inherently hard problems can be improved (on the average case) by using domain-specific heuristics. However, the domain-specific heuristics have the disadvantages of (usually) being very hard to find, and being different for different domains.

1.4. (Digression) Basic Complexity Classes

Computational problems can be classified in the following major classes according to their complexity:

a. Polynomial (tractable) - for each problem of this class an algorithm is known that solves the problem in time that is upper-bounded by a polynomial function of the problem size.

b. NP-complete (almost certainly intractable) - no algorithm is known that solves these problems in polynomial time. On the other hand, none of them is being proven to require exponential time. It is BELEIVED that NP-complete problems are intractable. If any NP-complete problem can be solved in polynomial time, then ALL NP-complete problems are provably solvable in polynomial time. A lot of NP-complete problems are well-known, important practical problems, all of which have resisted the attacks of researchers and practitioners trying to solve them in polynomial time. This provides further evidence that NP-complete problems are intractable.

c. P-SPACE complete (almost certainly intractable) - similar to the NP-complete problems, this (larger) class of problems in addition requires polynomial SPACE for their solution.

d. Provably Intractable Problems - for any problem of this class it has been PROVEN that they will take time that is lower- and upper-bounded by exponential function of the problem size.

Of all these classes of problems, only the tractable (polynomial) problems can be solved (in principle) in the worst case for any reasonably big problem size. Due to the astonishing growth rate of exponentials, the intractable (including the NP-complete) problems can not be solved in reasonable time. Usually, intractable problems with problem size of more than 30 elements are considered too hard for the current technology.

2. DEALING WITH COMPLEXITY

Many practical planning problems are intractable, meaning that they can not be solved satisfactorily in the general case. Several approaches have been proposed and used to address the solution of such problems:

a. Reactive Planning (Chapman) - instead of generating plans in ADVANCE, the agent has a prepared set of useful rules of behavior and starts acting in the world. At any given point, he decides on the following step based on the current state of the world and the set of rules of behavior.

   This approach certainly has its merits; however, it is difficult to guarantee any formal properties of the process, such as whether the agent will reach the goal (if adequate sequence of actions does exist), etc.

b. Restrict the Generality - instead of attacking the most general problem, try to solve a restricted version of the problem. Two basic approaches can be followed when trying to identify the most general TRACTABLE subproblem:

   - start with the most general problem, and add restrictions until a tractable approximation of the problem can be modelled.

   - start with a provably tractable problem, and relax restrictions as long as the problem is still tractable.

   One of the advantages of the SAS+ formalism (see 4.) is that it seems to support well the modelling of problems according to this approach.

   The challenge with this approach is to have expressive enough formalism to capture general planning problems, and at the same time to support easy-enough formulation of constraints that can be used to restrict planning problems to tractable, yet practically interesting ones.

   Since any given formalism will be overexpressive for some planning problems, any yet limited for others, it is natural to try to identify subclasses of problems that have mutually similar properties, and utilize such properties to (1) devise tailored algorithms for the corresponding problems, (2) prove the basic characteristics of the subclasses, and then (3) analyze how to reduce each subclass to a tractable subclass of problems. Note that no domain-specific heurustics is used, although the PROVABLE properties of the problems ARE being used to identify the subclasses.

c. Relax the Completeness Criterion - if the failure to find a plan (that otherwise exists) can be tolerated, then algorithms can be devised that can generate the plans (the ones they are capable of generating) in polynomial time.

d. Use Domain-Specific Heuristics - the performance of the planning algorithm can be improved in the average case using the domain-dependent heuristics. However, the domain- -specific heuristics have the disadvantages of usually being very hard to find, and being different for different domains.

To satisfactorily solve a practical planning problem, often a combination of the above techniques will be employed.

3. COMPLEXITY OF STRIPS PROPOSITIONAL PLANNING - SUMMARY

We breifly summarize some known results on the complexity of STRIPS propositional planning, as presented in [2]:

Note:
- '*' stands for 'any number of'
- '+' stands for 'positive (non-negated)'
- each number specifies the maximum number of corresponding entities.

  For example, '2 + precond' stands for "each operator has AT MOST two positive preconditions".

The restrictions are on the maximum number and type of allowed preconditions and postconditions of the operators.

Complexity Class	Examples of problems (each line defines a separate problem class)		
PSPACE COMPLETE:	* precond,	* postcond	
	* precond,	1 postcond	
	2 + precond,	2 postcond	
NP-HARD:	1 precond,	* postcond	
	1 + precond,	2 postcond	
NP-COMPLETE:	* precond,	* + postcond	
	1 precond,	1 + postcond	
POLYNOMIAL:	* + precond,	1 postcond	
	1 precond,	* postcond,	number og goals limited by a constant
	0 precond,	* postcond	

The results clearly illustrate that even in PROPOSITIONAL STRIPS, which itself is a very restricted formalism, most of the problems are very hard, even with additional restrictions on the number of preconditions and postconditions of each operator.

The results can be restated as follows [2]:

(1) Propositional planning is PSPACE - complete even if each operator is limited to one postcondition (with any number of preconditions)

(2) It is PSPACE-complete even if each operator is limited to two positive preconditions and two postconditions

(3) It is NP-hard even if each operator is restricted to one positive precondition and two postconditions

(4) It is NP-complete if operators are restricted to positive postconditions, even if operators are restricted to one precondition and one positive postcondition

(5) It is polynomial if each operator is restricted to positive preconditions and one postcondition

(6) It is polynomial if each operator has one precondition and if the number of goals is bounded by a constant

(7) It is polynomial if each operator is restricted to no preconditions

4. THE SAS+ FORMALISM

4.1. Background

The SAS+ planning formalism [1] is based on planning experiences in sequential control applications in industry, where the PROVABLE correctness and tractability (efficiency) are strongly required.

The SAS+ formalism is similar to the STRIPS formalism in that it relies on precondition/add/delete lists for operator representation. The major difference is that the domain (and therefore operators, too) is modelled by a multi-valued (as opposed to boolean TRUE/FALSE) variables.

Another slight difference is in the definition of the operators: in SAS+ each operator has three conditions:

(1) preconditions that must be satisfied before the operation (and are CHANGED by the operation)

(2) post-conditions that are asserted (or deleted) by the operators

(3) prevail-conditions that are required to be satisfied before the operation, and are NOT changed by the operation.

One might ask why a new planning formalism would be useful, especially knowing the result [1] that it is strictly equivalent in its expresiveness to the traditional propositional STRIPS. The reasons may include: (1) SAS+ seems to be better suited for sequential control applications, (2) it seems to be easier to identify restrictions in SAS+ that seem relevant to practical applications, (3) it provides a different, alternative, fresh look at the planning problems.

4.2. An Example of a Planning Problem in SAS+

The traditional BLOCKS WORLD can be modelled in SAS+ using two state varibles for EACH block:

- Position of the block (on another block or on the table)

- Is the block clear (true/flase)

Each state can be conveniently represented by a tuple of the state variables. For example, given a domain with three blocks (A, B, C), where in the initial state A is on B, and B and C are on the table, and the goal state is A on B, B on C, and C on table, can be represented as follows:

```
State Tuple
(template)           (PosA,  ClrA,  PosB,  ClrB,  PosC,  ClrC)

Initial State        (  B,   true,  Table, false, Table, true)
Goal State           (  B,   true,    C,   false, Table, false)

Move A-from-B-to-C
Preconditions        (  B,     u,     u,      u,     u,    u)
Postconditions       (  C,     u,     u,    true,    u,  false)
Prevail-conditions   (  u,  true,     u,      u,     u,    u)
```

(note: 'u' stands for "doesn't matter")

It is clear that there would be a lot of ground operators defined in the above fashion. However, nothing prevents us to use operator templates, using variables. The above ground representation is more convenient when complexity issues and expressiveness are analyzed.

4.3. SAS+ Planning Subproblems

Several standard problems can be stated for problems expressible in SAS+:

(1) SAS+ plan existence problems:

- unbounded plans ("is there a plan of any length ?")
- bounded plans ("is there a plan with no more than 'k' steps ?")

(2) SAS+ plan search problems:

- unbounded plans ("find a plan, doesn't matter how long")
- bounded plans ("find a plan with no more than 'k' steps")

(3) SAS+ minimal plan search:

- find a plan with minimal number of steps.

Of course, the same problems can be stated for any other planning formalism. However, SAS+ formalism seems to allow easier analysis of the complexity of such problems, especially for restricted problems (see 4.5.).

4.4. SAS+ Expresiveness

It has been proven [1] that SAS+ formalism is equally expressive at least to the following 'standard' propositional STRIPS planning formalisms:

(1) Propositional STRIPS without negative goals
    ('Classical Propositional STRIPS')

(2) Propositional STRIPS with negative goals

(3) Ground TWEAK

An interesting corollary of the proof is that the above formalisms (1), (2), (3) themselves are mutually equally expressive.

It follows that any problem that can be cast in any of the above 'standard' planning formalisms can be cast in SAS+, too.

It has also been proven [1] that the plan existence problem for any of the above formalisms and the SAS+ formalism can be POLYNOMIALLY reduced to the plan existence problem expressed in any of the other three formalisms.

From above, and
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil
])
From the fact that (for example) ground TWEAK is PSPACE-complete, it directly follows that the (unrestricted) SAS+ plan existence problem is PSPACE--complete (very hard, almost certainly intractable).

4.5. Dealing with Intractability in SAS+

Four special types of RESTRICTIONS have been defined in SAS+, and their impact on tractability has been analyzed in detail by Backstrom [1]:

(1) P (post-unique) - different operators can NOT change the SAME state variable to the SAME value

(2) U (unary) - each operator changes EXACTLY one state variable

(3) B (binary) - each state variable can have only TWO values

(4) S (single-valued) - different operators requiring the SAME state variable to remain constant during their occurence ('prevail-condition') must also require the SAME CONSTANT VALUE of the state variable

For example [1], the single-valuedness prevents the domain
From
including two operators such that one requires a certain room to be lit during its occurence, while the other requires the same room to be dark.

The restrictions can be arbitrarily combined, and such restrictions are designated by appending the above letters associated with each individual restriction. For example, 'PUB' designates a subclass of SAS+ problems having at the same time the following restrictions: post-unique, unary, and binary.

One especially important (at least theoretically) SAS+ subclass is PUS, for which a tractable algorithm exists that answers all standard (4.3.) planning problems.

5. SAS+ COMPLEXITY RESULTS AND SAS+ THEORETICAL IMPORTANCE

5.1. Tractable SAS+ problems

It has been proven [1] that PUS is the maximal fully tractable problem with respect to the four possible restrictions (since PUBS is a subclass if PUS, it is tractable, too). So only two of the possible 16 subclasses of SAS+ are fully tractable.

The SAS+ US (and so SAS+ UBS) problem is tractable for the non-optimal, unbounded plan search (and so plan existence) problem.

A polynomial algorithm has been devised for planning in SAS+ PUS. Note that the algorithm guarantees the OPTIMAL solution.

An example of a problem that can be cast in SAS+ PUS is a restricted version of the blocks-world problem, where one is NOT allowed to move the blocks
From one block to another (but only to or
From the
table).

5.2. Summary of the Complexity Results for SAS+ Planning

We summarize the complexity results for various problems cast in the SAS+ formalism.

	Plan Existence		Plan Search	
	unbounded	bounded	unbounded	bounded
PUBS	POLYNOMIAL	POLYNOMIAL	POLYNOMIAL	POLYNOMIAL
PUS	POLYNOMIAL	POLYNOMIAL	POLYNOMIAL	POLYNOMIAL
UBS	POLYNOMIAL	NP-complete	POLYNOMIAL	NP-complete
PUB	unknown	NP-hard	intractable	intractable
PBS	unknown	NP-hard	intractable	intractable
US	POLYNOMIAL	NP-complete	POLYNOMIAL	NP-complete
PB	unknown	NP-hard	intractable	intractable
PU	unknown	NP-hard	intractable	intractable
PS	unknown	NP-hard	intractable	intractable
UB	PSPACE	PSPACE	intractable	intractable
BS	PSPACE	PSPACE	intractable	intractable
P	unknown	NP-hard	intractable	intractable
B	PSPACE	PSPACE	intractable	intractable

U	PSPACE	PSPACE	intractable	intractable
S	PSPACE	PSPACE	intractable	intractable
unrestricted	PSPACE	PSPACE	PSPACE	PSPACE

It is interesting to note that it has been proven that the minimal plans in all but PUBS, PUS, UBS, and US restrictions can have exponential length with respect to the problem size. Therefore, the PLAN SEARCH PROBLEM can NOT be tractable anyway for such problems (generating an exponentially long plan can not be done in polynomially bounded time).

5.3. SAS+ Theoretical Importance

The analysis of SAS+ planning made by Backstrom [1] significantly extends our understanding of the complexity of planning. SAS+ provides a well-founded framework for future research on the complexity of planning.

SAS+ provides the first truly interesting fully tractable planning class (SAS+ PUS). In addition, SAS+ shows that releasing any of the restrictions of this class leads to intractability.

Another very important contribution of SAS+ is the proof of the strong mutual equivalence of the expresiveness of the 'standard' propositional planning formalisms, and (in addition) their equivalence to SAS+ planning formalism.

Finally, SAS+ demonstrates one feasible approach for handling the tractability of planning problems. By defining other restrictions (other than P, U, B, S), one may model various subclasses of planning problems and try to identify tractable, yet practically interesting subsets of planning problems.

6. CONCLUSION

Planning problems are in general computationally very hard. Therefore, various restrictions must be applied in order to allow solving realistic, practical problems.

Provably complete and provably polynomially upper-bounded algorithms are always a nice thing to have, and for some critical applications (such as industrial control applications) they are a basic requirement.

One feasible approach to handle the complexity of planning is to identify tractable subclasses of planning problems. It can be done by starting with a reasonably general class of problems, and then removing the not-so-badly needed expresiveness 'features' until a tractable subclass tailored to the specific need is identified.

It seems that many of the available options for reducing the complexity may have to be applied in CONJUNCTION in order to satisfactorily solve all but the most trivial realistic planning problems.

7. REFERENCES:

[1] Christer Backstrom, "Equivalence and Tractability Results for SAS+ planning", in Proc. 3rd Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR-92), pg 126-137, Cambridge, MA, USA, October 1992.

[2] Tom Bylander, "Complexity Results For Planning", in IJCAI [1991], pg 274-279.

[3] Kutluhan Erol, Dana S Nau, and V S Subrahmanian, "On the Complexity of Domain-Independent Planning", in AAAI [1992], pg 381-386.

[4] Naresh Gupta and Dana S Nau, "On the complexity of blocks-world planning", Artificial Intelligence, 56:223-254, 1992.

****

END OF NOTES

From THE SESSION ON APRIL 25th

****