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 93-003

(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 93. The seminar turned out to be mostly about classical planning techniques. Each class, a designated student took notes and mailed them to the class. A compilation of these notes appears below. A couple of times, interesting mails from outside colleagues were cc'd to the list.

Subbarao Kambhampati

```
From rao  Mon Feb  1 01:41:20 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA22452; Mon, 1 Feb 93 01:41:20 MST
Date: Mon, 1 Feb 93 01:41:20 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9302010841.AA22452@parikalpik.eas.asu.edu>
To: plan-class
Subject: Notes for the first four classes --by Rao
Reply-To: rao@asuvax.asu.edu

Notes for the first four classes of Planning Seminar

Written by Subbarao Kambhampati [Feb  1, 1993]

--INTRO

Start with some intro to planning (definition of the planning problem,
what makes it interesting, what makes it hard etc.)

Then go to classical planning assumptions (single agent, no
uncertainity, world stands still)

 --Point out that these assumptions are made to simplify the
problem for the present, and that we will ultimately look at ways of
relaxing all these assumptions.

The issues in classical palnning are representation and
search/reasoning

--SITUATIONAL LOGIC

First obvious idea is to use logic as a representation language and
logical inference (theorem proving) as a vehicle for doing planning.

discuss situational calculus representation,

-world represented as situational assertions
--actioms represented as situational axioms

--the idea of casting planning problem as one of shows that there does
exist a situation where all the goal wffs will be true.

SHow a simple planning example (e.g.,, blocks world putting A on top
of B when they are clear in the beginning)

What are the problems:

 --Frame problem, qualification problem, ramification problem
    (relate the last one also to the use of situation independent
    axioms -- the socalled physicial laws -- to derive indierect
    effects. This practice gives modularity-- but also adds the
    problem of having to do itra-situational theoremproving to
   compute the indirect effects)

 -- point out that all the problems are in a sense fundamental

 --talk about how all the possible solutions to these problems
   will involve using default reasoning (eg. assume things don't
   change by default, assume that the preconditions given are the only
   ones by default, assume that the effects given are the only ones by
   default)

So, two options-- put planning on back bunrner and work on ways of
doing temporal non-monotonic reasoning, or try to short circuit the
frame/qualification/ramification problems to the extent possible by
choosing simpler representations.

We will take the second approach, and look at STRIPS representation:

--STRIPS REPRESENTATION

in Strips representation, actions are represented with add delete and
precond lists. States are represented "extensionally" by the set of
assertions (fluents) that are true in the state. State change is
described extensionally -- an action A can be applied to a state S if
all the assertions in the precondition list of A are true (unify) in
in state S. In such a case the new state resulting from application of
A to S  A(S) is computed extentionally as S - D +  A.

Qualification problem is short circuited by assuming that ALL
preconditions of an action are present in the P list of an action
(nothing is left out). Ramification problem is shortcircuited by
assuming that every possible effect (either direct or indirect) of an
action is described in its A & D lists. Frameproblem is shortcircutied
by assuming that anything in S that doesn't appear in A&D will go
unscathed to the new state A(S).

[Comment: original strips used a slightly more expressive
representation than this-- rather than check if a precondition is
directly present, it checks to see if the precondtion is "derivable".
Shift from one state to other is still accomplished by the S -D +A
rule.  Thus seen from situational calculus viewpoint, Strips removes
inter-situation theorem proving, but keeps intra-situation theorem
proving. See Liftschitz paper regarding semantic problems of this rep.]

[note: It is instructive to point out the various ways in which strips
representation is less expressive than situational calculus
representations: You don't have synergistic effects, you don't have
domain axioms, and in the vanilla strips representation, you also don't
have conditional effects. ]

[While talking about qualification problem, we can talk about Simmon's
work on using a simpler domain model to do planning, and a more complex
domain model to do simulation. The tradeoffs in that enterprise.]

--GENERATING PLANS WITH STRIPS REPRESENTATION


Given initial and goal states, and actions represented in strips
action representation, the first obvious idea is to cast planning as a
process of finding a path from initial state to goal state. This is the
canonical graph-search problem, which can be shown to be P-Space Complete
(i.e., it belongs to the class of hardest problems solvable in polynomial
space. The result is attributed to Canny).

[clarify the phrases  "search state" and "world state". Search state
may are may not correspond to world state. ]

Two first ideas:
```

Forward planning (or Forward search in the space of states):

Initial state of the search: Initial state of the problem
Goal criterion: any state which subsumes the assertions in (i.e, is a superset of goal state)
Children generation: given a search state S, generate one child for each possible operator instance that is applicable in S. Child is generated by the usual operator application rules S -D +A.

Forward planning is sound and complete. It can generate "optimal plans" if we use an admissible search strategy

Backward planning (or Backward search in the space of states)

Since the goal state of the search is extensionally represented, we can also do backward search from goal state towards initial state.

The initial state of the search will be the goal state of the problem, and the termination condition of the search will be that the current state is a subset of the initial state of the problem.

The only tricky part is computing the children states. When you are at a state S, and you want to see what are the children states, you need to think of all possible actions that could have got you this state, and for each such action compute the description of the state that needs to be true before that action so that the current state would result after the action. This latter computation is known as REGRESSION. You regress the current state over an action to compute the state that needs to be present before the action so that the current state will happen after the action.

Regression is quite easy to formalize-- Suppose you have an actin A and a state S and you want to find regression of S over A (R(S,A)). We do this by regressing each individual assertion f in S over A separately. For f to be true after A, either A must have made f true, or f was already true before A was done, and A doesn't delete f. In either case, preconditions of A must also be true in the preceding state so A can take place. Accordingly, if P(A) are teh preconditions of A. Then R(f,A) = P(A) if A adds f, and P(A) & f & Preserves(f,A) if f is not added by A (here preserves(f,A) is the constraints that will ensure that A will not delete f.) This process can be mechanized quite easily, and can form the backbone of backward searching planners's child generation routine. Regression also has other possible uses in planning-- we will see this below, when discussing means ends planners.

[note: Regression of f over A may sometimes give rise to disjunctive states. For example, in blocksworld suppose we want to regress On(x,A) over the action Stack(B,y). If x=B and y=A, then On(x,A) regresses to true. If x!=B and y=A, or x=B and y!=A, then On(x,A) regresses back as On(x,A) (but will eventually lead to an inconsistent state since atmost one block can be on top of another and vice versa), and finally if y!=A and x!=B, then On(x,A) regresses to on(x,A)]

--EXPLOITING SUBGOAL INDEPENDENCE

Forward and Backward searching planning algorithms, while being sound and complete, are not exactly customized in any sense for planning problems. They just look at planning problems as yet another search.

The next step is to see if there are any particular type of regularities/assumptions that are more reasonable for planning problems, and if so, think of ways of exploiting them.

One of the assumptions about planning problems is what is called "subgoal independence" -- that is, given a conjunct of goals, you can work on each goal separately, find a plan for solving it, and concatenate all the plans in some arbitrary sequence to solve the conjunctive goal.

To the extent we believe that subgoal independence is more the rule rather than the exception for planing problems, we will do better by writing our planning algorithms to _exploit_ subgoal independence.

Ofcourse, independence is really only an idealization in that not every problem is going to have independent subgoals. So, what we really want is to for our algorithms to exploit independence where available, but still solve the problem even otherwise.

One possibility is to make the algorithm assume subgoal independence by default, but have a recovery strategy, when the assumption is seen to be wrong.

STRIPS is one the earliest and most famous planning algorithms that used this idea. Strips actually uses a Means Ends Analysis algorithm. It does goal directed forward search. Strips search state contains the current state, the current stack of subgoals.

Describe STRIPS in a simple example. Show how it solves the problems (Once it picks a goal to achieve, it will work on the goal and all its subgoals, before it picks another goal at the same level). recovers from the subgoal independence assumptions [When it splits a conjuctive goal into indpendent parts and puts them on the stack in some order, it keeps the conjunctive goal on the stack, as a check. After the individual conjuncts are made true separately, the full conjunctive goal is checked once again. If it is not true, then the independence assumption is violated. Strips attempts to split the conjuncts into a different order.]

Choice points for strips: (1) Which order to split the conjunct into (2) which operator instance to use to make a conjunct true (the latter is split into selecting the operator schema, and selecting bindings).

STRIPS is a sound but not a complete planner--i.e, if strips returns a plan it will be a correct plan. But, even with an admissible search strategy STRIPS may sometimes _fail_ to find optimal plan (e.g. sussman anomaly) and sometime fail to solve the problem (e.g. register swapping problem).

What is the best characterization of the problems that strips CAN solve and give optimal solutions? Is it independent subgoals? Independence of subgoals is sufficient but not necessary. The necessary and sufficient condition turns out to be "Serializable Subgoals" (Korf) -- Strips can solve a problem and give optimal solutions (modulo an admissible baselevel search strategy) if and only if there exists at least once sequence in which the individual goals can be attacked one after other, without undoing previously achieved subgoals.

[Digression: Some of the classic problems with Non-serializable subgoals -- such as Sussman anamoly and Register Swapping problem can be made serializable by augmenting the set of goals with additional subgoals -- eg, in sussman anomaly, the subgoal of putting C on table, and in register swapping the subgoal of making the third register have the value of one of the first two registers. However, this is not the case for all non-serializable subgoal problems -- eg. the one way rocket problem cannot be augmented with any additional subgoals such that it becomes serializable. Finally, serializability is a global property and is affected by the initial state, the goal state and the action representation-- example, register swapping problem is nonserializable, while a very similar block swapping problem is serializable.]

Note that although STRIPS is not Complete, the normal forward searching planning algorithm IS complete. So, why should anyone even look at STRIPS instead of a forward searching planning algorithm? The answer is that STRIPS and similar algorithms are written to exploit subgoal independence if it is present, where as normal forward search does not differentiate between problems with independent and problems with dependent subgoals. So, it takes the same time no matter what. [At this point, it can be said that forward searching planner can be improved through a heuristic which makes it exploit subgoal indepndence. Two points can be made in response 1. it is quite hard to do this unless the forward searching planner keeps track of the goal stack and 2. once it keeps the goal stack, it is acting very similar to STRIPS...]

--Linearity Assumption/Linear planning

The assumption that a conjunctive goal can be solved by solving the conjuncts in some arbitrary linear order (i.e., without any interleaving of subgoals) has been called "linearity assumption". Clearly, planners using linearity assumption can only (optimally) solve problems if they have serializable subgoals.

Originally, the term "linear planning" has been used for planners which use linearity assumption, and nonlinear planning is thus used for planners which used no linearity assumption (i.e., allow subgoal interleaving during planning).

Over time however, "linear planning" has also come to be identified with all total ordering planners, while nonlinear planning came to be identified with planning with partially ordered (and partially instantiated) plans. This is highly misleading since some total ordering planners (such as Waldinger's planner, RSTRIPS etc) don't make linearity assumption and are thus able generate optimal plans for even problems iwth non-serializable subgoals.

I would recommend that the terms "linear planning" and "nonlinear planning" be avoided. Instead, we could use phrases like "Total ordering planner with linearity assumption" (STRIPS), "Total ordering planner without linearity assumption" (Waldinger's), "Partial ordering planner" etc.

--GOING BEYOND STRIPS

In going beyond strips, our immediate interest is to see if STRIPS can be made COMPLETE in addition to being sound.

What possible changes can we do to STRIPS such that this can be accomplished?

One obvious choice of course is to renounce the linearity assumption and start using the goal stack as more like a goal list. If we do this, then we essentially go back to a version of forward searching planner.

What other possible changes can be made to STRIPS such that it can avoid the incompleteness while retaining, as much as possible, the advantages of goal-directed reasoning under linearity assumption?

One possibility is to note that when STRIPS fails, it often the whole plan and looks for another possible goal ordering, even though the plan it has until that point is probably almost correct, if only certain steps are switched. What we would really like to do is to pick any order of achieving the goals, and complete planning without giving up and changing the order (thereby redoing a lot of planning).

To be able to do the above however, we need to stop confounding PLANNING ORDER, i.e., the order in which individual goals are achieved during planning in a particular search path, with the execution order, and be ready to MODIFY the current plan midway through a search path. This requires keeping plan in the search space, and making some of the planning decisions based on the current plan. (Although STRIPS does keep the current plan in the search state, it never really USES the plan--it is just an appendage to the search state).

[Thus, the only way to separate planning order from execution order is to keep "Plan" in the search state, and modify it during planning.]

--State based planners that modify current plan during planning

The above discussion brings us to Waldinger's planner and RSTRIPS, which are state-based planners which modify the plan during the search.

Consider Waldinger's planner (WP) working on making On(A,B)&On(B,C) true from base state where all three blocks are on table. The planner starts by picking some random order in which to achieve goals. However, once it picks an order, it NEVER has to backtrack on the goal order (ie., plans can be produced in any planning order). Suppose it starts by picking On(B,C) --> On(A,B) as the planning order. In this case, WP acts verymuch like STRIPS, and completes the plan.

Suppose, WP picks the opposite order On(A,B) --> On(B,C). When it makes On(A,B) true using the plan

Pickup(A)-->Stack(A,B)

It remembers that On(A,B) needs to be true by the time planning is complete--so it needs to be protected (such protections are also done for preconditions of the of Pickup(A) --> Stack(A,B) true). Next, it decides to work on On(B,C). It continues to add steps for making On(B,C) ture until it finds that one of the steps undoes the protected assertion On(A,B). At this point, it realizes that On(B,C) can't be made true after the action Stack(A,B). So, it attempts to find another place within the current plan where On(B,C) can be made true.

There are two other spots in teh current plan -- immediately before Stack(A,B) and immediately before Pickup(A). For completeness, WP has to consider both the possibilities.

Suppose it picks the choice of attempting to achieve On(B,C) before Pickup(A). Of course, since what WP wants is for On(B,C) to be true at the end, it really needs to achieve R(R(On(B,C), Stack(A,B)), Pickup(A)) rather than On(B,C) at Pickup(A) (Where R(a,b) is the operation of regressing a over the action b).

In this case, the regressed value of On(B,C) before Pickup(A) is On(B,C). So, we attempt to make On(B,C) true here, by backward chaining. We introduce the actions Pickup(B) and Stack(B,C) in that order (detail: when we introduce Pickup(B), it deletes armempty(), which is being protected as it is required at Pickup(A). However, all is not lost since, the next action Stack(B,C) will restore armempty back. Thus, temporary protection violations can be tolerated.).

At this point, as long as we didn't undo any protections setup while making On(A,B) true, we are done solving the problem.

Note that we didn;t ever have to backtrack and consider the goals in a different order.

[Similar discussion can be done for Register Swapping problem also, which again is solvable for WP. Left as an exercise.]

[Detail: WP will not be complete if it doesn't look at all possible ways of making an assertion true. In particular, just because an assertion is already true doesn't mean that NO-OP is the only action that need be considered. WP has to also look at choices of making the assertion true by adding actual actions. Example: On(A,B)&On(C,Table) in initial state, and On(A,B)&On(B,C) are the goals, and suppose WP picks the order On(A,B) and On(B,C) for planning. If it attempts to make On(A,B) true through NO-OP, then it will get stuck for the next goal. If however, it also notes that On(A,B) can be made true either by the action No-Op or by the action sequence "Pickup(A)-->Stack(A,B)", then, the other goal, ON(B,C) can be regressed over this plan..]

--going from state based search to plan based search

One of the annoying things about all the previous planners is that while some of them do avoid confounding planning order with execution order, all of them insist on ordering the actions totally. Often times, we don't know apriori what the execution order of the actions corresponding to two goals are. In such cases, we would really like to keep the actions unordered to begin with, and put orderings between them as and when required.

What do we need to do to make this possible in a systematic way?

To begin with, we should note that all state-based planners discussed above insist on totally ordered plans, since they need complete description of the state preceding an action so they can simulate the action and compute the state after the action. If we go to partially ordered plans, the state before an action is NOT completely defined (although we can tell that certain things will be necessarily true before an action, there may be certain other things that may or may not be true based on the exact total order in which a partially ordered plan is executed.)

[While able to shift the order of the steps (through regression) within a search branch, it still assumes total ordering plans for doing the backward chaining part of its planning.]

The question is: do we really need the state-based representations? Do we really need complete description of the state preceding actions?

The reason we needed states has been that from the beginning, we looked at planning as a process of searching in the space of world-states, going from initial state to the final state. In this view, the plan is implicit as the sequence of state transitions. The correctness of a plan is judged essentially by simulating the execution of its individual steps from the initial state, and checking to see if the final state subsumes the goal state.

There is an alternative view of planning--that of characterizing it as a search in the space of plans. In this view, we start planning with a null plan which is a plan with two dummy actions a-I and a-G, where a-I is the dummy action corresponding to initial state--it has no preconditions, and has only effects corresponding to all the assertions in the initial state. The dummy action a-G corresponds to teh goal state-- it has no effects, and has only preconditions corresponding to all the assertions in the goal state.

Starting with this null plan, the idea of planning is to REFINE the null plan (add steps, orderings and other constraints) such that the plan becomes more and more correct.

To operationalize this, we need to precisely define the correctness of a plan.

A plan is correct, as long as every precondition of every action of the plan is necessarily correct in the situation preceding that action. Note that in this view, we are only asked to guarantee that the preconditions of an action are necessarily true-- we don't need to provide a complete description of the STATE preceding the action!!! This view thus effectively gets us out of state-based representations.

Obviously, the null plan is not correct, since the preconditions of a-G are not in general true in the beginning. Preconditions of an action which are not yet necessarily true are called Open-conditions of that action.

Planning proceeds by trying to make all the open-conditions necessarily true. For example, we may add a new action a-j to make one of the open conditions of a-G true. When we do this, of course, we are also introducing the preconditions of a-j as open-conditions of the plan. When there are no more open conditions in a plan (i.e, all of them are made necessarily true simultaneously), then we will say that planning is complete.

The discussion above gets us into planning as a search in the space of plan. It is instructive to note that search in the space of plans SUBSUMES search in the space of states. If we use totally ordered plans and insist on adding steps to the beginning or end of the plan, then we are essentially being equivalent to Waldinger's planner, and other regression based MEA planners (such as RSTRIPS described in Nilsson).

---Truth criteria

In the discussion above, we noticed that making a plan correct involves making its open-conditions necessarily true before the respective actions. To do this, we need to know two things:

1. how do we tell if a condition c is necessarily true before an action a-j in a plan ?

and

2. if c is not necessarily true before a-j in P, HOW DO WE make it true?

The first question is answered by providing the necessary and sufficient conditions (i.e., weakest conditions) that need to be satisfied by the plan for c to be necessarily true before a-j. Specification of such conditions is called a TRUTH CRITERION of a plan.

Once the truth criterion is known, ONE answer to qn 2, ie., ways of making a condition necessarily ture, is available by simply inverting the truth criterion.

Truth criterion of a plan depends on the representation of the plan (i.e., is it totally ordered? partially ordered? what sort of action representation is it using? etc.)

Consider a totally ordered and totally instantiated plan P which is made up of actions in strips representation.

In such a plan a condition c is necessarily true before at an action a-j if and only if some action a-k which precedes a-j has an effect (add list element) c and no actions coming between a-k and a-j deletes c.

[Although we could have said that c is true at a-j as long as the action immediately before a-j adds c, that would only be a sufficient, but not necessary condition].

Next, we will look at a truth criterion for partially ordered plans.

```
From catrava@seine.eas.asu.edu Wed Feb  3 14:07:01 1993
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 "^From:" nil nil nil])
Return-Path: <catrava@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 AA25442; Wed, 3 Feb 93 14:07:00 MST
Received: from severn.eas.asu.edu (enws295) by seine.eas.asu.edu (4.1/SMI-4.1)
        id AA23328; Wed, 3 Feb 93 14:04:06 MST
Message-Id: <9302032104.AA23328@seine.eas.asu.edu>
From: catrava@seine.eas.asu.edu (Serban Catrava)
To: cohen@enws318, annaredd@seine, catrava@seine.eas.asu.edu,
        dchen@cibecue.azo5.bull.com, pashupat@seine, gary@enws320.eas.asu.edu,
        wtsai@enws320.eas.asu.edu, Ihrig@enuxha, suresh@enuxha, gopi@enuxha,
        elder@enuxha
Cc: rao@enws228.eas.asu.edu
Date: Wed, 3 Feb 93 14:04:06 MST
```

Notes for the Feb, 2 class of Planning Seminar

Written by Serban Catrava

-- GENERAL TOPICS

1. Partial Ordering vs. Total Ordering in planning
2. Heuristics in planning

I) Motivations to switch from State to Plan based planning

-- in a state based planning the order of plan generation should be the same with the order of plan execution, while in a plan based planning and execution order may be different.

-- in plan based planning it is not necessary to (completely) specify a state. You only need to know what preconditions must hold true for each action.

-- in a plan based planning you are not limited of thinking in terms of transitions from one state to another (in contrast with plan execution which naturally shows the control flow from one state to another).

-- NB: search in the space of plans performed by adding steps only "at the beginning or at the end" degenerates in a trivial state based planning

II) Correct plans

Necessary Truth:
Assuming that a predicate P must hold true before an action S can be executed, what are the constrains the plan has to satisfy so that P is true?

Conditions
1. the step performed just before S adds P (makes it true)

OR

2. somewhere along the prev steps, an action adds P AND no latter step (a step between the "adding" step and S) deletes P

DEFINITION
A plan is said to be correct if all preconditions are satisfied (necessa
rily
correct according to necessary truth criterion).

III) Plans with partially instantiated actions

```
action: puton (A, B, C)
semantic: take A from B and put it on C
status: fully specified

plan:   puton (A, B, C) -> puton (B, D, E)
semantic: take A from B and put it on C THEN
          take B from D and put it on E
status: fully specified
```

BUT

```
action: puton (A, B, ?X)
semantic: take A from B and put SOMEWHERE (at this time)
status: partially instantiated

plan: puton (A, B, ?X) -> puton (B, D, ?Y)
semantic: take A from B and put it somewhere THEN
          take B from D and put it somewhere
status: partially instantiated plan
```

Why generating partially instantiated plans?

1. delay commitment
2. solving subgoals will not impose overconstrains for the rest of the subgoals

```
When commit?

1. When forced (unification)
2. At plan execution


IV) Necessary Truth in partially instantiated plans

A condition C is necessary true before an action S
If AND ONLY IF
        1. exist S' preceding S such that S' has an effect E AND
                E ~ C           ( ~ stands for unifies)
AND
        2. for every S" which comes between S' and S, if E in Delete (S")
                E !~ C          (!~ stands for does not unifies)

V) The planning process

++ Partially ordered / Partially instantiated (POPI) plans

        Idea: Try to keep the plan steps as unordered as possible.

        -- a partially ordered plan corresponds to at least one total
        ordered plan


++ Restricted form of Truth Criterion for POPI plans

        A predicate C will be necessary true for step S if
                1. exist S' such that   S' < S                  AND
                                        exist E effect of S'    AND
                                        E ~ C
AND
                2. for all S" such that S' < S" < S     AND
                                        for all D in delete (S") D !~ C

NB: 1. and 2. are sufficient but correct plans may be rejected by the
    above conditions.


++ General form of Truth Criterion for POPI plans

        A predicate C will be necessary true for step S if
                1. Same as above
AND
                2. for all S" such that S' < S" < S     AND
                                        for all D in delete (S")
                                            if D ~ C then
                                                exist Sw such that S" < Sw < S A
ND
                                                exist E in effect (Sw)
AND
                                                E ~ C


From pashupat@seine.eas.asu.edu  Sun Feb  7 16:25:12 1993
Return-Path: <pashupat@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 AA29352; Sun, 7 Feb 93 16:25:12 MST
Received: from thames.eas.asu.edu (enws294) by seine.eas.asu.edu (4.1/SMI-4.1)
        id AA29011; Sun, 7 Feb 93 16:23:35 MST
Message-Id: <9302072323.AA29011@seine.eas.asu.edu>
From: pashupat@seine.eas.asu.edu (Anand Pashupathy)
Date: Sun, 7 Feb 1993 16:19:10 -0700
X-Mailer: Mail User's Shell (7.2.4 2/2/92)
To: plan-class@parikalpik
Subject: Notes for the Feb 4 class

NOTES FOR THE FEB 4 CLASS

Written By :: ANAND PASHUPATHY

The basic topics of discussion are :

        1. Confusion between linear and non-linear planners
        2. Truth Criterion revisited
        3. Waldinger's planner revisited
        4. How to do planning ?
        5. How does Establishment and declobbering work?
```

1. Confusion between linear and non-linear planners.

STRIPS was seen to be a planner, which worked well for the general class of problems that could be easily serialized. This gave rise to the definition of *STRIP-like* planners being called *linear planners*. NOAH was a step forward from STRIPS, in the sense that, it did not consider total ordering of plans but partial ordering of plans and hence *NOAH-like* planners were classified as *non-linear planners*. To make matters worse, we have planners like Waldinger's planner, which is a non-linear planner in the sense that it solves problems like the register-swapping problems, which are inherently non-serializable and it is a linear planner in the sense that it is a totally-ordered planner. We can surmize from this that, calling planners serializable or non-serializable is not correct and we should stick to definitions like *Partially Ordered Partially Instantiated* planners or something more confusing than that!

To make matters better, we can classify planners to be one of the following types:

- Those which do search in the space of plans or states.
- Those which follow the linearity assumption.
- Those which have a total-ordering restriction.

TWEAK is a *partial ordering partial instantiating* planner, which does search in the space of POPI plans. Waldinger's planner can be classified as a planner which uses total ordering and still does not conform to the linearity criterion. There is an evolution of the planners, starting from STRIPS. STRIPS can solve a certain set of problems, TWEAK improves on it and solves those problems that STRIPS cannot solve, in a more elegant way (Partial Ordering) and finally we have NOAH, which does all that TWEAK does and also does *task reduction*.

2. Truth Criterion Revisited.

A proposition *p* is true before an action *s* in a non-linear plan if

- ESTABLISHMENT : There exists an action *t* which necessarily precedes *s* and provides an effect *n* which necessarily unifies with p.

- DECLOBBERING : For every action *c* such that c can possibly come between t and s and can possibly delete p, there exists another action *w* called the white knight which necessarily comes after c and before s and provides an effect *e* which necessarily unifies with p whenever *d* unifies with p. Here d is the delete literal of p.

If one knows the truth criterion, which provides the weakest proposition p at step s to be true, we can invert the truth criterion and generate plans. To generate plans, we must evaluate the truth value for each such proposition. When we say weakest condition, we mean the necessary and sufficient condition. Search strategies are an important segment of non-linear planners, but they open a completely different can of worms. One of the most important things to remember about the non-linear planners is that, there will ALWAYS exist a solution in the search space, which will be optimal. The search space consists of the children generation function and the start node. Any admissible search strategy should work fine.

3. Waldinger's planner revisited

One of the important question that comes to mind about Waldinger's planner is that, does it reorder the sub-goals? No, it does not and this is one of the major plus point in it's favor. Such planners are complete without having to do backtracking. Generically speaking, one can say that, planners that work in a space of plans, will not backtrack.

In such a scenario, there exists two states, the initial state and the goal state. The initial state has a set of preconditions and the goal state has a set of goal literals. When Waldinger's planner picks up any goal literal, irrespective of what it is, it will never go back and work on it. But this gives rise to a very interesting question. While one of the goal literal is being worked on, what if the preconditions of the other goal literal is destroyed. This will never be the case, because of the following :: *WALDINGER'S PLANNER WILL ALWAYS FIND A PLACE, TO PUT AN ACTION WITHOUT VIOLATING ANY OF THE PROTECTION INTERVAL CRITERION*. The important point to be noted is that the establishers may change, but the thing to be established will always remain the same.

4. How to do planning?

Planning can be done very easily using the following algorithm.

```
  Start

    P <---- Null Plan

      put P in the open list

    LOOP

      1. Pick a plan P from OPEN.

      2. If the plan is inconsistent, go back to LOOP. (efficiency hack)

      3. pick a random condition *c* at step *t* which
         is not necessarily true by MTC (Modal Truth Criterion).

      4. If you cannot pick such a condition, terminate and return P.

      5. Consider all possible refinments of P which will have c
         necessarily true

            add them all to the search queue (OPEN)
            go back to LOOP
```

We can pick up any one condition because the order in which we look at the goals is not important. If we look at all the conditions, then we will give rise to redundancy. As already stated, the planner will generate an optimal plan. Irrespective of what condition we pick, we will always get the optimal plan, time may be a consideration, as some plans may finish faster than the others. We can generate a plans by taking different conditions. Redundancy can be good or can hurt too.

5. How does Establishment and declobbering work?

Let us consider the following example.

(Note : Please refer to the example given in the notes. I will just explain the example and may not be able to draw the figure.)

As is visible from the figure, that we need P(x, y) to be true at S. We can easily see from the figure that it need not necessarily be true, as nothing is defined a priori. We have two options of making P(x, y) true.

One of the options that we have is to make S’ give P(x, y) such that, u codesignates to x and v codesignated to y.

The other option is to get S’’’ (which is another action schema), which also gives P(x, y). Now the question is, where to put S’’’ ? We have two different options at hand to work with.

We need to worry about ESTABLISHMENT. There are three ways of accomplishing it.

Separation :: One of the options here is of *separation*. In this case, we must make sure that the P(a, b) at Sc does not codesignate with the P(x, y) (That is a <> x and b<> y).

Promotion :: The other option is to put Sc after S such that whatever it tries to do, does not affect the goal literal on S.

White Knight : The last option is of the white knight. The white knight works in two ways. One of the specific ways of using the white knight is the principle of *demotion*. In such a case, we must put Sc before S’ such that, what comes immediately before S is only S’ and nobody else. The more general case of white knight would be when irrespective of where Sc comes, there will definitely be a white knight after that Sc and before that S to try and nullify the effect of the Sc. The white knight could be applied in two ways. Firstly, it could be obtained from the existing plan OR could be obtained externally as a new action. Let this be S+ with a action of +P(w, q). S+ should be added such that

Sc < S+ < S.

Further more, we must add enough codesignators such that this holds true. In this case, we would have a = x and y = b => w = x and q = y.

An intersting thing to note here is that, we already know that our plan is represented as P = <T, O, Pi>. When we are trying to make ESTABLISHMENTS or are trying to DECLOBBER, we are making changes to this three tuple, in the form of new orderings, new actions or new bindings.

--
Department of Computer Science and Engg   Anand Pashupathy
Artificial Intelligence Lab   Primitive : 602/965-2735 (o)
Arizona State University,   : 602/921-3633 (r)
Tempe, AZ 85287-5406   Not so Primitive : pashupat@seine.eas.asu.edu

From suresh@enws318.eas.asu.edu Thu Feb 11 10:59:06 1993
Return-Path: <suresh@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA02592; Thu, 11 Feb 93 10:59:06 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA02193; Thu, 11 Feb 93 10:51:18 MST
Date: Thu, 11 Feb 93 10:51:18 MST
From: suresh@enws318.eas.asu.edu (Katukam Suresh)
Message-Id: <9302111751.AA02193@enws318.eas.asu.edu>
To: plan-class@parikalpik

NOTES FOR FEB 9 CLASS

Prepared by: Suresh Katukam

The points discussed in this class are :

1. TWEAK MTC (Modal Truth Criterion)
    - Why is it polynomial?
    - When is it polynomial?
    - What is it really computing?
    - What is modal duality?

2. MTC: Is the full MTC including White Knight really needed for Plannin g?

3. Is POPI (Partially Ordered Partially Instantiated) plan really bett er?

TWEAK has a incomplete (i.e. a partially specified) plan while working which may solve the given problem. This incomplete plan may could be completed in many ways leading to complete plans depending upon the constraints being adde d to it. Planning is completed if all all completions of of the incomplete plan so lve the given problem.

Adding a constraint (remember that TWEAK is a constrint-posting planner) to a incomplete plan can often rule out all the completions (i.e. no plan exists i f proceeded further). In other words, the set of constraints is inconsistent and no longer it defines a valid incomplete plan. At this point, it has to backtrack . but the number of completions of a incomplete plan is exponential in size, so co mputing whether something (the added constraint) is possible is possible or necessary by searching completions is exponential in time.

TWEAK uses a plonomial-time algorithm that computes possible and necessary prope rties of an incomplet plan. Checking truth of a given plan is polynomial in time in th e number of steps of that plan. While generating a plan, inorder to check the corr ectness of the plan we check consistency in terms of ordering and binding constraints.

Notions of consistency :
- in terms of ordering constraints
- in terms of binding constraints

```
e.g. ordering inconsistency : A -> B -> C -> A ( initial A ).
     ( A -> B implies B should come after A ). In the above example, there is
     inconsistency in ordering b'cos A cannot be followed by C.

e.g. Binding inconsistency : X != Y , Y = Z, Z = X
     ( != indicates non codesignation, '=' indicates codesignation )

Checking consistency of ordering and binding is polynomial in time. Computing th
e
transitive closure of a graph with n nodes takes O(n^3) of time. ( n^3 = n*n*n )
.
Checking inconsistency of bindings is nothing but checking transitive closure of
codesignatioan bindings and non codesignation bindings which is also polynomial
in time. It is polynomial because the variables in binding can take infinite val
ues
though they can take only finite values.

Checking CSP ( constraint satisfaction problem ) i.e. binding values to variable
s
is NP hard.

Binding constraints are : Codesignation constraints (say C, where C is a 2 tuple
                             of the form <x , y> where x codesignates with y)
                          Non Codesignation constraints (say NC, and similar to
                            above C but x non codesignates with y)

Inconsistency exists in Binding constraints if a tuple from from C is in NC. Che
cking
this inconsistency takes polynomial time. Underlying assumption is (mentioned ab
ove)
that variables can take infinite values.

Say variables cannot take infinite values,
e.g.  x = { A , B }, y = {A , C}, z = { ...(some values)}
and constraints are     x != B, y != C, z = x, z != y

Then Transitive Closure of C ( Co designation constraints set ) is TC(C) = <z x>
and TC(NC) = { <x B>, <y C>, <z y> }

Since it is not infinite domains, x = A and y = A ==> inconsistency in the const
raints.
But in case of infinite domains for x, y, z, it is not inconsistent.

In finite domains, non codesignation ==> codesignations of variables.
TWEAK Truth Criterion is polynomial because it assumes the domains for variables
are infinite. Infact, TWEAK Truth Criterion doesn't make sense for finite domain
s.

So Truth Criterion may fail bacause checking CSP assumes infinite domains for al
l
variables.

This gives us two different truths:
        (i) Conditional Truth
       (ii) Absolute Truth

Conditional Truth: MTC assumes while working on problem that previous node of cu
rrent
node will be executed without any problems and provides the effects required for
 current
node.

e.g. +p      +p
     ti  -->  s  --> tg
              q      p     ( Here, tg ( goal node ) requires 'p' and s provides
'p'
                             where as s requires 'q' ).
 In this case, by looking at tg and s, it is correct according to MTC, though it
 is
correct locally not globally.

Absolute Truth: Checking global truth i.e. checking MTC at every step of plan gi
ves
rise to correct plans. It is also called Projected Truth because it is projected
 over
a step ( previous ).

Absolute Truth for aboce plan is : []( (q, s) & (p, tg) )
        ( Note : [] means necessarily, & means conjunction )
        []( P & Q ) = []P& []Q
                where P and Q are propositions

No. of maximum preconditions for a plan are = n * e, where n is no. of steps of
plan
and e is maximum no. of precondions ( it is finite foe any given plan ). for act
ion.

All preconditions must be true inorder plan to be correct. i.e. conjunction of
all preconditions should be true. Checking this will take O(n).

TWEAK uses Conditional truth Criterion in generating a plan.

Say P is a partially order partailly instantiated plan : < T,O,B> ( B for Bindin
gs)

Then Pc belongs to completions(P) if the follwoing ocnditions hold:

1) Pc has same actions of P
2) The ordering in Pc is a tatal ordering which is consistent with O of P.
3) The variables in Pc are all bound to constants such that these variable bindi
ngs
   are consisitent with B of P.

Thus, there could be n! for a POPI plan of n tasks. It could be more than n! if
variables bindings are more than one way.

Checking a plan whether it is possibly true by MTC (conditional truth) is also
polynomial.

The MTC for possible case is:
                              A proposition p is possibly true in a situation s
iff two conditions hold: there is a situation t equal or possibly previous to s
in which p is possibly asserted; and for every step C necessarily before s and e
very
proposition q necessarily codesignates with p which C denies, there is a step W
possibly between C and s which asserts r, a proposition such that r and p codesi
gnate
whenver p and q codesignate.

But ()(P & Q) != ()P & ()Q.  So checking possiblity of a plan is exponential.

[Note : () means possibly. It is equivalent to diamond symbol which is used in c
```

lass)

In other terms, making sure a plan is possibly true is checking all possible plans. Picking correct one is exponential in time.

[Note : Checking variables with finite domains and checking conditional affect for tasks is NP hard.]

From elder@enuxhb.eas.asu.edu Fri Feb 12 16:13:08 1993
Return-Path: <elder@enuxhb.eas.asu.edu>
Received: from enuxhb.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA03671; Fri, 12 Feb 93 16:13:08 MST
Received: by enuxhb.eas.asu.edu id AA18326
  (5.65c/IDA-1.4.4 for plan-class@parikalpik.eas.asu.edu); Fri, 12 Feb 1993 17:01:34 -0700
Message-Id: <199302130001.AA18326@enuxhb.eas.asu.edu>
From: elder@enuxhb.eas.asu.edu (Greg Elder)
Date: Fri, 12 Feb 1993 17:01:33 PST
X-Mailer: Mail User's Shell (7.2.1 12/20/90)
To: plan-class@parikalpik.eas.asu.edu
Subject: Notes for 11 Feb 93

Notes for the 11 Feb 93 AI Planning Seminar

Written by Greg Elder

General Topics of Discussion:

1. Recap of Conditional Modal Truth and Modal Truth
2. Efficiency of Partial Ordering Planning (POP) vs Total Ordering Planning (TOP)
3. Systematic Nonlinear Planning (SNP)

1. Difference between Conditional Modal Truth (CMT) and Modal Truth (MT)

   - Both may be necessary and possible, i.e., one can speak of necessary CMT, necessary MT, possible CMT, and possible MT.

   - CMT requires the following:

     (1) Establishment
     (2) Declobbering/non-deletion

   - MT requires the above two conditions plus a third

     (3) Every step which preceeds the step for which you are trying to "make true" must be executable.

2. Efficiency of POP vs TOP

   - TOP commits too fast to an ordering/binding

   - POP puts off ordering/binding until forced to do so. Thus, it will reduce backtracking and be more efficient in terms of planning (not plan execution).

   - Thus, the reason for doing POP is for efficiency in planning, and not for finding the least constrained plan.

   - Each PO plan corresponds to many (exponential amount) of TO plans.

     -- Size of search space for TO plans > search space for PO plans

     -- Per node cost of TO plans and PO plans are both polynomial (O(n) for TO and O(n4) for PO).

   - Keep in mind that size of search space for TO plans being greater than size of search space for PO plans is valid only when the PO plans correspond to a disjoint set of TO plans. This is not necessarily true if they are not disjoint.

   - It is possible for TWEAK to generate plans which have overlapping linearizations.

   - TWEAK's search is not systematic, i.e., it may look at the same node more than once.

   - TWEAK is not guaranteed to have a search space that is smaller than a TOP. Thus, there is no guarantee that it is more efficient.

2. SNP

   - McAllister's SNP does not have complete Modal Truth Criterion (no white knight).

   - Goals which are not supported by a causal link are "open".

   - Causal links act a type of protection. If a causal link exists, then SNP knows that the node has been examined and it will not look at it again (systematic). SNP will not find the same exact PO plan more than once.

   - Constraints are added when dealing with threats to ensure nodes remain valid (causal link valid).

   - SNP uses "truthness" that is sufficient but not necessary. (Don't need Truth Criterion that is necessary and complete for searching POPs.)

```
             +u, -p                +p, q
             ------                ------
   ---- >  | S1 | -------------> | W1 | --------------
   |         ------                ------              |
   |                                 (u)               |
   |                                                   V
 ------                                              ------
 | ti |                                              | tg |
 ------                                              ------
   |                                                 p, q, r
   |         -p, +v                +p, +r
   |         ------                ------              ^
   ---- >  | S2 | -------------> | W2 | --------------
             ------                ------
                                   (v)
```

For the above plan, TWEAK would say that it is good. SNP, on the other hand, would not say it is a good plan because it would see causal links which are threatened.

SNP would be correct for the plans ti -> S1 -> W1 -> S2 -> W2 -> tg and Ti -> S2 -> W2 -> S1 -> W1 -> tg

--
Greg Elder   elder@enuxhb.eas.asu.edu
Department of Computer Science   or   elder@seine.eas.asu.edu
Arizona State University

From gary@enws320.eas.asu.edu Wed Feb 17 15:41:03 1993
Return-Path: <gary@enws320.eas.asu.edu>
Received: from enws320.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA07351; Wed, 17 Feb 93 15:41:03 MST
Received: by enws320.eas.asu.edu (4.1/SMI-4.1)
        id AA02365; Wed, 17 Feb 93 15:29:35 MST
Date: Wed, 17 Feb 93 15:29:35 MST
From: gary@enws320.eas.asu.edu (Kevin Gary)
Message-Id: <9302172229.AA02365@enws320.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

Planning notes for Tuesday 2/16

by Kevin Gary

Agenda:
  1> Systematicity - What does it mean?
  2> Why is SNLP systematic?
  3> What is the relationship between UA and SNLP?
  4> Does systematicity give us anything?
  5> Weld's paper

Review of partially-order planning (POP) versus totally-ordered planning (TOP)

- From last class, we determined that the only justification for looking at POP is to gain efficiency in plan generation. However, there may be other justifications as to why we would want to look at POP. These justifications are motivated from the perspective of "reasoning about plans", and include:

  - plan reuse -> A PO plan corresponds to a set of TO plans, one TO plan for each completion of the PO plan. Hence a PO plan requires less storage (say in a plan library), and is also more easily modified due to a lesser number of constraints than a TO plan.

  - incompleteness -> It may be the case that the agent executing the plan must handle surprise (unexpected) events. The planner may be made to handle these situations through "projection", where the planner simulates future uncertain events in order to reason about what may or may not be true in the future (necessary and possible truth in a projection). A POP handles these situations better since it has more flexibility through its lesser constraints.

  - distributed planning -> POP allows for more open choices in deciding which agents in a multi-agent environment work on what subproblems. It is also easier to merge the resulting plans on the subproblems into a single unified plan.

other peripheral issues/questions

  - justified plans - A plan where no constraints may be removed without losing correctness.

```
        example:   +c         +a         +b
                  ------     ------     ------       ------
                  | Si | ==> | S1 | ==> | S2 |  ===> | Sg | (b)
                  ------     ------     ------       ------
                                         (c)
```

  - Here S1 clearly plays no role in the correctness of the plan, and may be removed.

However, the following problem is more difficult

```
        example:   +c         +a         +c
                  ------     ------     ------       ------
                  | Si | ==> | S1 | ==> | S2 |  ===> | Sg | (c)
                  ------     ------     ------       ------
                              (c)        (a)
```

Here, every step, in itself is doing something useful. However, it is still possible to remove a _group_ of steps and still keep the plan correct. In particular, S1 and S2 can be removed and Si can give c to Sg. Verifying that this plan is not minimal is obviously more complex than in the previous case.

I n general determining a set of constraints to remove from a plan is a complex (NP-hard) problem. Heuristic "hacks" such as using a cost (g) function based on the number of steps in the plan can ensure that shorter plans are considered before longer ones.

  - looping in satisfying goals - "looping on the goal stack"

```
     G
     W
     G ^ W                    G appears on the stack multiple times.
     p2                       It is possible for the same subgoals to
     p1                       appear on the stack multiple times. This
     p1 ^ p2                  may lead to infinite looping.
     G
```

  - STRIPS can be made to find these situations. However, it is not easy to find these situations in POP and still maintain soundness and completeness. This problem is related to plan minimization (open problem).

The idea is that if we know that a plan is non-minimal, then we can prune it and still retain completeness. This ability is

particularly useful when we are doing depth-first search variants.
However, the general plan minimality problem is NP-hard [Fink & Yang]

SNLP and UA

- UA claims that the search space it explores would never be larger than TO, and in fact you expect it to be much smaller since each PO plan in UA's search space corresponds to a disjoint set of TO plans.

systematicity and search

1> None of the plans you look at in PO space have the same linearizations
2> Search never visits the same node twice.

- <1> is a redundancy criterion; even if you consider each plan in PO space only once (as in <2>), you may still consider PO plans that share linearizations (overlapping sets of corrresponding TO plans). This criterion requires that the PO plans you look at correspond to disjoint sets (forming an equivalence class) of TO plans.
- <2> is systematicity. It says we won't look at the same node in the (PO) search space more than once, but it doesn't say anything about the redundancy criterion in <1>.

- UA doesn't guarantee systematicity, which means it can be possible to do an infinite amount of search in its finite space. UA does guarantee that the plans in its PO space satisfy <1>.
- To get systematicity, a bookeeping measure of some sort is required to check to see that you haven't considered the same node before. Using a closed list is impractical since you can't control the size of the closed list through the search strategy - in the worst case you would have to store every node in the space. The second problem here is that checking for a duplicate plan is itself an NP-hard problem for PO plans (checking isomorphism of two transitive closure graphs).

- If you could incorporate a closed list efficiently, then UA could obtain <1> and <2> and clearly be superior to SNLP.
- SNLP guarantees systematicity through its causal link representation and algorithmic approach (as opposed to using a closed list). See the proof of systematicity in McAllester's paper.

Planning in UA

- UA also tries to satisfy open preconditions.

interaction - If one step has an effect that is needed (is a precondition) of another step, then those two steps are said to _interact_.
- This is the notion of interaction that UA uses. It is a _Lifschitz_ completeness criteria - still guarantees completeness even though it may be more restricted than needed

```
example:     +q                +p
          ------            ------
          | S1 | ==========>  | S2 | ===\        ------
          ------            ------     \       | p  |
             \        +r+q     (q)      |===> |    |
              \      ------            /       | r  |
               ====>  | S3 | ===========/        ------
                     ------
```

- Here, S3 interacts with S2, so UA wouldn't say this is complete, although TWEAK's MTC would say its correct.

- UA's search is unambiguous, since any precondition in UA is assumed to be necessarily true or false when considered.
- UA guarantees there are no overlapping linearizations in TO space.
- The representation in UA only allows something to be deleted if it is in the precondition list (no loss of generality.
- UA looks at putting a step with +P before -P at the same time as looking at putting -P before +P, in order to ensure it won't return to the choice again (even though the former option won't give a correct plan).

For next class:
- Can SNLP still come to different PO plans that share linearizations?
- Find an example using the UA search strategy where you come to the same plan more than once.

- So UA adds a constraint on all steps that interact

From rao Fri Feb 19 11:35:15 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA08364; Fri, 19 Feb 93 11:35:15 MST
Date: Fri, 19 Feb 93 11:35:15 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9302191835.AA08364@parikalpik.eas.asu.edu>
To: plan-class
Subject: new papers added
Reply-To: rao@asuvax.asu.edu

The following papers have been added to the readings file in library (the one in ailab was missing-- so the papers are kept on the shelf. Please add them to the file, and leave the file on the shelf to the extent possible).

1. ADL action representation language

2. Planning with actions with conditional effects (pednault)--total ordering planning

3. Nonlinear planning with ADL operators

You should read these three, followed by UCPOP paper in that order.

Rao

From ihrig@enws318.eas.asu.edu Fri Feb 19 17:57:39 1993
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA08692; Fri, 19 Feb 93 17:57:39 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA04139; Fri, 19 Feb 93 17:49:41 MST
Date: Fri, 19 Feb 93 17:49:41 MST
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
Message-Id: <9302200049.AA04139@enws318.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

Feb 18, 1993   Notes by Laurie H. Ihrig

Agenda:
1. Can we show that UA has redundancy?
2. Comments re SNLP
3. Barrett and Weld on which planner is better

Definition of Redundancy: Same plan in two different paths in search space.
Definition of Equal Plans:
1. Plans have same steps.
2. Plans have same causal links.
3. Plan has same safety conditions (how threat resolved)

example:

```
S1                        S2                       S3
puton(A,B)    ---->       puton(A,C)     ---->     puton(A,B)

S2                        S4'                      S5'
puton(A,B)    ---->       puton(A,C)     ---->     puton(A,B)
```

These two plans are equal since there is a one to one mapping of steps (although step names don't match)

Suresh has discovered that the search tree of UA has the property that if you look at a single level, every node on that level will have the same number of steps. This is because UA never does simple establishment (using step already in plan). Also, threats are resolved right away. This means that if there is redundancy in the search space, then the redundant plans must be at some level i.

Example:

```
             S1        ------>       S2
            (P)                      (Q)
```

Suppose that another step S3 is added to above plan:

```
                       -P +Q
                         S3
```

```
UA will order steps as either     S3 --->S1--->S2
                           or     S1 --->S3--->S2
```

SNLP, on the other hand, does not resolve the threat right away. The claim by Minton that SNLP plans in a space of unambiguous (every precondition is either necessarily true or necessarily false) plans is not true. When a causal link is added, threats that result are not resolved immediately. It is possible to fold threat resolution into causal link establishment, so that whenever a causal link is added, all threats with respect to that causal link are resolved.

Systematicity means that no two complete nonlinear plans with different paths to the root node will be equal. See Harvey, Ginsberg, and Smith "Deferring Conflict Resolution Retains Systematicity" for a proof that SNLP is systematic.

Redundancy in the search space means that previous work may be repeated. We can avoid this by using a closed list, but it is hard to determine equality of nodes (plans).

Causal links involves the concept of a protection interval which is an old idea. They provide a record of which establisher establishes which precondition. Everything done after respects the previous protection intervals. The search space is pruned on that basis. It may be that the solution node is one step away, but since you always respect establishment, the branch fails. This could be problematic. SNLP, by bookkeeping, avoids redundancy, but at the expense of committment, this time a different type of committment--an arbitrary step is assigned to be the contributor.

Example:

```
  Suppose we have a goal:  on(A,B)
                           on(B,C)
                           handempty
   and initial situation:  on(A, Table)
                           etc.
```

and suppose that handempty is picked as the first goal to solve and the start step is picked as the establisher of handempty.

Initially there is a node on the open list:

```
                     ti  ---->  tg
```

There are two steps in this initial plan.

The children of this node are:

1. ti -----> tg with ti the establisher of handempty.
   This plan has two steps.

2. ti -----> stack(x,y) ---->tg with stack(x,y) the establisher of handempty. This plan has three steps

3. ti -----> putdown(x) ---->tg with putdown(x) the establisher of handempty. This plan has three steps.

At this point we have no idea what eventually will be the establisher of handempty. It is a high frequency condition (Most steps have it as an effect). Committing early to an establisher will not mean that you no longer have completeness, but it may mean that you waste time. If you allow retraction of the choice of contributor then you may have redundancy.

Committment 1 will never be valid, but other choices may be premature as well. Abstraction would allow you to know that handempty should be worked on last. We can add abstraction to a planner to increase efficiency.

Gopi's example of redundancy in UA:

```
       operators:                         goal:

               O1[+g1,  +x]                  g1, g2, g3

       (x,g1)  O2[+g2, -g1]

               O3[+g3, +g1]


              <g1,g2,g3>
```

```
              /         \
         g1 by O1      g1 by O3
           /              \
        O1                 O3
        |                   |
        |                   |
    g2 by O2             g2 by O2
      /   \               /   \
  O1 O2    O2 O1       O3 O2  O2 O3
    |                          |
    |                          |
 g3 by O3                   x by O1
                               |
    /                          |
O1 O2 O3                    O1 O2 O3
```

Laurie adds another example:

```
     operators:                        goal:

          O1[+g1, +g3]                      g1, g2, g3

          O2[+g1, +g2]


             <g1, g2, g3>
               /       \
         g1 by O1      g1 by O2
           /              \
          O1               O2
          |                |
          |                |
         g2 by O2      g3 by O1
          |                |
          O1               O1
         < >              < >
          O2               O2
```

In SNLP which keeps track of causal links these plans are not equivalent.

Efficiency is not guaranteed by a small search space size. For example, overcommittment might increase inefficiency despite a small search space.

Why does SNLP care about a positive threat?
see paper by Harvey et al.

If you are interested in eliminating redundancy you have to decide who is the establisher of a precondition and stick to it.
In other words, search is in the space of equivalence classes defined on the space of totally ordered plans. Every totally ordered plan should correspond to only one partially ordered plan.
You should be able to compute the partially ordered plan from the totally ordered plan, that is, given a totally ordered, get a partially ordered, and only one such plan.

Example:
Given a plan

```
+P +Q                  +P
  S1    ------->       S2   ----------->  S3
                                        (P,Q)
```

What is the partially ordered plan that defines the equivalence class for this totally ordered plan.

We must get the steps, the causal links and the safety constraints.

P = < S,C, V>

1. The steps are the same as in the totally ordered plan.
2. Causal links: The unique last step that gives a condition is the establisher

```
              P                          Q
   eg.    S2 ---->S3               S1 ----->S3
```

   The number of causal links is equal to the number of preconditions.

3. Safety conditions: We have to consider all threats as defined by SNLP and then the way each is resolved is clarified by looking at the plan. eg. S1 s a threat to S2 --> S3. If it was resolved by putting S1 after S3, then this is not consistent with the order in the plan. Note: you must also consider positive threats. Otherwise the same partially ordered plan corresponds to more than one totally ordered plan.

Practical way of looking at the efficiency of Partial Ordering vs Total Ordering Planners:

Bottom line is: Even if guaranteed that search space is smaller, not guaranteed that in average case the planner will do better. Increased redundancy in search space doesn't necessarily mean a loss of efficiency eg Macroperators increase redundancy but improve performance for the average case, although not for the worst case.

Barrett and Weld (see readings) do empirical studies. The way to do this is to start with a hypotheses eg. you could have the hypothesis that every domain with the stack action causes total ordering planners to be more efficient than partial ordering planners.
It is hard to find a large scale domain in which one can easily test a hypothesis. Therefore we must look at artificial domains.

Barrett and Weld look at relative efficiency of planners for
1. independent subgoals
2. serializable subgoals
3. nonserializable subgoals

If serializability was independent of planning methodology then you wouldn't expect partial ordering planners to be better. But this is not the case. Barret and Weld change the notion of what it means to be serializable. Some domains are serializable only for partial ordering planners.

STRIPS had serialzable subgoals if it could work on goals in order without undoing previous goals. STRIPS merges plans for goals by concatenating them. Partial ordering plans merge plans for goals by interleaving them. There are more domains in the world where interleaving will work than domains where concatenating will work.

```
Therefore, there are more domains that are serializable for
partial ordering planners.

  If there are n goals, then there are n! orders of these goals.
If one order is serializable then the problem is serializable.
If .999 of these orders are serializable then the problem is
trivially serializable.  If .001 of these orders are
serializable, then the problem is laboriously serializable.
Some domains are trivially serializable for SNLP but
laboriously serializable for other planners (Waldinger's).
Barrett and Weld compare
                TOPI (like STRIPS)
                TOCL (like Waldinger's)
                POCL (like SNLP)
by generating problems randomly and checking how each planner does.
This is a practical way of looking at the efficiency of partial
ordering vs total ordering planners.



From rao  Sun Feb 21 19:41:59 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11565; Sun, 21 Feb 93 19:41:59 MST
Date: Sun, 21 Feb 93 19:41:59 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9302220241.AA11565@parikalpik.eas.asu.edu>
To: plan-class
Subject: a new puzzler...
Reply-To: rao@asuvax.asu.edu

Buoyed by the success of the UA counter example puzzler, I decided to
throw another one (which is part easy, and part open)

Couple of classes back (the one in which Kevin took notes), we
discussed the notion of minimal or justified plans and their relation
to looping control.

In particular, I suggested that if a certain (partial) plan is known
to be non-minimal (in that a subset of the actions in that plan will
be able to achieve all the goals achieved by the original
(partial)plan), then we can prune it during search, and thereby
improve efficiency.

Consider SNLP and this pruning strategy. Forget about the cost of
checking justifiedness (as I said, perfect justification, or
recognizing _every_ non-minimal plan as a non-minimal plan, is
NP-hard. However, we can use some greedy algorithms that are sound in
that if they say that a plan is non-minimal, it is guaranteed to be,
but are incomplete in that some plans endorsed as minimal plans by
this procedure may in fact be non-minimal).

Your task is to tell me

1. Is such pruning strategy is a complete or admissible heuristic for
   SNLP in that if SNLP can find a plan for a problem without it, it
   will still find a plan for that problem with the heuristic. (Either
   prove that it is complete, or give a counter example showing it is
   not complete.)

2.  If you proved that such a pruning technique will not be complete
    for SNLP then

    2.1. What possible change to SNLP will make the technique
    complete?

    2.2. Can you think of some other restrictive pruning criterion
    which will keep completeness?


As always, I know the answer to part of the puzzler, but am hoping
that you will come up with answers/ideas also for the parts that I
don't know!


Group thinking is fine... No grading... Just a puzzler, in the cartalk
style ;-) Answers requested by Tuesday's class.


From rao  Mon Feb 22 16:39:33 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA01183; Mon, 22 Feb 93 16:39:33 MST
Date: Mon, 22 Feb 93 16:39:33 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9302222339.AA01183@parikalpik.eas.asu.edu>
To: plan-class
Cc: rao
Subject: clarifications re the puzzler
Reply-To: rao@asuvax.asu.edu


1. When I say "prune a non-minimal plan from the search space", what I
   mean is that whenever I pick a  plan from open list for expansion,
   I will check if its is non-minimal, and if it is, will not expand it.
   (this is equivalent to assuming that this node has no children).

2. If you happen to comeup with a counter example which shows that
   this heuristic will be incomplete for SNLP, then check to see if
   that example will also show incompleteness in UA.

Rao

From ihrig@enws318.eas.asu.edu Mon Feb 22 15:45:22 1993
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 "^From:" nil nil nil])
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA01107; Mon, 22 Feb 93 15:45:21 MST
Received: from enws322.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA07133; Mon, 22 Feb 93 15:37:18 MST
Message-Id: <9302222237.AA07133@enws318.eas.asu.edu>
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
To: rao@enws318.eas.asu.edu
Date: Mon, 22 Feb 93 15:37:18 MST

Puzzler Answer                                            Laurie H. Ihrig

If a certain (partial) plan is known
```

```
to be non-minimal (in that a subset of the actions in that plan will
be able to achieve all the goals achieved by the original
(partial)plan), then we can prune it during search, and thereby
improve efficiency.

Consider SNLP and this pruning strategy. Forget about the cost of
checking justifiedness (as I said, perfect justification, or
recognizing _every_ non-minimal plan as a non-minimal plan, is
NP-hard. However, we can use some greedy algorithms that are sound in
that if they say that a plan is non-minimal, it is guaranteed to be,
but are incomplete in that some plans endorsed as minimal plans by
this procedure may in fact be non-minimal).

Your task is to tell me

1. Is such pruning strategy is a complete or admissible heuristic for
   SNLP in that if SNLP can find a plan for a problem without it, it
   will still find a plan for that problem with the heuristic. (Either
   prove that it is complete, or give a counter example showing it is
   not complete.)

Counter example:

     operators:          O1 [+g1]

                  g3   O2 [+g1, +g2, -g3]

                  g1   O3 [-g1, +g3]

     initial state:  g3

     goal:         <g1, g2, g3>              plan: O2 --> O3 --> O1


                        <g1,g2,g3>

                        /         \
              g1 by O1             g1 by O2
                      /             \
                     /               \

                -O1-                   -O2-

                  |                      |

              g2 by O2             g2 by O2

                  |                      |
                  |                      |

                  O1                     -O2-
                 <  >
                  O2

                  |                      |
                  |                      |
              prune O1                   |
                  |                      |
                  |                      |
                 -O2-                    |
                  |                      |

                  |                      |

               g3 by O3             g3 by O3
                  |                      |
                  |                      |

                 O2                     O2
                <  >                   <  >
                O3                      O3
               fail                    fail
                (no way of resolving threats)

2.  If you proved that such a pruning technique will not be complete
    for SNLP then

    2.1. What possible change to SNLP will make the technique
    complete?  adding white knight capability

                                                     Laurie
```

```
From rao  Tue Feb 23 19:13:56 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA02305; Tue, 23 Feb 93 19:13:56 MST
Date: Tue, 23 Feb 93 19:13:56 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9302240213.AA02305@parikalpik.eas.asu.edu>
To: plan-class
Subject: Next class
Reply-To: rao@asuvax.asu.edu
```

Next class, we will discuss UCPOP which allows extending the language
to have actions with conditional effects. The minimal reading
requiremnts are UCPOP, and any one of the Pednault's papers.

Rao

```
From AZEDC@ACVAX.INRE.ASU.EDU  Thu Feb 25 00:15:30 1993
Return-Path: <AZEDC@ACVAX.INRE.ASU.EDU>
Received: from ACVAX.INRE.ASU.EDU ([129.219.10.1]) by parikalpik.eas.asu.edu (4.
1/SMI-4.1)
        id AA03093; Thu, 25 Feb 93 00:15:30 MST
Received: from ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF #2382 ) id
 <01GV4ABHMPSI009UTC@ACVAX.INRE.ASU.EDU>; Thu, 25 Feb 1993 00:07:30 MST
Date: 25 Feb 1993 00:07:30 -0700 (MST)
From: AZEDC@ACVAX.INRE.ASU.EDU
Subject: class notes
To: plan-class@enws228.eas.asu.edu
Message-Id: <01GV4ABHMZFO009UTC@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"plan-class@enws228.eas.asu.edu"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT
```

Notes for the class of feb 23
by Eric Cohen

Agenda: puzzler
          Weld et al

1. Puzzler discussion

We discussed a few classes back the notion of minimality of plans and some mechanism for avoiding loops. Given a plan P=<S,O,Pi>, if a subplan of this plan can solve the problem, for instance by removing a step si from S, then P is non-minimal.

```
    +he    +hf
... s1     s2    (goal state)
    (hf)   (he)  (hf)
```

In the example above, it is obviously non-sense to keep adding steps s1 and s2.

Note that in our formulation of planning, there is nothing that specifically prevents this from happening, unless for instance an admissible strategy such as BFS or A* is used.

The idea of this puzzler is to come up with some mechanism that would detect this problem, and in such cases prune the search tree so that non-minimal plans not be explored, thereby increasing efficiency. Here, to prune means we will remove the node from the "open list", and therefore it will never be considered for expansion again.

The question is, is such a pruning strategy complete or admissible for planners such as SNLP ? By complete it is meant that if the planner can find a plan for that problem without this pruning, it will also be able to find it.

In this discussion, we are not concerned about the computational effort required to verify the minimality (or justifiedness) of the plan.

The answer to this problem requires a discussion on what is meant by minimality. Using the example above, if there is no other goal than hf, then obviously the plan is non-optimal; on the other hand, if there are other goals, there may be other refinements which will make this plan optimal with respect to its goals (so it is not minimal in the "total" sense, even though it was so in the "local" sense). To illustrate, consider the example:

```
                    A
      A  B          B
     ------      -------
                   (he)

                   (on A,B)
```

Suppose we pick goal he first. There are three possible plans giving he:

```
I -> nothing     -> G
I -> putdown (x) -> G
I -> stack (x,y) -> G      <- According to our definition,
                              this is not the minimal plan (it has 3 rather
                               than 2 steps).
```

However, looking at the global plan below, we notice that this does not mean the plan cannot be refined further:

I -> pickup (A) -> stack (A,B) -> G

Also notice that there are no other minimal plans that satisfy both goal preconditions; therefore, if we had pruned this node, by removing it from the search space there is loss of completeness.

To illustrate how SNLP will solve this problem:

```
                               I -> G   (he) (on A,B)
```

SNLP will choose either goal precondition; there are two possible causal links:

```
+on(A,B)            +he
S -> G       and    I -> G


     I -> stack(A,B) -> G

     here, since stack(A,B)
     represents a positive  threat to goal precondition  he, SNLP will
     try to put it either before or  after the causal link; since this
     is not possible, the branch will be abandoned.
```

But, without positive threats, we will have on this branch:

```
     -he
I -> pickup(A) -> stack(A,B) -> G
```

SNLP will try to protect the causal link I->G ; it does not allow change of contributors in the same branch, which means that if we prune other branches that don't look so promising, a branch which would contain a correct plan may not be visited.

Another question was whether other planners (such as UA) would suffer from the same loss of completeness. Such planners allow for contributors to be shifted in the same branch, and as a result non-systematic planners such as UA don't lose completeness when pruning is used.

The next question is whether SNLP can be changed, and if so how, in order to maintain completeness using this pruning technique.

Going back to Tweak, we remember that the White Knight criteria allowed for conditions which had been clobbered to be reasserted. Thus, to obtain completeness, we need to introduce the WK criteria; however, if WK criteria is used, it will have to be used across the board, and not only during the pruning part. Nevertheless, when we went from Tweak to SNLP, one of the fundamental differences between them was that SNLP did not have WK, so we would be changing the "essence" of the planner.

Now suppose we have some sort of minimality (or justifiedness) criteria, which is sound, in the sense that if it decides that the plan is non-minimal, it is guaranteed to be so; when it says that a plan is minimal, it may be the case that it is not so. Checking for this necessary condition is polynomial; efficiency may be reduced when the criteria decides incorrectly that the plan was optimal. Perfect justification is NP-hard.

The next question is, is there a planner that is complete using the pruning strategy ?

We notice that, if a problem has a solution, then it also has a minimal solution (say, the minimum number of steps).

We need to prove that there exists a sequence of refinements (bindings, steps and orderings), going from an initial to a goal state. The proof is by induction (proof not shown).

Thus, it is perfectly possible that we will work on the same goal more than once, which could be ok as in the following example:

```
 I -> pickup(A) -> stack(A,B) -> G
                                (he)
```

But here, in a Tweak fashion, we will have to declobber:

```
     C
             +hf
             --->
   A  B                on(A,B)
  ------

  pickup(A) -> stack(A,B)

  since stack(A,B) deletes hf, we'll have to add a step that
  declobbers it.
```

In STRIPS, it was possible to check minimality by verifying whether a given condition occurs more than once; in that case, there is a loop. However, the analogy to POPI plans doesn't carry:

```
                    +he
 I -> pickup(A) -> stack(A,B) -> G
                                (he)
```

The next topic was the discussion on whether there exists some other restrictive pruning criterion that will keep completeness: paper "Temporal Coherence", from Drummond, Curry - 1989 IJCAI/ 1989 Computational Intelligence

This paper discusses pruning criteria for POPI plans. The proof needs to show that there is no loss of completeness.

Given a partial order plan [not containing the initial step], at a certain point in time some conditions are supported and others aren't.

We take all the outstanding goals, and the unsatisfied preconditions of the existing steps (called Bulk Preconditions). To illustrate, consider the 3 blocks world problem:

```
                      A
                      B
      A B C           C                  (on(A,B)) (on(B,C))
```

```
Suppose we work on on(B,C) first:          stack(B,C)
bulk preconditions:
(outstanding goal)                         on(A,B)      (*)
(preconditions of stack)                   clear(B)     (*)
                                           clear(C)
                                           handempty
```

However, two of the bulk preconditions are inconsistent (*) and the branch is therefore pruned. This can be checked by a statement such as
 for all x,y, on(x,y) => not (clear(y))
(of course, a statement like that is not used by the planner because we want to avoid ramification; it is only used to check inconsistency, which tells us whether we should prune or not)

Discussion of the completeness

Using SNLP, the plan stack(A,B) -> stack(B,C) (which is temporally consistent) is not complete.

For Tweak where we don't backtrack the goal orders, it is not complete. Illustration:

```
                       A
     A                 B
     B C               C
    -----            -----
```

Tweak works on unsatisfied conditions, so it picks on(B,C).
bulk ...... stack(B,C), clear(B), clear(C)
However, on(A,B) now is no longer true, so it needs to put on(A,B) back. Since on(A,B) and clear(B) are temporally inconsistent, the branch will be pruned, and since there is no other node in the queue giving the solution, we lost completeness.

As in Waldinger's planner, Tweak would have to do a step addition of stack(A,B) to maintain completeness.

Problems with the temporal coherence:
. it increases redundancy much more than it reduces (prunes) branches
. with a complete Tweak, the time improvement is not significant; gains in the pruning are offset by the time to set up the strategy

An important point to make is that we are still forcing the planner to look at the conditions in exactly the reverse order of the sub-goals (execution is tied up to planning order).

2. Weld et al paper

All planners can be thought of doing some sort of merging of sub-plans:
 STRIPS concatenates (adds) sub-plans
 POPI interleave sub-plans

Given sub-goals G1 = plan (P11 P12 P13) and G2 = plan (P21 P22 P32),

sub-goals are serializable if there exists a P1i and P2j such that the combined method P1 -> P2 or P2 -> P1 is a correct plan

Concatenation is a special case of interleaving:

```
     a - b
   /       \
   \       /
     c - d
```

in STRIPS, we have a-b-c-d or c-d-a-b, and in POPI we have a-c-b-d, a-b-c-d, c-d-a-b, ...

Degrees of serializability
The efficiency of a planner is closely related to the degree of serializability:
independent > trivially s. > laboriously s. > non-serializable

Weld et al. use artificial domains such as D S2:
To make Gi true, we need to make A1i -> A2i (setup, then make condition true)
To make G1G2, A11 -> A12 -> A21 -> A22

STRIPS will solve the latter as A11 -> A21 -> A12 -> A22, which is not a correct plan.

So, this domain is serializable for TOCL/POCL (it is respectively laboriously and trivially serializable).
The domain is non-serializable for TOPI.

From gopi@enuxha.eas.asu.edu Mon Mar 1 15:50:22 1993
Return-Path: <gopi@enuxha.eas.asu.edu>
Received: from enuxha.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA07853; Mon, 1 Mar 93 15:50:22 MST
Received: by enuxha.eas.asu.edu id AA21317
(5.65c/IDA-1.4.4 for plan-class@parikalpik.eas.asu.edu); Mon, 1 Mar 1993 15:47:49 -0700
From: Bulusu Gopi Kumar <gopi@enuxha.eas.asu.edu>
Message-Id: <199303012247.AA21317@enuxha.eas.asu.edu>
Subject: Notes for the Feb 25 class (fwd)
To: plan-class@parikalpik.eas.asu.edu
Date: Mon, 1 Mar 93 15:47:46 MST
X-Mailer: ELM [version 2.3 PL11]

Thursday, 25 Feb 1993

AGENDA

Conditional Effects
UCPOP
ADL

Conditional Effects

The traditional STRIPS representation can be extended by adding the following additional features

* Disjunctive preconditions

  of the form : (p1 V o2)

* Universal quantification

  of the form : For all X P3(X)

* Conditional effects

  of the form : if (p1(x)) then p2(y)

Of these, the simplest extension is conditional effects.

An example

```
operator : puton (x,y,z)
Preconditions : clear(x), clear(z)
Add :   on (x, z)
        if (isblock (y)) then clear(y)
        if (~isblock(z)) ~clear(z)
        ~on (x,y)
```

The above example shows how a blocks world operator which can handle both blocks and table can be defined using conditional effects.

How to extend an existing planner to handle conditional effects ?

A normal existing planner can be easily extended to handle conditional effects in the operators.

Handling conditional effects in STRIPS

Conditional effects can be easily handled in STRIPS. This is because the state before a step is completely defined in a total order planner. This means that STRIPS has to make sure in addition to the preconditions of the step, all the conditions necessary for the required conditional effect to be given by the step are also made true before the step. This implies that a STRIPS styled planner has to add in addition to the preconditions, the conditions associated with the desired effect, on the goal stack.

Handling conditional effects in a Partial Order Planner

Handling conditional effects in a partial order planner like TWEAK is more complicated. The basic difference in handling operators having conditional effects is in the computation of MTC. The cost of checking for necessary truth of a goal or precondition in the absence of conditional effects is polynomial. With conditional effects, this cost becomes exponential. Intuitively, to check necessary truth all possible completions of the plan have to be checked when operators have conditional effects, the cost of which is exponential ! (This was discussed in detail in a previous class on TWEAK)

In short conditional effects result in the cost of checking for MTC increase to exponential from polynomial. The question that arises is, is it worth considering conditional effects at all, if even the basic step of the TWEAK planning process takes exponential time. (The overall planning takes exponential time in any case)

Overall complexity of the planning process with conditional effects
----

A single operator with conditional effects can always be broken down into an exponential number of simpler operators without conditional effects. Thus given a planning domain, the number of operators when conditional effects are allowed will be less than the number of operators when the conditional effects are not allowed! This implies that the branching factor in the search space decreases when conditional effects are allowed. As already discussed this will offset to a very large extent any increase in the per step complexity. To summarize, conditional effects shift complexity from branching factor to per step complexity. Further, planning is undecidable with or without conditional effects. All these facts imply that making operators more expressive and powerful will not in anyway increase the overall complexity of the planning process.

Does TWEAK use conditional effects

Quite interestingly, TWEAK uses conditional effects that arise due to partial binding. In this context, it should be noted that there are two ways in which the process of checking for a correct plan can become NP-Hard :

1) If the variables have finite domain
2) Ordering is not total order, if then type of condition effects allowed

What needs to be done to SNLP to handle conditional effects

Consider the following partial plan

```
    if Q then P
    if T then R
    +-------+                                  +---------+
    |  1    |                                  |         |
    | S     |                                  | S       |
    |       |                                  |         |
    +-------+                                  +---------+
    (SP1, SP2)                                    (P)


                     If C then ~P
                     +-------+
                     |   11  |
                     |  S    |
                     |       |
                     +-------+
```

If SNLP selects S1 to contribute P to S, in addition to adding a link from S1 to S, all preconditions of S1 (SP1, SP2) are added as open conditions. In addition since a conditional effect (P) is being used a subgoal (Q) to provide that effect should be added. In addition any step which is a potential threat (S11) should be dealt with in the following ways

* Promote (Make S11 come after S)

* Demote (Make S1 come after S11E)

* Confront, which can be done by

  ** Separation

  ** Make sure ~C is necessarily true before S11

From rao Thu Mar 4 12:09:36 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA10745; Thu, 4 Mar 93 12:09:36 MST
Date: Thu, 4 Mar 93 12:09:36 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303041909.AA10745@parikalpik.eas.asu.edu>
To: plan-class
Subject: Dynamic Logics--some notes
Reply-To: rao@asuvax.asu.edu

Folks--

In the last class, I discussed Dynamic Logic as a possible vehicle for modeling plans and planning. Here are some notes on Dynamic logic, that you might find useful.

Rao
[Mar 4, 1993]

Extending STRIPS action representation: Dynamic Logic.

Reading: Rosenchein’s paper in Readings in Planning Book
         Also, Henry Kautz’s M.S. thesis

Dynamic logic is a variant of modal logic which has been used to provide semantics to programs. Here the idea is to talk not only about the truth of propositions like p, but also truth of propositions after execution of certain program expressions.

For example if h is a program expression (say, an action, a conditional, an iterated action etc), and p is a propositional formula, then the following are all dynamic logic wffs:

p   [true if p is true in current world]
[h]p [intuitively, true if p is true in every world resulting from execution of h in the current world. corresponds to nectruth

<h>p [intuitively true if p is true in _some_ world resulting from execution of h from the current world. corresponds to posstruth]

Program expressions are built from simple actions and their compositions.

Simple actions are modeled in terms of dynamic logic axioms such as (where a is an action and q and r are arbitrary propositional logic formulas)

q => [a]r

which reads, if q is true in the current world then r will be true in every world resulting from the execution of h in the current world
and

q => <a>r

(which means that if q is true in the current world then r will be true in some world resulting from the execution of a in the current world)

**It is important to note that q and r in the above formulae can be _arbitrary_ propositional formulae, including those that involve disjunction:

eg: [P & Q] => [a] [R V W] (where V is disjunction and & is conjunction operator respectively]

Apart from modeling simple actions, dynamic logics allow a rich language for composing the actions into programs. The program constructs allowed in dynamic logic are identity actions (^) which maps the current world to the same world, simple actions, sequencing of actions (a1;a2), disjunction or union of actions (a1 V a2), iteratio actions (eg h*, where h is executed some arbitrary number of times) and conditionals like p?. The construct p? maps the current world to itself if the propositional formula p is true in the current world. If not, it maps the current world to an empty set (i.e, the plan diverges). (See Rosenchein's paper in Readings in Planning).

The semantics for dynamic logics are provided through Kripke structures or possible worlds (which are originally invented for modal logics).

In particular, program expressions like h, which can be seen as composite actions in planning terms are interpreted as mappings from worlds (states) to se
ts
of worlds (states).

Note that this is a marked departure from STRIPS where we have the semantics of an action defined in terms of Syntactic modifications to states.

Thinking of actions as mappings from states to states allows us to model not only simple STRIPS type actions, but also more complex actions such as conditional actions (where mapping depends upon certains things being true or false in the given state), iterated actions and non-deterministic actions.

In dynamic logic formulation, a STRIPS planning problem can be stated as:

Given E which is a specification of the formulae in the dynamic logic, and a goal forumula G, a solution to the dynamic logic programming problem is a program expression e such that:

1. e halts when executed from the current world
2. [e]G is true

Note that halting or termination is separated from goal establishment. A program that does not halt (ie., it maps current state to an empty set, in the kripke semantics), is called a divergent program. If e satisfies both 1 and 2 then it is called a totally correct solution.

First order dynamic logic can be defined on top of FOPC the same way propositional dynamic logic is defined on top of propositional logic

>From the discussion above, it should be clear that inference in dynamic logic is strict superset of inference in the corresponding non-dynamic logic. In particular, in Propositional dynamic logic, not only do we ask queries of type is p true? but also queries of type is <h>p true? is [h]p true?

Inference in propositional dynamic logic can be shown to be Exptime-Complete [McAllester], while inference in propositional logic is NP-hard. In the case of first-order dynamic logics, inference is semi-decidable (just as it is in the case of first order logic).

A more relevant question is how costly is it to reason about the effects of actions in dynamic logic. We will see that the notion of regression of a formula over an action plays an important part in reasoning about the effects of actions. Simply put, regression of a formula phi over an action a

(typically written as a^{-1}(phi))

is just the weakest conditions that need to be true before the execution of a such that phi will be true after the execution of a.

thus a^{-1}(phi) <=> [a]phi

Turns out that computing regression of formulas over actions axiomatized in dynamic logic, is NP-complete in the case of propositional dynamic logic, and it is only partially decidable in the case of first order dynamic logic. Part of the reason for this is that these languages allow disjunctive effects for actions.

i.e, we can write formulas such as [h](p1 V p2) in PDL. One of the ways Pednault's ADL avoids this complexity is to insist that there be no disjunctive effects (this is one of the things Pednault means by saying "actions must be completely specified". Note that ADL does allow initial state to be incompletely specified). This allows ADL to ensure that regression can be computed in polynomial time for all the actions.

Although, one could have used the same restriction for dynamic logics too, in practice, people feel that Dynamic logic is some what awkward/unnatural as a basis for plan generation. This is the reason why ADL, which sticks to STRIPS representation, is preferred more.

Rao
[Mar 4, 1993]

From rao Thu Mar 4 12:11:20 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA10752; Thu, 4 Mar 93 12:11:20 MST
Date: Thu, 4 Mar 93 12:11:20 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303041911.AA10752@parikalpik.eas.asu.edu>
To: plan-class
Subject: mini-puzzler: Convert briefcase problem to STRIPS representation
Reply-To: rao@asuvax.asu.edu

As I suggested in last class, I would like y'all to try and convert briefcase problem in Pednault's CIJ paper to pure strips representation (i.e., without any conditional or quantified effects). One of you can then explain the solution in the class.

Rao
[Mar 4, 1993]

From annaredd@enuxha.eas.asu.edu Thu Mar 4 15:24:45 1993
Return-Path: <annaredd@enuxha.eas.asu.edu>
Received: from enuxha.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA11033; Thu, 4 Mar 93 15:24:45 MST
Received: by enuxha.eas.asu.edu id AA10156
(5.65c/IDA-1.4.4 for plan-class@parikalpik.eas.asu.edu); Thu, 4 Mar 1993 15:24:57 -0700
Date: Thu, 4 Mar 1993 15:24:57 -0700
From: "Francis V. Annareddy" <annaredd@enuxha.eas.asu.edu>
Message-Id: <199303042224.AA10156@enuxha.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

Notes for the March 2nd class:

Agenda:

Syntactic change to STRIPS algorithm to add conditional effects.

Contents:

1.0 Conditional effects revisited:

1.1 For establishment:

```
      If a then P
        ---                            ---
       | S1| ----------------------->  | S2|
        ---                            ---
                                       (P)
                   If Q then ~P
                        ---
                       | S3|
                        ---
                   FIG 1.
                   P
```

In FIG 1, S1 ----> S2 is the causal link. If this causal link has to materialize, the condition (a,S1) is to be added over and above the preconditions of S1. If S1 does not have any other preconditions then a is the only precondition.

1.2 Clobbering:

For the precondition P to S2 in FIG 1, threats are not only those which definitely delete P but those which conditionally delete P. An example of the conditional effect "If Q then ~P":
Q : If a block is free
P : Don't put B on the block

So, the conditional effect is "If a block is free, put B on the block".

Four ways of preventing clobbering are:

(a) promotion: In FIG 1, S3 could be placed after S2.
(b) Demotion: In FIG 1, S3 could be placed before S1.
and
(c) confrontation: In FIG 1, ~Q could be made true.
(d) Separation:

```
            P(A,B)
        S1 -------> S2

           ~P(X,Y)
             S3
          FIG 2.
```

In FIG 2, so as to make sure that ~P(X,Y) does not clobber P(A,B), i.e., we have to see to it that ((X ~= A) V (Y ~= B)).
Separation would involve putting additional constraints in the Pi part of the plan <T,O,Pi>.

Note: Lifschitz in his paper "On the Semantics of STRIPS" talks about the subpart of situational calculus that is dealt with in STRIPS - readings for next class.

^L

2.0 Pednault's approach:

Pednault tried to look at conditional effects with respect to a representation as far from STRIPS and as close to Situational Calculus as possible and still be able to avoid the frame, qualification etc. problems without using default reasoning. He was thus trying to set up an upper limit on the expressiveness of representation by taking it as close to situational calculus as possible. Though researchers at the time had difficulty finding usefulness of his work, his work was actually intended for a broader class of problems than planners like TWEAK were. With the arrival of UCPOP, his work received recognition.

A couple of questions that Pednault tried to address were:

(a) Is STRIPS representation the minimal one that short circuits the Frame problem or is there a representation that subsumes the STRIPS representation and still able to short-circuit the Frame problem?
(b) What problems of planning can classical planning solve?
If we knew then we could write a planner for that subclass of problems.

3.0 Classical Planners vs. Reactive Planners:

At around '87 when Pednault completed his work, there was a realization that even for simplistic action representations, like TWEAK (which is among the lower rung of useful Classical Planners), planning is undecidable. This frustration caused researchers to move towards Reactive Planning from Classical Planning.

In Reactive Planning execution of the plan is interwoven with the building of the plan. Classical Planning was a good choice when planning time of the plan was separated from its execution time. If these two times are related, then within the time that the planner has, it comes up with a plan that is more oriented towards the goals by making quick decisions to avoid the wrong paths rather than just picking a random plan and going with it.

^L

4.0 Motivation for ADL:

ADL integrates the semantics and the expressive power of situation calculus with the notational and computational benefits of the STRIPS language. ADL picks the best parts of STRIPS and situational calculus and avoids the difficulties encountered with each of them individually. It does this by choosing a middle ground between them. One of the ideas behind going to STRIPS from situational calculus is to go from intentionally represented states to extensionally represented states.

Situational calculus way of saying how an action changes a state is holds(q, R(a,s)) - here this says q will be true in R(a,s). You also have to say what is not changed - frame problem.

In STRIPS, a : S --> S - D + A just changes a state. STRIPS extensionally represents states. It syntactically modifies a state to go to another state. So, you cannot have actions having conditional effects because you do not explicitly know what is true after an action.

If we talk of actions as not causing a change of one state into another but from one state to a set of states (S --> {S}), then this allows us to handle actions with conditional effects, iterations, nondeterministic actions etc. For example, S --> {S1, S2} i.e. S causes S1 or S2, thus allowing actions with nondeterministic effects. This small syntactic change to STRIPS, improves greatly the kind of problems STRIPS could handle.

Here, an analogy could be drawn with programming languages where programs in them when executed, will take you into a state where something is true. In STRIPS, loops and nondeterministic actions are not allowed. So as to allow these in programming languages, people used dynamic logic, a variant of Modal logic, wherein you can talk about the correctness of programs and wherein actions are thought of as a mapping from a state to a set of states.

^L

4.1 Dynamic Logic:

For a certain world W, in FOPC, Q? is an allowed question for a well formed formula Q. In dynamic logic, questions like <h>Q? and [h]Q? are also allowed where h is an arbitrary function. [h]Q? implies "Is Q true in every world after executing h?". For this, we should also make sure that things like a register taking a very high value would not take place if the execution of h makes use of such a register.

In <h>Q if you allow h to be
  (i) a - program statement
  (ii) Q? - conditional action
  (iii) (a1 V a2) - disjunction
  (iv) a(n) read as 'a to the power of n' - execute a n times.

N.b.: If (ii) and (iii) are allowed then IF..THEN..ELSE statements could be written - (Q? a1 V ~Q? a2).

If h is allowed to assume all these possibilities, then we can come up with most statements of programming languages.

But the other question is 'How costly is it to check <h>Q or [h]Q'. To check Q? (Is a well formed formula true?) itself the complexity is NP-Hard. The complexity of the above true is Exp-time-complexity. When actually building a plan, you may have to check <h>Q? many times which would make the problem even harder.

All the logics we talked about can be related as below:

```
        Propositional Logic ---> First Order---> Situation calculus
                                 Predicate Calculus
                    |                 |
                    |                 |
                    V                 V
             Propositional    First Order Modal
              Modal logic          Logic
                    |                 |
                    |                 |
                    V                 V
            Propositional       First Order
            Dynamic Logic      Dynamic Logic
```

A predicate in Situation Calculus comes about by adding and argument to a corresponding predicate in FOPC - so, you obviously cannot have Situation propositional logic.

^L

5.0 Syntax of ADL:

ADL tries to keep away from STRIPS representation and tries to stay close to the dynamic logic way of representation. ADL schemas resemble STRIPS operators augmented with conditional add and delete lists, allowing the description of context dependent effects.

Briefcase moving Problem:

Please refer to the problem itself on page 357 of Pednault's paper on "Synthesizing plans that contain actions with context-dependent effects".

Using ADL syntax, four actions suffice to define and solve the problem.

(a) Putin(X) - put X in the briefcase,
(b) Takeout(X) - remove X from the briefcase,
(c) MoveB(l) - Move B to the location l.

Also, In(X) implies a state in which X is in the briefcase and At(X,Y) implies X is at location Y.

```
Putin(X) :
        Add: In(X) if (there exists) l [ At(X,l) & At(B,l) ]

Takeout(X) :
        Del : In(X).

MoveB(l) :
        Add: At(B,l); (for all)X (In(X) --> At(X,l))
        Delete: (for all)m At(B, m) (m ~= l)
                (for all)Z (At(Z, m) & In(Z)) (m ~= l)
```

Observe that here we have the niceness of the STRIPS representation - add and delete lists and we also have universality which allows for more expressive representation that STRIPS.

6.0 How do we give semantics to this representation?

The semantics of this ADL representation is got by relating it to situation calculus. Every problem in ADL can be written as a problem in situation calculus and so by knowing the semantics of situation calculus, we can know the semantics of ADL.

For the conversion of a problem from ADL to situation calculus, please refer to page 358 of Pednault's paper on "Synthesizing plans that contain actions with context-dependent effects".

```
        Situational Calculus
                |
                |               |
               ADL              | expressivity
                |               V
                |
             STRIPS
```

```
From rao  Thu Mar  4 19:30:18 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11203; Thu, 4 Mar 93 19:30:18 MST
Date: Thu, 4 Mar 93 19:30:18 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303050230.AA11203@parikalpik.eas.asu.edu>
To: plan-class
Subject: a *way* of writing briefcase problem in strips
Reply-To: rao@asuvax.asu.edu
```

Here is one way of writing Briefcase problem in strips rep.

Qn. Are there any problems that could be solved by the previous representation that couldn't be solved by this rep?

qn. Am I right in saying that the above(x,y) predicate couldn't be handled in STRIPS? Or can you think of a way of representing it?

qn. I said that things which would be synergistic effects for strips can be dealt with in ADL as ordinary effects because of ADL's ability to use conditional effects. Is this also true for effects of simultaneous effects (ie. things of the type, lifted(table) is true in a state s if and only if lifting-from-right(table) and lifting-from-left(table) are both true at the same time.

```
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
To: rao@enws318.eas.asu.edu
Date: Thu, 4 Mar 93 15:42:35 MST
```

Briefcase Problem in Strips:
  My simple solution is to delete the location of objects when they are in the briefcase. This means that the initial situation would be

```
   (at b home) (at d home) (in p)
                             ;note that (at p home) is not there
                             ;since p is in the briefcase

operator mov-b
  parameters ?m ?l
  preconditions
      (at b ?m)
  add (at b ?l)
  del (at b ?m)

operator put-in
  parameters (?x ?l)
  preconditions
      (at ?x ?l)
      (at b ?l)
  add (in b)
  del (at ?x ?l)             ;note that put-in deletes at

operator take-out
  parameters (?x ?l)
  preconditions
       (at b ?l)
       (in x)
  add  (at x ?l)
  del  (in x)                ;note that take-out adds at


Goal: (at b office) (in d) (at p home)
                             :note that this is the equivalent goal

Final Plan :
   (take-out p home)
   (put-in d home)
   (move-b home office)
```

Laurie.

```
From rao  Thu Mar  4 19:54:13 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11212; Thu, 4 Mar 93 19:54:13 MST
Date: Thu, 4 Mar 93 19:54:13 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303050254.AA11212@parikalpik.eas.asu.edu>
To: plan-class
Subject: Regression for non-deterministic actions (response to Gopi's qn)
Reply-To: rao@asuvax.asu.edu
```

note: non-deterministic actions are actions with disjunctive effects. The idea is that any real action will wind up committing to one or the other disjunct of its disjunctive effect in a given real execution.

Gopi asked whether regression is defined for actions with non-deterministic effects. Pednault's CIJ paper (pp. 366, first col. end) discusses this. The general idea is that when we have actions with non-deterministic effects, we can no longer define regression as necessary and sufficient conditions -- in particular, as Gopi was probably driving at, there is no sense in talking about the necessary and sufficient conditions when you don't know which disjunct of a disjunctive effect is going to be true.

In particular, consider an action a which has a disjunctive effect
 p V q, no preconditions and no other effects.

Now, what are the weakest preconditions that need to be true before a so we know for sure that p will be true afterwards?

Well, sometimes, just a simple execution of a will give p (i.e., non-deterministically select p of p V q and make that true). So, in those cases, weakest conditions is "True".

But, some times the action's execution may non-deterministically commit to q out of p V q, and in those cases, the only way to ahve p

```
true after a is to have p to be true before a.

So, weakest conditions are "sometimes p, and sometimes True".
Since we don't know a will give p and when it will give q, we can't
write the quoted expression above as a logical formula.

In otherwords, there is no way we can express necessary as well as
sufficient conditions.

Now, consider defining p as the regression of  p over a. This seems
reasonable in that irrespective of what a winds up doing, if p is true
before a then p will be true after a. Moreover, this is the only
guaranteed statement we can make as to what needs to be true before a
to guarantee truth of p after a.

So, this is the only reasonable way to define regression for
non-deterministic actions. In otherwords, what we are doing is to
define regression as weakest *sufficient* conditions rather than
necessary and sufficient conditions.

Now, when you have regression defined as weakest sufficient
conditions, then the nice distribution properties that we discussed in
the class no longer hold.

In particular, as discussed above a-(p) = p and a-(q) = q

so what is a-(p V q)?
if we write it as a-(p) V a-(q) then we will get it as p V q -- i.e.,
p v q needs to be true before a for p v q to be true after.

However, this is clearly not the weakest sufficient conditions for
ensuring p V q after a. In fact, a will always give p v q so really,
weakest sufficient conds of p V q over a is True, which is different
from p V q

in other word a-(pVq) != a-(p) V a-(q)

This is what throws a spanner into the works, and makes regression
much costlier (we can no longer say that regression of a complex
formula can be written in linear time as regression of the atomic
formulas :-().

Rao

ps: thanks to Gopi for asking this qn. in the class


From ihrig@enws318.eas.asu.edu Fri Mar  5 11:21:16 1993
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 "^From:" nil nil nil])
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
	id AA11601; Fri, 5 Mar 93 11:21:15 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
	id AA01216; Fri, 5 Mar 93 11:19:01 MST
Message-Id: <9303051819.AA01216@enws318.eas.asu.edu>
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
To: rao@enws318.eas.asu.edu
Date: Fri, 5 Mar 93 11:19:01 MST

Answer to questions March 5, 1993        Laurie H. Ihrig

1.  My answer to the briefcase question requires some
preprocessing on the part of the user.  For example, if
the goal of at(d, office) is required, one must consider
this goal to be a disjunction:

    at(d,office) V (in(d) ^ at(b, office)).

This can be handled in STRIPS with my briefcase domain,
providing that the two goals can be attempted in succession.
If this is possible then all potential goals in the briefcase
domain that are solvable by UCPOP are also solvable by STRIPS.

2.  ABOVE in the blocks' world is similar to AT in the
briefcase domain in that it can be treated as
a disjunction:

the goal of above(a,b) can be represented as
on(a,b) V (on(a,?x)^on(?x,b)) V (on(a,?x)^on(?x,?y)^on(?y,b) etc.

with the number of disjuncts equal to the total number of blocks
minus one.

3. Simultaneous effects can be handled in STRIPS by representing
each action by a pair of operators, one that starts the action
and has an effect that indicates that the action is in progress,
and another that ends the action.  In the case of the lifting example,
there is one ending action whose precondition is that
both lifting-right and lifting-left are in progress.  The operators
are:
operator LIFT-RIGHT
  preconditions   on(table,floor)
  add             lifting-right(table)

operator LIFT-LEFT
  preconditions  on(table,floor)
  add            lifting-left(table)

operator END-LIFT
  preconditions  lifting-right(table)
                 lifting-left(table)
  add            lifted(table)
  del            on(table,floor)

GOAL  :  lifted(table)

FINAL PLAN: lift-right
            lift-left
            end-lift

                                 Laurie.

From rao Thu Mar  4 19:30:36 1993
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 "^From:" nil nil nil])
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
	id AA11203; Thu, 4 Mar 93 19:30:18 MST
Message-Id: <9303050230.AA11203@parikalpik.eas.asu.edu>
```

```
Reply-To: rao@asuvax.asu.edu
From: rao (Subbarao Kambhampati)
To: plan-class
Subject: a *way* of writing briefcase problem in strips
Date: Thu, 4 Mar 93 19:30:18 MST


Here is one way of writing Briefcase problem in strips rep.

Qn. Are there any problems that could be solved by the previous
representation that couldn't be solved by this rep?

qn. Am I right in saying that the above(x,y) predicate couldn't be
handled in STRIPS? Or can you think of a way of representing it?

qn. I said that things which would be synergistic effects for strips
can be dealt with in ADL as ordinary effects because of ADL's ability
to use conditional effects. Is this also true for effects of
simultaneous effects (ie. things of the type, lifted(table) is true in
a state s if and only if lifting-from-right(table) and
lifting-from-left(table) are both true at the same time.


---------------

From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
To: rao@enws318.eas.asu.edu
Date: Thu, 4 Mar 93 15:42:35 MST

Briefcase Problem in Strips:
  My simple solution is to delete the location of objects when
they are in the briefcase.  This means that the initial situation
would be
   (at b home) (at d home) (in p)
                             ;note that (at p home) is not there
                             ;since p is in the briefcase

operator mov-b
  parameters ?m ?l
  preconditions
      (at b ?m)
  add (at b ?l)
  del (at b ?m)

operator put-in
  parameters (?x ?l)
  preconditions
      (at ?x ?l)
      (at b ?l)
  add (in b)
  del (at ?x ?l)             ;note that put-in deletes at

operator take-out
  parameters (?x ?l)
  preconditions
       (at b ?l)
       (in x)
  add  (at x ?l)
  del  (in x)                ;note that take-out adds at

Goal: (at b office) (in d) (at p home)
                             :note that this is the equivalent goal

Final Plan :
   (take-out p home)
   (put-in d home)
   (move-b home office)
                                                         Laurie.

From @VM.USC.EDU:AGWCT@ASUACVAX.BITNET  Tue Mar  9 15:25:25 1993
Return-Path: <@VM.USC.EDU:AGWCT@ASUACVAX.BITNET>
Received: from VM.USC.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA15585; Tue, 9 Mar 93 15:25:25 MST
Message-Id: <9303092225.AA15585@parikalpik.eas.asu.edu>
Received: from ASUACVAX.BITNET by VM.USC.EDU (IBM VM SMTP V2R2)
   with BSMTP id 2583; Tue, 09 Mar 93 14:00:16 PST
Date:     Tue, 9 Mar 93 14:59 MST
From: <AGWCT%ASUACVAX.BITNET@VM.USC.EDU>
Subject:  Notes for Mar. 4
To: plan-class@enws228.eas.asu.edu
X-Original-To:  plan-class@enws228.eas.asu.edu, AGWCT

Notes for the Class of Planning Seminar on Mar. 4, 1993
Written by Wan-Chu Tsai

Agenda:
1. More formal issues of ADL representation
2. ADL and Total Ordering Planner
   * Regression
   * Causation and Preservation
   * Regression Assertionability
3. ADL and Partial Ordering Planner
   * Avoiding multiple sets of secondary precondition
   * Dealing with secondary precondition in a lazy fashion

Last class we see there is a uni-directional mapping from ADL to Situation
Calculus, this class we will start with situation calculus to see the
motivation of using its subset, ADL.

Reference Paper by Pednault:
ADL : Exploring the Middle Ground Between STRIPS and the Situation Calculus

(P.S. For all axiom schemes, please refer to Pednault's paper)

1. Two types of axioms (axiom schemes) that need to be written down:

   (1) State-change axiom   (Ref: section 2, 2nd paragraph of Pednault's
       paper)
   (2) Frame axiom   (Ref: section 2, 3rd paragraph)

2. ADL is a subset of schemes that can be derived from these two axiom
   schemes:

   (1) ADL requires that there be a separate axiom for each predicate.
       (relation and function)
       This is the syntactic restriction.
       The state-change axiom for a relation is shown in the 5th paragraph,
       section 2.
       The first axiom is called ADD clause, it represents all the conditions
       under which action A will add R.
       The second is called DELETE clause, it represents all the conditions
```

```
        under which action A will delete R.
        If we take any ADL axiom and write it in situation calculus, it will
        look like one of these two clauses.
        ADL allows updates, values of functions can be changed.  There is
        also state-change axiom for function, shown in the 7th paragraph,
        section 2.
        We are interested in under what conditions the function will have
        value y.  At any given time, function has unique value.

    (2) The only effects of an action are those mentioned in the axiom.

 3. Questions :

    (1) What can't ADL do?
        * Disjunctive effects are not allowed, i.e. actions with
          nondeterministic or incomplete specification can not be handled.
        * ADL does not allow existential quantification, but it allows
          universal quantification.

        The relation between ADL and situation calculus is analogous to that
        between PROLOG and first order logic.

        ADL can do more efficient inference.

    (2) Do we need to write frame axiom?  No.
        Because we have written down what changes will happen when applying
        action, by state-change axioms, the frame axioms are in fact written
        implicitly.
        Two frame axioms:
        If R is true before action A, then after action A, R is still true.
        If R is false before action A, then after action A, R remains false.
        Frame axioms for relation and function are shown in the 9th and 10th
        paragraphs of section 2, respectively.
        ADL is more restricted than situation calculus in syntactics.
        Not writing frame axioms explicitly can avoid doing default reasoning.

    (3) Although ADL can't have disjunctive effects, disjunctive preconditions
        are allowed.
        a v b -> c  is equivalent to a -> c and b -> c.
        Proof:      a v b -> c
                =>  ~(a v b) v c
                =>  (~a ^ ~b) v c
                =>  (~a v c) ^ (~b v c)
                =>  (a -> c) ^ (b -> c)

        * Important: If we have axioms with disjunctive conditions, we can
          separate it as above only when we don't have disjunctive states.

        There is no restrictions on initial and goal states, because one of
        the disjunctive conditions is true, the whole disjunction is true.

        UCPOP, unlike ADL, does not allow disjunctive preconditions and goals.

 4. If there is no restriction on initial state or goal, can we have action
    with disjunctive effects?  No.  Because there is a difference between
    reasoning in the action and checking the initial and goal states.
    ADL uses regression to reason about the effects of action.
        For example,
           a1  ->  a2  ->  a3  ->  a4
        Question: Is p true after a4?
            Let ai~1 be the regression operator of action ai.
            Let P = a1~1(a2~1(a3~1(a4~1(p))))
        Given I be the initial state,
        The above question is equivalent to "whether true or not  I |= P?"

    * Regression will be simple without disjunctive effects, which is
      polynomial in ADL.

    * Two steps:
       (1) Compute regression P
       (2) Compute I |= P

       ** The notion of regression is together with the complexity of truth
          criteria.
          If there is disjunctive effects, P is undecidable.
       ** The complexity also depends on the completeness of the initial
          state.

    * Why not doing theorem proving in ADL?
       (1) It is not the frame problem that causes the switch from situation
           calculus representation to STRIPS.  It is because resolution
           theorem proving is notoriously hard to control.
       (2) The syntactics of STRIPS is easy for direct search using proper
           heuristics.

 5. Example in ADL:  Please refer to section 3 for the example of put(b, l).

    * Discussion:
       on(C, B) ^ above(A, B) (*1) is equivalent to on(C, B) ^ on(A, C) (*2)
       This equivalence is not a simple example for STRIPS.  We don't know
       that (*2) can be got by solving (*1) before we solving this problem.
       For STRIPS to solve this problem, we need two axioms:
       (1) Forall x,y on(x, y) -> above(x, y)
       (2) Forall x,y,z on(x, y) ^ above(y, z) -> above(x, z)

       Thus, STRIPS cannot represent the example of put(b, l).

       ** When one action makes on(C, B) true, while the other action makes
          on(A, C) true, it is the combination of two actions that makes
          above(A, B) true.  ADL will explicitly represent this condition.

       ** ADL allows more problems to be solved, because it is more
          restricted.

 6. Compute ADD and DELETE clauses:

    (1) To compute the ADD clause, look at the add list of the action.  The
        example is shown in the 5th page of Pednault's paper, right below
        Table 2.
        * In the formula, the existential quantification is needed because b
          and l are parameters and must be treated as constants.

    (2) To compute the DELETE clause, look at the delete list of the action.
        The example can be found right below the ADD clause mentioned above.

 7. Compute regression:
    Given action a, condition p,

    * What is the condition that must be true so that when action a is done,
      p will be true?
      (1) a adds p.
      (2) p is true before executing a, and a does not delete it.
```

The bottom of the 2nd column on the 6th page of Pednault's paper shows an example of computing put(A,B)~1[above(A, D)].

* There are some equations that can be applied to all atomic subformulas of a formula, shown in the middle of the 2nd column on the 6th page of Pednault's paper.
  Regression represents global properties. Using these properties, we only have to compute regression for every action once. This is the unique for ADL. And the reason is because of the lack of disjunctive effects.

```
From rao  Wed Mar 10 09:45:25 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA15895; Wed, 10 Mar 93 09:45:25 MST
Date: Wed, 10 Mar 93 09:45:25 MST
Message-Id: <9303101645.AA15895@parikalpik.eas.asu.edu>
From: rao (Subbarao Kambhampati)
Sender: rao
To: plan-class
Subject: puzzler 3
```

Consider the ART-MD-RD and ART-1D-RD domains described in my "Utility of Systematicity" paper.

Weld et al (TR) show that ART-MD and ART-1D are trivially serializable for SNLP (POCL).

I would like you to attempt to prove that ART-MD-RD (or ART-1d-RD) is laboriously seriablizable for POCL while it is trivially serializable for MP, MP-I and UA (see my systematicity paper for the description of MP and MP-I)

Rao
[Mar 10, 1993]

```
From rao  Wed Mar 10 17:47:54 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA16279; Wed, 10 Mar 93 17:47:54 MST
Date: Wed, 10 Mar 93 17:47:54 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303110047.AA16279@parikalpik.eas.asu.edu>
To: plan-class
Cc: rao
Subject: Midsemester survey
Reply-To: rao@asuvax.asu.edu
```

The following is a survey on the way planning seminar is going. I would like responses from all of you who are attending the meetings (whether or not you are registered for the course).

You may want to save the mail to a file, edit it with your responses, print the file (using lzpr or some such) and leave it in my mailbox in the dept (this will ensure anonymity to a very reasonable extent).

I would appreciate textual rather than binary (yes/no) answers/feedback. I would like to get these forms by Friday.

I made the questions acting as my own devil's advocate, but may still have missed some useful questions. If you have comments that are not related to any of the questions, *please* feel free to add.

cheers
Rao
[Mar 10, 1993]

Feedback for CS 591 Planning Methods in AI seminar
You do not have to write your name.

Qn 1. Do you like/hate/don't-care the way the seminar is being conducted?

Qn 2. Do you think the the seminar is sufficiently/insufficiently interactive?

Qn 3. Is the style of seminar conducive to fostering your thinking? Or is it done in a way to _intimidate_ your thinking/creativity? Or is it just one long boring lecture?

Qn 4. Does the style of "asking questions" make you feel as if the instructor is trying to "show off" rather than make you think?

Qn 5. Do you notice any "favoritism/bias" towards certain students/viewpoints being displayed by the instructor?

Qn 6. Do you feel that the discussion in the class becomes too esoteric, and too stuck in wierd details that you can't see as being useful?

Qn 7. Do you keep up with the readings? (remember your answer is anonymous) Do you think that the class demands you to read TOO MANY papers and get a little from each, instead of reading a few papers thoroughly?

Qn 8. Don't you think it is just great that the class always ends on time (more or less)?

Qn 9. How uncomfortable are you with the format of the course, which is open ended, and perpetually touching the state of the art? Would you have been happier with stuff that is more established and traditional?

From elder@enuxhb.eas.asu.edu Thu Mar 11 14:54:41 1993
Return-Path: <elder@enuxhb.eas.asu.edu>
Received: from enuxhb.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA17802; Thu, 11 Mar 93 14:54:41 MST
Received: by enuxhb.eas.asu.edu id AA07122
(5.65c/IDA-1.4.4 for plan-class@parikalpik.eas.asu.edu); Thu, 11 Mar 1993 14:4
6:51 -0700
Date: Thu, 11 Mar 1993 14:46:51 -0700
From: Greg Elder <elder@enuxhb.eas.asu.edu>
Message-Id: <199303112146.AA07122@enuxhb.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

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
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  Fri Mar 12 14:52:33 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA00610; Fri, 12 Mar 93 14:52:33 MST
Date: Fri, 12 Mar 93 14:52:33 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303122152.AA00610@parikalpik.eas.asu.edu>
To: plan-class
Subject: classes reminder
Reply-To: rao@asuvax.asu.edu
```

Please remember that we will not be meeting in the week after spring break. The next regular class meeting will be 30th March. At that time, we will be looking at Hierarchical Planning (Readings: Tate's Nonlin, Sacerdoti's Noah, Wilkin's SIPE in Readings in planning).

We will make up for the one last class in April.

Rao

ps: Please remember to turn in the survey forms today.

```
From ihrig@enws318.eas.asu.edu Fri Mar 12 14:44:30 1993
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil "Laurie Ihrig" "ihrig@enws318.eas.asu.edu "
 nil nil nil "^From:" nil nil nil])
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA00531; Fri, 12 Mar 93 14:44:30 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA05987; Fri, 12 Mar 93 14:42:07 MST
Message-Id: <9303122142.AA05987@enws318.eas.asu.edu>
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
To: rao@enws318.eas.asu.edu
Date: Fri, 12 Mar 93 14:42:07 MST
```

Rao
This is my second try at a counterexample to Dsep space <= SNLP--for your eyes only until I can check it out more thoroughly.   Laurie.

```
OPERATORS:

move(?X,?Y)
      precond   atdudley(?X)
      add       atdudley(?Y)
      del       atdudley(?X)

untie-nell(?L)
      precond   atdudley(?L)
                tied-nell(?L)
      add       untied-nell
      del       tied-nell(?L)

GOAL:
        atdudley(track) and  untied-nell

INITIAL SITUATION:

        atdudley(track)
        tied-nell(track)

one path of SNLP derivation would be:

        0-------G
            .
            .
            .
```

```
    .

 atdudley(track)
  ________
  |        |
           V
  0-------G

      .
      .
      .
      .

  0-------untie-nell(L1)---------G

             L1=track


      .
      .
      .
      .


               atdudley(track)
   ______________________________________________
  |                                              |
  |           atdudley(L1)                       |
             _____________________               V
            |                     |
                                  V

  0------move(X1,Y1)-------untie-nell(L1)-------G

         Y1=L1               L1=track
         X1/=Y1

         L1/=track from separation

         inconsistent constraints    pruned by SNLP
                                     not pruned by Dsep

      .
      .
      .
      .

          atdudley(X1)
       ___________________
      |                   |
                          V
0---move(X2,Y2)--------move(X1,Y1)------untie-nell(L1)-------G

     Y2=X1               Y1=L1           L1=track
     X2/=Y2              X1/=Y1

            not pruned by Dsep

      .
      .
```

```
                  .
                  .

 0----move(X3,Y3)-------move(X2,Y2)-------move(X1,y1)----untie-nell-----G

                  not pruned by Dsep

                   etc
```

```
From rao  Fri Mar 19 13:31:49 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA09988; Fri, 19 Mar 93 13:31:49 MST
Date: Fri, 19 Mar 93 13:31:49 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303192031.AA09988@parikalpik.eas.asu.edu>
To: plan-class
Subject: A puzzler problem [forwarded message from Jim Hendler]
Reply-To: rao@asuvax.asu.edu

        From ginsberg@CS.Stanford.EDU Fri Mar 19 13:29:54 1993
        Date: Fri, 19 Mar 93 10:29:38 PST
        From: Matthew L. Ginsberg <ginsberg@CS.Stanford.EDU>
        Message-Id: <9303191829.AA15462@t.Stanford.EDU>
        To: dam@ai.mit.edu, hendler@cs.umd.edu, mc@CS.Stanford.EDU,
                mcdermott-drew@cs.yale.edu, minton@cs.umd.edu, nau@cs.umd.edu,
                principia@cs.umd.edu, wilkins@ai.sri.com
        Subject: odd little planning problem
        Cc: ginsberg@CS.Stanford.EDU
        Status: RO


        The situation is the blocks world, but there’s an extra rule -- you can’
t
        build a 4-block tower.  So an extra precondition to move(x,y) is that
        either y is on the table or y is on a block that is on the table.

        The initial situation is:

                        c
            a     b     d
           --------------

        The goal is to get a on b and b on c.

        Any comments?

                                                        Matt



------- End of forwarded message -------


From rao  Sat Mar 20 18:08:12 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11183; Sat, 20 Mar 93 18:08:12 MST
```

```
Date: Sat, 20 Mar 93 18:08:12 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9303210108.AA11183@parikalpik.eas.asu.edu>
To: suresh@enws318, gopi@enws318, ihrig@enws318, dchen@enws228, cohen@enws318
Cc: plan-class
Subject: a small combinatorics (enumearation) problem
Reply-To: rao@asuvax.asu.edu

Here is a simple combinatorics problem for which I need an answer by
Monday.

Suppose you have m distinct action (a1 a2... am)

and you want to find out how many action sequences of lengh less than
or equal to n are there

Clearly the answer is m^n . This is in some sense an upper bound on
the size of the search space of a total ordering planner.


The question now is how many DISTINCT (ie. non equivalent) partial
orderings of size less than or equal to m can we make? We will
consider two partial orderings to be equivalent if the set of their
topological sorts are identical (in otherwords, they have same
transitive closure).

It is clearly greater than m^n since every distinct operator sequence
is also a distinct partial order. But there are many more

(e.g a1//a2  (a1-a3)//a4 etc (where // is to be read as paralle to).

The question is EXACTLY how many more?

Any help will be hightly appreciated... I still don't know the answer

Rao
[Mar 20, 1993]

ps: I believe it may be related to (but slightly different from) the
following problem

How many  directed acyclic graphs with distinct transitive closures
can be drawn on n vertices?


Rao
[Mar 20, 1993]

From gopi@enuxha.eas.asu.edu  Thu Apr  1 16:37:36 1993
Return-Path: <gopi@enuxha.eas.asu.edu>
Received: from enuxha.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA23391; Thu, 1 Apr 93 16:37:36 MST
Received: by enuxha.eas.asu.edu id AA02956
  (5.65c/IDA-1.4.4 for plan-class@parikalpik.eas.asu.edu); Thu, 1 Apr 1993 16:33
:47 -0700
From: Bulusu Gopi Kumar <gopi@enuxha.eas.asu.edu>
Message-Id: <199304012333.AA02956@enuxha.eas.asu.edu>
Subject: Class notes of March 30 1993
To: plan-class@parikalpik.eas.asu.edu
Date: Thu, 1 Apr 93 16:33:43 MST
X-Mailer: ELM [version 2.3 PL11]
```

```
                                 AI Planning
                                   3-30-93

-------------------------------------------------------------------------------

Agenda
------
        * Foundations for Automatic Planning : Classical approach and
          beyond - Symposium Summary

        * Systematicity

        * HTN (Hierarchical Task Network) planning

-------------------------------------------------------------------------------


Symposium Summary
-----------------

The current state of the art in classical planning is represented by
ADL. Infact work is in progress by Dan Weld, et al, on an enhanced
version of UCPOP named ZENO.

ZENO
----

ZENO allows

        * Goals quantified over time
        * Metric constraints

An example of a metric constraint would be

 ex : FOR ALL t Quantity of gas in plane at time t >= 15 gallons

This is a constraint, which is continuous in nature, which means that
the check needs to be done at an infinite points. But, if the Quantity
of gas changes in a particular fashion, say, linear then it is
sufficient to check just at the end points of the function.

In general constraints are used by the planner to check for
consistency. Existing planners check for binding/ordering constraints.
Similarly metric constraints have to be checked for consistency. The
problem is that it is not even possible to check for arbitrary metric
constraints. However if the constraints are linear, it becomes a
linear programming problem.

A linear programming problem is in the form

maximize : a1x1 + a2x2 + a3x3 + .. anxn
given

        c11x1 + ..... c1nxn >= 0
        c21x1 + ..... c2nxn >= 0

        ...
```

cm1x1 + ... cmnxn >= 0

In planning, it is just sufficient to find if the constraints are consistent. This is a smaller problem than the full LP problem. Which means that the polytope described by the constraints is not null. This can be checked by the first phase of the famous SIMPLEX algorithm.

Metric constraints can be used to describe ordering constraints, hence these can be checked in the same way. ZENO allows for such linear constraints.

Practical planning

A few questions arise in the case of practical planning

* What is the state of practical planning ?
* How to avoid classical planning assumptions ?

Opinions

STRIPS - Blocks World fanaticism

Why is it that most of the researchers are focusing on the STRIPS styled operators in the Blocks world domain ? Is it because it is the most difficult domain ? Is it because other problems are hard to solve??

SNLP Takes the role of STRIPS

Most of the work done in the past involved STRIPS (including criticism). The focus has now shifted to SNLP ! Are the reasons similar to the above ? Is that a healthy sign ??

Questions

What Should be a real planner ?

Definition 1

"Real Planner" is something for which

* Some body is willing to pay (and buy)
* There is a neat GUI
* 90% work is done in domain modeling

While being an important aspect of a planner, most of the planners do not take into account the user in their design, and hence are written for an efficient implementation and not in a way that will allow user to understand the planning process and/or selective participation by the user in the planning process.

Problems with this definition

* Why can't I use Mac Project ? (look at 2nd point in defn)

- Only one person claimed to have sold his planner ! But the planner was used for research purposes by the customer !!

- Domain modeling involves a lot of work by the programmer. In general this may be important in the practical view, but may not provide right directions for research

Definition 2

A planner should be able to work in a real domain and support real engineering to acquire domain knowledge. A real domain is defined as a domain which includes the following :

* Events
* Resources
* Duration and Deadlines
* Missing information
* Uncertain effects
* Continuous change
* Context dependent effects
* Execution and replanning

Process planning

An example of this would be process planning. In general process planning involves generation of an efficient plan, consisting of a sequence of mechanical operations (drilling, milling, boring, filing etc) to make a part, whose description is given. It is a problem which exists and could benefit from automation. It is in some ways easier than blocks world and in other ways difficult.

Easier because, there are not too many interactions between the various operations. In general they can be performed in any order.

Difficult because, often locally efficient methods (drilling instead of honing) may result in inefficient plans ! Honing 2 holes may cost less than drilling one and honing the other, the later may involve change of the tool used. (Changing tools, changing the orientation of the part etc are more costly than which operation is actually done). In general to produce an efficient plan, the planner needs to know the following

* Geometry
* Mechanical operators
* Amount of force to apply,etc

All this needs to be used at the same time and thus will increase the complexity of the problem and hence the planner.

Real Engineering

This deals with the problem of acquiring domain knowledge. Domain knowledge should be acquired in a way which is simple to the user.

One way of acquiring such knowledge may be through HTN. Domain experts typically organize their knowledge in a hierarchical manner. Therefore it is easy for them to provide information in the same way rather than in the form of say STRIPS styled operators.

Conclusion on "what is a real planner"

The first definition seems to be an advice to the practisioner and the second definition seems to be an advice to the researcher.

TASK REDUCTION IS HERE TO STAY

Rather than thinking about planning as working on a goal, using an operator to achieve it, working then on the preconditions of the operator (sub goals), Hierarchical Task Network planners look at the goal, look at what are the tasks (abstract) which achieve the goal, and then work on these tasks, till there are no abstract tasks left (when all the tasks left are primitive actions).

NOAH was the first planner to use this technique. NOAH differed from the earlier STRIPS in the following ways

* It was a partial ordering planner
* It used task reduction

An example for a HTN operator could be

```
Task : Make-Hole-By-Drilling

SubTask : Position-Hole
SubTask : Rough-Drill
SubTask : Finish-Drill
```

This is how humans think ! "Reduction Based View of Plans" and many of the planners work this way ! However there is no formalisation of HTN !

The question is - Why not ? Is it because it is just an efficiency hack ?

The fact remains that, even if it is an efficiency hack, it is too important a detail to ignore

Against this backdrop, two alternatives exist for an SNLP like planner

* Take the knowledge in the form of HTN's and then split it into its internal form ! In the process ofcourse the planner will lose all the important control information present in the HTN.

* Take information in the form of STRIPS styled operators (not a good marketing idea)

However the right solution is to understand and formalise task reduction. If information provided by the user can be coded into search control rules, then HTN's can be ignored. Howeer HTN's provide more information than search control rules. They have the intermediate structure of knowledge. It is extremely difficult even for a human to acquire this knowledge ! In particular, this knowledge can't be provided by current learning techniques. It then seems logical to get this information from the user --> "Task reduction is here to stay"

gopi

From AZEDC@ACVAX.INRE.ASU.EDU Sun Apr 4 19:15:05 1993
Return-Path: <AZEDC@ACVAX.INRE.ASU.EDU>
Received: from ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA27044; Sun, 4 Apr 93 19:15:05 MST
Received: from ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF #2382) id
<01GWMHCDQRNK001ABB@ACVAX.INRE.ASU.EDU>; Sun, 4 Apr 1993 19:12:19 MST
Date: 04 Apr 1993 19:12:18 -0700 (MST)
From: AZEDC@ACVAX.INRE.ASU.EDU
Subject: Planning class notes, Apr 1
To: plan-class@enws228.eas.asu.edu
Message-Id: <01GWMHCDR1AQ001ABB@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"plan-class@enws228.eas.asu.edu"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT

Notes for April 1, 1993 (by Eric)

Agenda:
. syntax of hierarchical task networks (a little bit of semantics)
. differences between HTN and precondition abstraction

Precondition abstraction
. Introduced in ABSTRIPS
. The idea is that we work on the higher-order preconditions and then move on to the lower level ones
. Precondition abstraction eventually evolved into Alpine (by Knoblock). We could think of investigating the performance comparison between planners without abstraction, with abstraction given by the user, and with abstraction given by the system

Precondition abstraction vs. HTN
. Planning at various levels of space abstractions (ABSTRIPS)

```
p1          actions of pi+1 are in the same terms as in
:           pi but description of the world in pi+1
p2          is more detailed
:
p3
```

. Planning at successive levels of plan abstractions (SIPE,NONLIN,NOAH)

```
p1          actions of pi+1 are refinements of actions of pi
:           but the representation of the world is in the same
p2          terms at all levels
:
p3
```

Hierarchical task networks
HTN was introduced in NOAH, which went a different way from ABSTRIPS (incidentally, NOAH introduced two new concepts, the HTN and the partial ordering planning).

In HTN, we have low level operators and express them as task reduction schemas.
Example:

```
        task:          to travel from x to y        (a.k.a. achieve tasks,
                                                    non-primitive tasks)
        schema:   travel by train
          step0:  buy insurance
          step1:  book train ticket
          step2:  go to station
          step3:  get in train             (a.k.a.   do   tasks,  primitive
                                           tasks)
```

The higher level task is decomposed into the lower level ones:

```
        travel                        (LEVEL 0)
          :
    ----------------------------------------
    insure    book      station   get in
```

Note that tasks may be strictly or partially ordered with respect to each other; for instance, it may be the case in the example above that step 0 can be done in parallel with steps 1 and 2, step 1 must be before 2, 2 and 0 before 3 and so on. Notice that this is important ordering information

```
        travel                        (LEVEL 1)
          :
    : ------------------:-------
    : book    station   : get in
    :                   :
    : ----------------- :
      insure
```

Interactions between tasks are not always trivial. For example, at level 0 we might not be aware of any interactions between a step "buy paper" which occurs in parallel with all others, but perhaps at the level 1 we might be able to do so.

Planning consists of picking a task and reducing it to more primitive, executable tasks. We start from the dummy tasks (null plan) and a "to-achieve-all-goals" non-primitive task. We consider planning done when all the tasks are primitive and the plan satisfies the preconditions of the tasks.

Notice that it is immaterial whether a given task is to be considered primitive or non-primitive. This is a relative notion, and what is primitive for one agent may be non-primitive for another one.

Notice also that we are throwing away the notion of executability. Tasks may or not be executable.

Finally, it is important to notice that we still want to use all the theories we have seen so far, e.g. making sure dependencies among steps are taken care of, avoiding ordering steps when that is not really necessary, and so on.

With the hierarchical structure, we are hoping to take care of the most important interactions beforehand, leaving the "details" for later. The less important interactions will be taken care of if and when they appear.

The important interactions are provided by the user. We could think of the planner optimizing/ verifying the interactions provided by the user as well (sort of like "reverse engineering" the interaction information given by the user)

Notion of hierarchical reasoning

. constraints on the solution plans are iteratively refined
. each solution plan (or set of) at each iteration is used as a 'skeleton plan' to guide the next iteration
. there is no objective hierarchy independent of the problem. Hierarchy is a point of view => hierarchical planning is an heuristic-based reasoning aimed at reducing the average computing time. One person's detail is another person's importance
. hierarchical planning hopefully avoids propagating mistakes; recovering from mistakes not detected earlier is often very costly

Some planners make sure any effect promised by a task is actually given:

```
        +p
     :  sg1  taska :
   --:      (p)   :----    It may be the case that when sg1 is decomposed,
     :   taskb    :        the effect  p  is  not  given  by  any  of  the
                           primitive
                           tasks;  in  this  case  we  may  add  an  extra
                           primitive tasks which provides it
```

HTN does not increase expressiveness; all effects of the top level actions are present in the low level actions. For example,

```
   to-do:    roundtrip(x,y)      In planners  like NONLIN,  the information
                                 of
    step1:   travel(x,y)         roundtrip is not  "greater" than the "sum"
                                 of the
    step2:   travel(y,x)         information of the parts

                                 It could  be the case,  however, that this
                                 information were greater than the sum.  In
                                 this  case, we are departing from the idea
                                 that   HTN   is   only  providing   control
                                 information, because now it is also giving
                                 increased expressiveness
```

Comments
. Each action is of the form "achieve (atom)"
. Each action is either primitive or decomposable

A primitive action is directly executable by an existing program whose preconditions are satisfied.
A decomposable action is a too-high-level goal or some precondition not achieved yet.

. Each operator maps a decomposable action into a non-linear plan of more detailed actions
. Each action is described by delete and add lists, by the operator that generated it

In addition, the effects of the parent actions are transferred to the last action of the generated subplan

. The goal achieved by an action is considered as its principal effect (element of the add list). The principal effect inherited from a parent action is also a principal effect of the more detailed action

When a subplan sp is generated by an operator, the principal effects of each action in sp, except the last one, are considered the preconditions of the final action

Example:

```
     +q
     T2 ...... Tx             LEVEL 0
     (r)       (p)


     ........
     ........
               ...... Tx      LEVEL 1
     ........


        -p
     .. Ty ..
```

When the planner decomposes the high level tasks, the htn is considered a a "good" hierarchy if it doesn't just accept what the user specified. In other words, it will point out that there is a possibility that condition p is not necessarily true at step Tx. It is common however to find htn's do not do that (they just take whatever was specified by the user).

Macro-operators
Macro-operators are "black boxes", which are introduced when there are efficiency/ performance issues. For example, in STRIPS, the order in which sub-goals are tackled is very important; for that, we may incorporate that knowledge into a new macro-operator. Thus, we'd like to remember that knowledge, and REUSE it any time we are faced with a conjunctive goal, as is shown in the example below:

```
on(a,b)^on(b,c) ..... use operator op1(a,b,c) below

operator op1(x,y,z)
 step-a: stack(y,z)
 step-b: stack(x,y)
```

As the domain becomes more "organized" and we improve the planning techniques, efficiency and reuse becomes less important. For instance, when we go from TO to PO, goal ordering is not really important, therefore there is less motivation for macrops. Perhaps there will be new reasons for remembering things, but goal ordering is no longer one of these reasons.

Notice that macrops increase the branching factor and the memory requirements, so we don't really want to introduce macrops unless there is a good reason for doing it.

Macrops vs htn
Using the analogy of Dr Rao, Macrops are like compiled programs which are reused (like subroutines) by other programs. The motivation for HTNs is different; htns are more like the way us humans tend to see things. We decompose tasks into smaller ones, and that provides us with a smaller set of primitive tasks which may or may not be ordered with respect to each other, however, there are much fewer combinations of different orders to be considered (which is referred to as better control information).
Also, components are visible in htn, whereas they are not visible in macrops.

Basic algorithm

. Create the first-level plan with the goal to be achieved

. While a non-primitive plan has not been generated, do:

  - generate a new level of plan by decomposing the decomposable actions and leaving the primitive actions intact

  - criticize the new plan to eliminate bad interactions among actions and take advantage of good interactions (*)

(the hierarchy of plans is often called a procedural net)

(*)

This is one of the things that was "lost" after NOAH. After NOAH, planners didn't use this notion of critics

The planners we have seen so far are "myopic", in the sense that they will often try to find a plan when there isn't one. For example, in the plan below it's no use putting tasks Ti and Tj ad infinitum, because there is no task that will make such plan correct, so might as well quit and give up.

```
              +hf  +he,+hf
     initial   Ti   Tj
              (he) (hf)
```

Some planners used in industry incorporate domain specific knowledge critic; for instance, say if a battery is low, it's no use doing some task, we may use that information and prune that plan (that is, remove it from the list of plans being considered for expansion).

NOAH put too much confidence on the critics, so much so that it never used searching. However, Sacerdoti's threats included the promotion but forgot about the demotion. Today, researchers agree that we should use searching, to formally define the planning process

From rao Fri Apr 9 13:34:32 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
 id AA03859; Fri, 9 Apr 93 13:34:32 MST
Date: Fri, 9 Apr 93 13:34:32 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9304092034.AA03859@parikalpik.eas.asu.edu>
To: plan-class
Subject: filter conditions and loss of completeness
Reply-To: rao@asuvax.asu.edu

Yesterday I was trying to give you an example of how filter conditions may lead to incompleteness and was unable to do so.

Turns out that the right way to construct such an example is to think in terms of an operator being prematurely rejected, there by leading to loss of completeness..

TO see this consider a somewhat fanciful example where you have two goals:

 G1: have-an-airport
 G2: go-to-sfo

Suppose, in this case there is no airport in the initial state, and you decided to work first on G2 and then on G1.

Suppose further that there is an operator that takes you to SFO which has as a filter condition "there be an airport"

So, at this point, since airport isn't present in the init state, you reject this operator.

However, since SNLP doesn't work this goal ever again, that means that this operator will never be seen again...

Now, we you go on to work on G1 which will in fact introduce an airport, and by that time unfortunately, you already lost the operator of using this airport to go to SFO.

This problem wouldn't have been there in STRIPS which will work on both goal orderings.

This problem also will not be there in TWEAK and UA etc., which can work on a goal more than once, as necessary.

Rao

```
From wtsai@enws320.eas.asu.edu  Fri Apr  9 17:49:10 1993
Return-Path: <wtsai@enws320.eas.asu.edu>
Received: from enws320.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA03942; Fri, 9 Apr 93 17:49:10 MST
Received: by enws320.eas.asu.edu (4.1/SMI-4.1)
        id AA13775; Fri, 9 Apr 93 17:42:29 MST
Date: Fri, 9 Apr 93 17:42:29 MST
From: wtsai@enws320.eas.asu.edu (Wan C. Tsai)
Message-Id: <9304100042.AA13775@enws320.eas.asu.edu>
To: plan-class@enws228
```

Notes for the Class of Planning Seminar on Apr. 6, 1993
Written by Wan-Chu Tsai

Agenda:
1. HTN Planning
   * The unresolvable conflict problem
   * The filter conditions (operator afflication problem)
2. Precondition Abstraction
   Next class read ABSTRIPS

---

1. NOAH-like planning:

```
    +------------+
    |            |
    |         +-----+
    |      -> | SG1 | -
    |     /   +-----+  \
    | __ /              \__\
    |    \              /  /
    |     \   +-----+  /
    |      -> | SG2 | -
    |         +-----+
    |            |____________________________________
    |                                                |
    |   OP1   +---------------------------+          |
    +-------->|     +-----+               |          |
              |   ->| A11 |-              |          |
              |  /  +-----+ \    +-----+  |          |
             -+-<            >-->| A13 |--+-         |
            / |  \  +-----+ /    +-----+  | \        |
           /  |   ->| A12 |-              |  \       |
          /   |     +-----+               |   \          |
         /    +---------------------------+    \         |
      __/                                       \____\   |
        \                                       /    /   |
         \    +----------------------+         /         |
          \   |   +-----+    +-----+ |        /          |
           ---+-->| A21 |--->| A22 |--+-------    OP2    |
              |   +-----+    +-----+ |<-----------------+
              +----------------------+
```

. SG1 is decomposed by applying OP1
. SG2 is decomposed by applying OP2

* Problems with HTN:
  1. Control information required is not totally doamin independent.
  2. Sometimes it is hard to decide to which abstract level the decomposition process should stop.
  3. HTN is too flexible that something that is guaranteed in SNLP no longer is guarantee, [such as the use of primitives?]

* We assume here that HTN only requires control information that is domain independent, and this information can be given by operator description.

2. Basic algorithm for HTN:

```
I : Initial state
G (G1, G2, G3) : Goal state

(There are two types of tasks : goal task and action task.)

d <-- Initial Task Net    [ Achieve(Goal) ]

Let Que <-- {d}

Repeat

  (0) If Que empty, fail.

  (1) Pick a task net d from Que
      /* A choice point here, can backtrack if necessary. */

  (2) If every task in d is primitive or phantomization, and
         d is consistent (by MTC)
      Then exit with success, return d.

  (3) Pick t in d, s.t. t is non-primitive.
      /* A cut, no backtrack allowed here. */
      (Precondition abstract can be used here, if have it.)

  (4) Let M be the set of methods which can achieve t.
      Pick m in M
      /* Another choice point. */

  (5) Reduce t with m to R(t,m).
      (i.e. extend to lower level.)

  (6) Merge (d - t) with R(t,m), i.e. d' = Merge((d-t),R(t,m)).
      When doing merge, accessor and contributor need to be taken care of.
      m can partially support this job because it provides ordering
      information.  But, something must be aware of:
```

(a) It is possible that the effect given by a parent node (high level) is not given by any single node of its child nodes (low level). For example, neither the operator 'Travel from Pheonix to L.A.' nor the operator 'Travel from L.A. to Pheonix' can give the effect 'Round trip from Pheonix to L.A.'. But the parent of these two operators can give it.
(b) It is also possible that more than one child nodes give the effect. Need to choose which one to be the contributor.

NONLIN takes very syntactic approach to solve these problems:
let the first action be the accessor, and the final action be the contributor.
If there is no unique first action or final action, then add a dummy action. The causal link of reducing t will become the following:

```
     p        will be reduced to        p
<t1 ---> t>  ===================>  <t1 ---> tbeg>
    e         will be reduced to           e
<t ---> t2>  ===================>  <tfinal ---> t2>
```

tbeg and tfinal are dummy actions.

(7) CRITICIZE/deal with interaction.
There may be some effects in lower level that are not mentioned in higher level. The interaction caused by these effects needs to be handled carefully.
For example,

Let d be the following:

```
      +e1
    +----+     +----+
 --->| t1 |---->| t2 |
    +----+     +----+
       |        (e1)
       |          ^
       |    +e    |
       |  +---+   |
       +->| t |---+
          +---+
                           e1
  The causal link :  < t1 ----> t2>
```

Let d' = Merge((d-t), R(t,m)) be the following:

```
      +e1
    +----+            +----+
 --->| t1 |----------->| t2 |
    +----+            +----+
       |               (e1)
       |                ^
       |                |
       |    +----+   +----+
       |  +-| t' |-->| t" |
       |  | +----+   +----+
       +->|             ^
          |    -e1      |
          |  +-----+    |
          +->| t"' |----+
             +-----+
```

A problem here: t"' deletes e1, this is not described in higher level.

This algorithm separates children generation from MTC checking.

```
The output of the critics : Cdp(d') --> d1', d2'
                                        d1' : t"' <  t1
                                        d2' : t2  <  t"'
So, to deal with interaction:

  Let set <-- {d}
  For each cnt in CRITICS   (This could be quite complicated, depending
                             upon the types of critics.)
      For each Tset in d
          Remove Tset from set
          Add C(Tset) into set
```

(8) Que <-- Que + set

/* End of Repeat */

* Critics are not restricted in HTN.

* To improve the efficiency of critics, we need to change the critics.

* SNLP never merge actions to make optimal plan.

3. In SNLP, if we want to make something true, we can use exist action to do so.
In HTN, for example,

```
        +-+
   I :  |B|      G : Clear(B)
        +-+
        |A|
       ------
```

We want to have a plan that do nothing, so we need something like SNLP's simple establishment.

Phantomization : a task that is not executable, i.e. a task that has already been achieved.

A[C] : make C true by phantomization, this is a choice point.

4. Comparison:

* NONLIN does not keep full search path, it does backtrack by undoing.
  SNLP keeps complete search path.

* NONLIN does not have systematicity.
  Within the same branch, using phantomization may work on the same goal twice.
  SNLP concentrates on systematicity.

* HTN works on fragment of plan, if something goes wrong, instead of giving up the whole plan, it makes some change to solve the problem.

* Critics can be retractive critics.

* In refinement planning, planner never does retraction, the only thing to be done is backtrack.
  Transformational planning allows retraction.

```
      If full retraction is allowed, then no backtrack is required.

    * NONLIN is transformational planning, but most time it does addition.
      It adds backtrack to NOAH, but it tries to avoid backtrack.



From gopi@enuxha.eas.asu.edu Mon Mar 15 17:36:52 1993
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
        [nil nil nil nil nil nil nil "Bulusu Gopi Kumar" "gopi@enuxha.eas.asu.ed
u" nil nil nil "^From:" nil nil nil])
Return-Path: <gopi@enuxha.eas.asu.edu>
Received: from enuxha.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA03771; Mon, 15 Mar 93 17:36:51 MST
Received: by enuxha.eas.asu.edu id AA05059
  (5.65c/IDA-1.4.4 for rao@parikalpik.eas.asu.edu); Mon, 15 Mar 1993 17:36:52 -0
700
Message-Id: <199303160036.AA05059@enuxha.eas.asu.edu>
X-Mailer: ELM [version 2.3 PL11]
From: Bulusu Gopi Kumar <gopi@enuxha.eas.asu.edu>
To: rao@parikalpik.eas.asu.edu
Subject: Distribution properties . . . (fwd)
Date: Mon, 15 Mar 93 17:36:50 MST

Forwarded message:
> From gopi Thu Mar 11 16:21:28 1993
> Date: Thu, 11 Mar 1993 16:21:25 -0700
> From: Bulusu Gopi Kumar <gopi>
> Message-Id: <199303112321.AA19057@enuxha.eas.asu.edu>
> To: rao@parikalpik.eas.asu.edu
> Subject: Distribution properties . . .
> Cc: gopi@enuxha.eas.asu.edu
>
>
>     This is regarding the distribution properties that regression
>     operator has when there are no disjunctive effects ...
>
>
>     I feel that presence of disjunctive effects should not affect
>     the distributive properties really with a small modification
>     to the idea of distributive property
>
>     op ( a V b) = op (a) V op (b)
>
>     Here in some sense a and b are the dimensions of the state
>     and the operator can be computed for each dimension separately
>     and or'ed to get the lhs
>
>     If redefine the dimensions to include (on an operator basis)
>     all the disjunctive effects the operator can have then we
>     can say that
>
>     op (a V b) = disjunction of op on values in all dimensions
>             which affect a  V B
>
>     We can further define a V B to be one of the dimensions
>
>     then op ( A V B) = op (A) V op (B) V op(A V B)
            This is new                  this is for only the dimension
            defn of dimension            AVB
>
>     each of these can be precomputed for all operators !
>
>     Thus all the original properties hold
>
>     The only difference is that the cost of finding all dimensions
>     affected by (A V B) would have been linear o(n) in the number of atomic
>     formulae in the original case, and would be o(n * k * m) now where k
>     is the maximum number of effects an operator can have and m is the
>     maximum number of dijuncts in any effect which is still linear
>
>     Is there something wrong with the above argument ?
>
>     (it seems to be intuitively true)
>
>     with regards
>
>     gopi
>
>


From gopi@enuxha.eas.asu.edu Mon Mar 22 18:59:09 1993
Return-Path: <gopi@enuxha.eas.asu.edu>
Received: from enuxha.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA12316; Mon, 22 Mar 93 18:59:08 MST
Received: by enuxha.eas.asu.edu id AA10291
  (5.65c/IDA-1.4.4 for rao@parikalpik.EAS.ASU.EDU); Mon, 22 Mar 1993 18:59:37 -0
700
Message-Id: <199303230159.AA10291@enuxha.eas.asu.edu>
X-Mailer: ELM [version 2.3 PL11]
Status: RO
From: Bulusu Gopi Kumar <gopi@enuxha.eas.asu.edu>
To: rao@enuxha.eas.asu.edu (Subbarao Kambhampati)
Subject: NONLIN, use-only-for-query
Date: Mon, 22 Mar 93 18:59:35 MST


        I feel that use-only-for-query conditions should be in such
        a way that they do not affect the effects of an operator.

        In operators.lisp (on ?x ?Z) is taken as an use-only-for-query
        condition, which seems to be wrong since, when this binding is
        changed later on, anyone taking (clear ?z) (for some instance
        of z) will no longer get that effect, and I don't think NONLIN
        takes care of that.

        With regards

        gopi

From rao Thu Apr  1 19:06:01 1993
Status: RO
X-VM-v5-Data: ([nil nil nil t nil nil nil t nil]
        ["1789" "Thu" "1" "April" "93" "19:06:00" "MST" "Subbarao Kambhampati" "
rao " nil "54" "HTN class insights" "^From:" nil nil "4"])
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA23430; Thu, 1 Apr 93 19:06:00 MST
Message-Id: <9304020206.AA23430@parikalpik.eas.asu.edu>
Reply-To: rao@asuvax.asu.edu
```

```
From: rao (Subbarao Kambhampati)
To: rao
Subject: HTN class insights
Date: Thu, 1 Apr 93 19:06:00 MST

People don't consider starting with larger piece of knowledge to be
a cheating...

1. Critics being lost in the shuffle.
    Noah wanted to planning with big pieces, and thus came up with
     the idea of HTN planning, and partial order planning.

NOAH had a bunch of critics-- none of them were sound and
     complete

    In the process of making deleted precondition interactions
    systematic, nonlin threw out all the other critics.

    Tweak threw out HTN and plan fragment stuff, and formalized
    partial ordering planning with operators

    SNLP formalized search

    But, all of planning is still very myopic. We do need some
    macro-critics


2. For noah, partial ordering was a real necessity
     He wanted to make large plans by starting with
      Large planning fragments.

     Once you start with large plan fragments, partial order planning
     is almost inevitable.

3. Reuse: Why macro task reduction schemata don't make sense?

4. subroutine analogy-- the lower level subroutines DO change the
   higher level gloabl variables sometimes.... although that is not
   considered a good programming practice.

5. comparison iwth Macrops: Macrops have input and output--but no
   hierarchical structure.

   By the time youcome to HTN planning, macrops becoming less useful
   [Point this out in learning paper]

6. Put this stuff in learning paper-- in a way, partial ordering
   planning provides a more useful substrate for doing learning, since
   you want to put plan fragments together.

7. Put Minton's comments about how it is difficult to use more difficult
   planning strategies vis a vis planning

8. Philosophize in the end... the things that Knoblock can automate
   are exactly the things that aren't useful for partial order
   planning



From rao Tue Apr  6 19:06:13 1993
Status: RO
X-VM-v5-Data: ([nil nil nil t nil nil nil nil nil]
        ["1896" "Tue" "6" "April" "93" "19:06:12" "MST" "Subbarao Kambhampati" "
rao " nil "58" "htn planning ideas" "^From:" nil nil "4"])
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA29885; Tue, 6 Apr 93 19:06:12 MST
Message-Id: <9304070206.AA29885@parikalpik.eas.asu.edu>
Reply-To: rao@asuvax.asu.edu
From: rao (Subbarao Kambhampati)
To: rao
Cc: rao
Subject: htn planning ideas
Date: Tue, 6 Apr 93 19:06:12 MST


1. NONLIN not only allows contributor change for phantom tasks [in
   that it uses causal links merely as a guidance], it also allows
   actual dephantomization, which is akin to saying that I am undoing
   my method choice. This is retraction!!

   Once you allow retraction, you might as well also allow dependency
   directed backtracking-- this is what I was asking Gopi to do.


   In the literature, people always talked about refinement planners
   and transformational planners. Tranformational planners don't do
   backtracking-- they do retraction. Refinement planners don't do
   retraction.

   Nonlin doesn't-- it uses refinement to add, and allows a bit of
   retraction through dephantomization.

   DDB--does it or does it not actually allow retraction?



2. The differing ideas in terms of how much you belive in your
   original choice.

   SNLP which does search at a lower level and doesn't quite believe
   is best off doign systematic search.

   While NONLIN and HTN planning which believes that the plan
   fragments are largely interaction free and planning is largely easy
   to do, is better off doing dependency directed backtracking.

   In this sense, doing retraction and talking about systematicity
   seems quite stupid


3. Criticism can be done all at one time, or as and when you feel. As
   long as the final check is that the plan is correct, you can avoid
   criticising in the beginning

4. Critiques can be retractional: You can think of dealing with
   deleted precondition interaction by removing plan fragments, adding
   white-knights or backtracking (each of which is progressively more
   systematic in some sense)

5. merging techniques
```

6. backtracking on tasks..

7. The algorithm-- why should it be that clean to begin with? Is it just to satisfy our quest for theoretical beauty?

Send mail to Drew McDermott

Rao
[Apr 6, 1993]

From ihrig@enws318.eas.asu.edu Wed Apr 14 15:38:02 1993
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA08500; Wed, 14 Apr 93 15:38:02 MST
Received: from enws322.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA03485; Wed, 14 Apr 93 15:34:56 MST
Date: Wed, 14 Apr 93 15:34:56 MST
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
Message-Id: <9304142234.AA03485@enws318.eas.asu.edu>
To: plan-class@parikalpik.eas.asu.edu

Class notes April 8, 1993 by Laurie H. Ihrig
Agenda:
  filter conditions
  unresolvable conflicts

The argument has been put forth that planning research has formalized only one aspect of planning (MTC), but practical planners provide more than this.

Practical planners must modularize planning knowledge, provide control information.

Trend is toward user friendly:
NOAH had SOUP code, no well established syntax. This was thrown out and replaced by user-provided templates.

Example of a task-reduction schema:

```
A[Clear ?X]

reduces to

A[Clear ?Y]\
            puton(?Y,?Z)
           /
A[Clear ?Z]

expansion:
    1. goal (Clear ?Y)
    2. goal (Clear ?Z)
    3. action (puton ?Y, ?Z)

conditions:
  :use-when (on ?Y ?X) at 3
  :precond  (Clear ?Y) at 3
  :precond  (Clear ?Z) at 3

effects:
  :assert (Clear ?X) at 3
  :delete (Clear ?Z) at 3
```

Use-when is a filter condition, planner won't try to achieve it.

For example, having an airport in your home town might be one precondition of flying, but it is nonachievable since there is no way to build an airport if one does not exist. On the other hand, on(?Y, ?X) is achievable. It is just not advisable to do so.

The problem with filter conditions such as on(?Y,?X) is that by the time you come to work on a schema, Y is no longer on X. This is the problem with using filter conditions in PO planning. It is not a problem for state-based planners since once you add an action in order to make a goal true, the plan is extended from there in one direction only, and the filter condition that is needed at that point in the plan will always hold at that point where it is needed. In plan-based planning, actions may be added anywhere in the plan. These actions may delete the filter condition.

Filter conditions aren't really necessary if you are using a best-first search, since establishing a condition by using an operator would cause the plan to be ranked as worse that a plan that takes the condition from the initial state.

If you don't use filter conditions, you might have a goal loop.

Example:
Having an open door is a filter condition for getting into the room that has the keys. If having an open door is not considered to be a filter condition, then you might get a loop such as:
enter (room), open(door), get(keys), enter(room), open(door), get(keys) etc.

SNLP would have more of a problem than other PO planners in using filter conditions, since it is harder in SNLP to prune paths without sacrificing completeness.

Example: (Laurie's)
  Suppose you have two operators, one to PICKUP a block (deleting clear ?X) and another to PUTDOWN a block on the table (adding clear ?X). It seems reasonable that you should not attempt to make a block X clear by first picking it up and then putting it on the table since performing the pick up action requires the block to be clear in the first place. In other words, holding (?X) is a filter condition for achieving clear(?X) by using the PUTDOWN action (just as open(door) is a filter condition for getting into the room that holds the keys).
  However, suppose we define a problem where the initial condition has block A on top of another block, and the goal is (clear(A) and on(A,table)). Pruning the path that attains clear (A) by the PUTDOWN action will result in SNLP failing to find a solution. The only path left commits to establishing clear (A) by a link from the initial start step (since A is clear in the initial state) but this link will be threatened by the PICKUP action that is added later in achieving the second goal. This problem pertains to SNLP only since it is related to the way SNLP attains systematicity by committing to causal links.

Wilkins provides a contorted explanation of the problem of using filter conditions in PO planning. He views it as a problem of heirarchical planning, and calls the problem heirarchical promiscuity. It occurs at a certain point in the planning process, where for example, we have a plan S1---S2, and expand S2 assuming that a filter condition is true, then later expand S1 to include a step that deletes this condition. We can solve this problem by making sure that S1 is expanded before S2, but then we are doing STRIPS planning in the guise of SIPE. This is not really a problem for heirarchical planning only. It is dubious to use filter conditions in any planner that is searching in the space of plans.

HTN planning is not well understood. HTN's are very flexible: there is very little pruning. We would like to prune plans that have unresolvable conflicts, for example:

```
               -Q
                t1
               (P)
          <           >
               -P
               t2
               (Q)
```

but in HTN planning this conflict could be resolved at a lower level, for the plan may be expanded to:

```
                         -Q
            t1'    ---    t1"
            (P)
       <                      >
                          -P
            t2'   ---     t2"
            (Q)
```

The interaction can now be resolved by imposing an ordering such as t1' t2' t1" t2".

You therefore could not prune this path without losing completeness. (You can always set up the domain so that this is the only correct path).

Qiang Yang tried to come up with sufficient and necessary conditions for pruning paths in HTN planning. One condition he proposed is that when a step is expanded to several steps, there is one of these steps that is considered to be the main action, and this step holds all preconditions and effects. This is, however, very restrictive.

The End

From rao Thu Apr 15 11:29:13 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA09553; Thu, 15 Apr 93 11:29:13 MST
Date: Thu, 15 Apr 93 11:29:13 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9304151829.AA09553@parikalpik.eas.asu.edu>
To: plan-class
Subject: Readings--
Reply-To: rao@asuvax.asu.edu

When you read FORBIN, do also read the chapter on planning in Charniak and McDermott's text book (chapter 9). It will explain the FORBIN planning philosophy better

rao

From suresh@enws318.eas.asu.edu Sun Apr 18 14:17:34 1993
Return-Path: <suresh@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA11578; Sun, 18 Apr 93 14:17:34 MST
Received: from enws322.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA05677; Sun, 18 Apr 93 14:14:22 MST
Date: Sun, 18 Apr 93 14:14:22 MST
From: suresh@enws318.eas.asu.edu (Katukam Suresh)
Message-Id: <9304182114.AA05677@enws318.eas.asu.edu>
To: plan-class@parikalpik

Class Notes for 15th April

PRECONDITION ABSTRACTION :

Approaches :
1) ABSTRIPS
2) ALPINE
3) PABLO
4) DISTRIBUTION

What is an abstraction?

Dividing the problem space into different levels where important goals are solved at higher level and less important goals are solved at lower levels. There should be a downward refinement which will lead to final plan without undoing the steps that have been done at higher level. i.e. when working on one level to achieve all the goals of that level, it shouldn't undo any of the goals acheived at its higher level.

The importance of a goal can be
1) The critcality value given by the user.
2) The interactions that the goal is involved with.
3) length of the plan to acheive it.
4) no. of ways of achieving a goal.

1) The criticality value is given by the user. These criticality values depends upon the user intution. (Generally the difficulty in achieving a goal is taken into consideration.) This approach is used in ABSTRIPS.

2) Goals are given equal or higher importance to the preconditions of the operator used to achieve it. If the goals are strongly connected in the graph (explained below) then all those goals are given equal importance. This approach is used in ALPINE. It divides the given problem space into different levels in which the MONOTONICITY property is applicable. According to this, it seperates out those features (goals) of the problem that can be

```
   solved and held invariant while remaining parts of the problems are
   solved.

3) The length of plan to achieve it can be directly taken as its
   importance level. But if there are many ways of acheiving a goal
   with different plan lengths, then maximum plan length should be
   (may be) taken as its importance level. This approach is used in
   PABLO.

4) The number of ways of achieving a goal is taken as the importance
   of that goal.  If there are many of ways of acheiving a goal, then
   it is less important. In other words, the number of binding
   conflicts can be taken into consideration.  e.g :

        Consider the goals are (OBJECT X) and (STEEL X)
         Say there are 100 objects and one steel object. Then there are 100
         ways of acheiveing (OBJECT X) and only one way of acheiving (STEEL X).
         So (STEEL X) should be given higher importance ( i.e solved first ) tha
n
         (OBJECT X)

    This approach is known as distribution.


STRIPS : In STRIPS, steps are added only at the end of the existing
        plan.  By doing this, the step which is added may undo the goal that
         is already achieved. That is, it may not be possible to add a
         step without undoing the previously acheived goal. OR the
         goal which has been deleted has to be achieved again.

         To avoid this, ABSTRIPS seperates the goals into different
         levels where the important goals are achieved first.

         e.g. Consider the goals : (ON A B), (ON B C)

         Say, (ON A B) is achieved first. While working on (ON B C),
         it has to undo (ON A B) in order to achieve (CLEAR B) for (ON
         B C).  or it will backtrack and select different goal order
         (i.e. (ON B C), (ON A B) )

         In STRIPS, the planning order is strongly related to
         execution order. i.e goal order affects the performance of
         the planner. If wrong goal order is chosen, palnner has to
         backtrack to select the correct goal order OR it has to
         reachieve the previously undone goal.

         If all goals are independent of each other, then goal order
         doesn't matter while planning. Since, most of the goals
         interfere with other goals, the order directly affects the
         performance of the planner.


ABSTRIPS :

        In ABSTRIPS, the operator preconditions are given criticality values.
        e.g.


        (UNSTACK X Y) :
                        1          2       2
          precond : (arm-empty)(clear x)(on x y)
          add     : (clear y)
          dele    : (arm-empty) (clear x)

        The numbers given above the precondtions literals are the
        critical values of the precondition. i.e. critical value of
        (on x y) is 2.

        While solving the problem, the goals with higher critical
        values are solved first before lower level critical value goals are
        solved.

        e.g : Consider the same goals as above : (ON A B) (ON B C)
              These two goals are given same critical values. these goals are so
lved
       first before solving other the preconditions of goals.

                                 +-----------------------+
      Planning :                 |   (ON A B) (ON B C)   |
                                 +-----------------------+
                                     /         \

              Level 2:    (STACK A B)     (STACK B C)
                                                       These two
              interefere with each other, these will be ordered. (OR
              you can say that, it works on (ON B C) first, then (ON A
              B) by taking different goal order.

                     +--------------------------------+
                     |  (STACK B C)      (STACK A B) |
                     +--------------------------------+

              At every level, all interactions are taken care of by
              working on different goal orders till all interactions
              are removed.

              Now, preconditions of (STACK B C) , (STACK A B) are
              worked out And the steps which are used to achieve the
              preconditions of (STACK B C) are inserted before (STACK
              B C). And the steps which are user to achieve the
              preconditions of (STACK A B) are inserted before (STACK
              A B) after (STACK B C).

      IMPORTANT : This kind of planning is done in partial ordering
      planning where
              steps can be inserted any where in the plan.

              But in above planning, plan to achive preconditions of
              (STACK B C) is STRIPS planning.  i.e steps are are added
              only at the end of the plan untill (STACK B C).  Then
              the plan to achieve (STACK A B) is inserted after the
              plan to achieve (STACK A B).


              Therefore, above kind of abstraction is just a heuristic
              in partial order planning. This heuristic helps in
              picking up goals from the open list to be worked on.
              i.e. higher criticality value goals are worked out first
              before lower criticality value goals are worked out.
```

```
TWEAK : In this, steps can be inserted any where in the planning. So
        above kind of abstraction is not a abstraction here. It is just a
          heuristic. But TWEAK may have to backtrack when a protection
          is violated. i.e. a step may come in between two steps and
          it may undo the effect of previous step. There are no
          protection intervals are maintained.

ABTWEAK : In this, protection intervals are maintained. i.e. no step
          can be inserted between two other steps if it deletes the
          effect of the previous step. It divides the probelm into
          different leves where the lower level steps do not interfere
          (undo the effect of higher level steps)) with higher level
          steps.

          IMPORTANT : But in SNLP, protection intervals are
          maintained. So this kind of abstraction is also of no use.

          In SNLP, Goals should be divided into different levels based
          on their importance but not on the interactions of the
          goals. e.g. hand empty can be achieved in many ways. so this
          should be solved at the end. Abstraction based on the length
          of the plan to achieve may be a good abstraction in SNLP.
          Probably, Abstraction based on the distribution may also be
          a good abstraction.


To be disscussed : PABLO and DISTIBUTION.

From rao  Mon Apr 19 11:45:51 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA13188; Mon, 19 Apr 93 11:45:51 MST
Date: Mon, 19 Apr 93 11:45:51 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9304191845.AA13188@parikalpik.eas.asu.edu>
To: plan-class
Subject: missing notes
Reply-To: rao@asuvax.asu.edu

As of now, the notes for the following two classes are not yet sent to
plan-class:

13th April (last Tuesday): Discussion of HTN planning, Critics, Forbin,
    real time planning etc.

Last class before spring break: Wrap up of ADL, discussion of ADL and
 partial order planning, Zeno

Whoever is in charge of these notes, please let me know as to their status

Rao

From suresh@enws318.eas.asu.edu  Mon Apr 19 12:27:31 1993
Return-Path: <suresh@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA13231; Mon, 19 Apr 93 12:27:31 MST
Received: from enws322.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA05794; Mon, 19 Apr 93 12:24:19 MST
Date: Mon, 19 Apr 93 12:24:19 MST
From: suresh@enws318.eas.asu.edu (Katukam Suresh)
Message-Id: <9304191924.AA05794@enws318.eas.asu.edu>
To: plan-class@parikalpik

Class Notes for 13th April


Two ways of pruning search space we have discussed so far:

1. Order consistency   Complexity is O(n*3)
2. Safety (Resolving threats)  This is a Constraint Satisfaction Problem
      and is solved in exponential time

There is also a third way

3. Window Check  (in the case of time dependent problems)
                     which is a polynomial process

used by FORBIN to strengthen consistency check

(We can see NOAH-->FORBIN as a scale of increasing strength of
consistency checking and decreasing dependence on search.  NOAH
didn't have search (in the sense of backtracking on abstraction
but needed it).  We should look at FORBIN with as much
care as SNLP.  It is a more practical planner, allowing deadlines,
resources.  It also provides a different treatment of continuously changing
quantities and allows deduction of conditions separate from action
effects.  TWEAK also did  the latter for STRIPS action representation by
adding axioms that are separate from actions.  In FORBIN clear(x)
is 'clipped' by on(y,x).  ZENO also deals with full temporal info).

Answering question on time dependent planning, Rao explains:

total time =planning time + execution time

Dean uses deliberation scheduling to determine how long to plan.
In this case we have

total time = scheduling time + planning time + execution time.

The basic assumption of Dean's anytime algorithm is that the utility
if the solution found improves monotonically with time.  If this
can be approximated by a piecewise linear function, then it is
polynomial process to schedule--but we must be able to calculate
the utility of a computation.

Example:
Suppose you have two computations  AlphaJ and DeltalJ.  AlphaJ
results in three solutions Alpha1, Alpha2, Alpha3.  Then the
utility of doing AlphaJ is the maximum of the utility of the 3, ie.

U([AlphaJ])=max(U(Alpha1), U(Alpha2), U(Alpha3))

but how in the beginning do you know the utility of AlphaJ-- this
is a problem.


From AZEDC@ACVAX.INRE.ASU.EDU  Wed Apr 21 00:13:50 1993
Return-Path: <AZEDC@ACVAX.INRE.ASU.EDU>
Received: from ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA15506; Wed, 21 Apr 93 00:13:50 MST
Received: from ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF #2382 ) id
 <01GX94G2AG34005A4A@ACVAX.INRE.ASU.EDU>; Wed, 21 Apr 1993 00:10:33 MST
```

Date: 21 Apr 1993 00:10:32 -0700 (MST)
From: AZEDC@ACVAX.INRE.ASU.EDU
Subject: class notes, April 20
To: plan-class@enws228.eas.asu.edu
Message-Id: <01GX94G2AG36005A4A@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"plan-class@enws228.eas.asu.edu"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT

Class notes of April 20, by Eric

Agenda:
1) precondition abstraction methods: ABSTRIPS, ALPINE, PABLO
2) HTN planning continued

ABSTRIPS
The idea is to put different preconditions at different abstraction levels.

ABSTRIPS is not concerned with the "length" of the plan; there are no critics that tell the planner to remove plans from consideration (ABSTRIPS did mention the idea of plan length, but the idea is not explained neither has been implemented).

ABSTRIPS is not concerned with abstraction hierarchy generation; it presupposes someone will provide the abstraction for us.

Since STRIPS can only concatenate (that is, it cannot interleave) steps, the idea of abstraction at each level makes a lot of difference for STRIPS, but would not be so interesting with SNLP.

```
(level x)      O1   state2    O2   state3    O3        The pre-
                                                       conditions
                                                       of step O2 are
                                                       true in state2
(level x-1)         O1a  O1b  ...
```

At level x-1 we find a plan that will make O1 true; at the lowest level of abstraction, all the preconditions will be visible; at a level i, the pre-conditions i, i+1, ... n will be visible.
Al level x we know that precondition x is true, but at level x-1 we don't really know whether precondition x-1 is true. So, we need to check again.
When the precondition is indeed true, we obtain a new state, resulting from the step at the level x, plus the steps that have been derived from the abstraction.

If the state at level x-1 subsumes the state at level x, we can proceed because we are not undoing things that were done at the previous levels; otherwise, we need to backtrack.

ALPINE (Knoblock)
In this planner, when working on level x-1 we never undo things that were done at the previous levels.
This property is called ORDERED MONOTONICITY, and is a property of the abstraction hierarchy. Since the operators in the lower levels do not undo things, we don't have to worry about backtracking (it comes free).

We remember that a few classes ago it was pointed out that hierarchies which have downward refinement property are desirable.
Does ordered monotonicity guarantee downward refinement property (very few hierarchies have that!) ? NO.
Since the downward refinement property is a hard problem, we don't expect to find many of those.
To illustrate, consider the example: at a higher level we have O1 O2 O3 When we refine, we find out that O1 interacts with O3, so we cannot achieve either of them without removing one, and thus we need to backtrack and consider other alternatives.

If we had the guarantee that there is no need to undo things, we would have to worry too much about backtracking (ordered monotonicity reduces the complexity). Nevertheless, this only makes sense for STRIPS.
An additional complication is introduced in STRIPS: when there is inconsistency, STRIPS cannot tell if the problem was due to a wrong high level abstraction that was given or whether it was the refinement that introduced the problem. By that time it is too late already (we already committed to bindings, orderings, etc.), and the only thing STRIPS can do is backtrack. Remember that STRIPS had to pick one particular goal order (which in this case turned out to be wrong), so now another order needs to be attempted. Of course, undoing everything is a very costly operation.

If we had the guarantee of ordered monotonicity, we wouldn't have to undo the refinement of the operator.

In STRIPS, steps are ordered with respect to each other; SNLP postpones the ordering, that is, steps are kept unordered as much as possible. Because of that, there may be one such ordering that, if we were using SNLP, would make the inconsistency go away; for that reason, SNLP would be better than STRIPS.

State based planners with protection violations (like INTERPLAN) can detect violations; note that they change drastically the nature of the planner. In any event, all this kind of planner can do is backtrack. It seems that SNLP would do better than these planners, because as we have just noted, refinement is better than backtracking.

The goal order problem that existed in STRIPS now becomes the sub-goal order problem in ABSTRIPS. Dependency directed backtracking can be introduced at the expense of increased planner complexity.

Idea: put preconditions that interact with each other at the same level (i.e. clusters)

```
example:  C1  will undo C2, C1', C2'
          C2  will undo C1, C1', C2'
          C1' will undo C2'
          C2' will undo C1'
```
the clusters are (C1 and C2) and (C1' and C2')

We can build a graph with operators affecting one another, and draw a strongly connected graph (a set of vertices connected through arcs); this graph detects potential interactions. The algorithm computes this in polynomial time.

Example	(preconds)	operator	effects
	C1'	O1	C1,C2,~C2'
	nil	O1'	C1',~C2'
	nil	O2'	C2',~C1'

```
          C2'             O2        C2,~C1,~C1'

The graph is:       -->  C1
                    :  :      :         :
                       v      v         v
                    C2 ->     C2' <-    C1'
                    :             ->    ^
                    --------------------:
```

```
The strongly connected graphs are:    C1 -> C2
                                         <-

and                             C2'-> C1'
                                   <-

The next step is to separate in clusters          C1 C2
                                                 : : : :
                                                 v v v v
                                                  C1' C2'
```

Now we can do a topological sort of the figure above, which gives a linear ordering; the top of the resulting list gets a high level of criticality, the end gets a low level.

Any abstraction that uses this has the ordered monotonicity property (this is a sufficient but not necessary condition). There are less constrained conditions; if we put too many links, the abstraction hierarchy collapses and we only get one strongly connected graph. To avoid that we introduce relaxations:

- use problem specific instead of domain specific relaxations; for example, for a given domain we have the graph

```
                          C1 <- C3
                          :     ^
                          v     :
                          C1' --:
```

  if we are only interested in C1 and C2 and need not consider C3, this cycle does not collapse (it would otherwise), and now it satisfies the ordered monotonicity property FOR THAT PROBLEM

- avoid putting links to a delete list
- filters/ primary effects (avoid picking operators for the wrong reasons)
- if a condition is not true, it cannot be made true anyway, so don't pick it

Abstractions like the above may not capture what is important or not (like in the example of handempty and handful; the OM builds the graph and there may not be a relation between the graph and the importance we know exists for he and hf)

PABLO
Rather than thinking about these links, we are interested in the length of the plan

Predicate relaxation

$$P^{o}_{rel} = P$$

$$P^{n}_{rel} = P^{n-1}_{rel} \text{ OR } [\text{ for all } i \text{ REGRESSION } (P^{n-1}_{rel}, O_i)]$$

In other words, if some predicate is true, either is was already true, or it becomes so after regression. The idea is to this process for all predicates; if a predicate becomes true, say, on the 2nd level and on the 10th level, this gives an indication that the shortest plan will make it true on 2nd level.

This idea is closer to what ABSTRIPS had intended. Also note that we may associate a distribution, that is, we may say that 90% of the times the predicate is true on the 2nd level, and 10% of the times it is true on the 10th level, in which case we may or may not decide that 90% is good enough and thus we accept that probability and use it for the shortest plan.

There is a problem with that approach when a tautology is achieved. This will be discussed next class.

From rao Thu Apr 22 09:59:12 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
id AA16501; Thu, 22 Apr 93 09:59:12 MST
Date: Thu, 22 Apr 93 09:59:12 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9304221659.AA16501@parikalpik.eas.asu.edu>
To: plan-class
Subject: forwarded message from Drew McDermott
Reply-To: rao@asuvax.asu.edu

The following message contains some thoughts on HTN planning vis a vis SNLP type planning, and comments on it by Drew McDermott and Austin Tate.

------- Start of forwarded message -------
From: mcdermott-drew@CS.YALE.EDU (Drew McDermott)
To: rao@asuvax.asu.edu
Cc: mcdermott@CS.YALE.EDU
Subject: thoughts on hierachical planning, critics, transformational planning et c.
Date: Thu, 22 Apr 1993 12:43:13 -0400

Date: Fri, 16 Apr 93 01:01:17 MST
From: rao@parikalpik.eas.asu.edu (Subbarao Kambhampati)
Reply-To: rao@asuvax.asu.edu

Drew:

Recently, while "teaching" HTN planning in my planning seminar, I had an opportunity to think more about the discussions on the importance of hieararchical planning, critics etc at the symposium. The following are some thoughts on the HTN planning vs. SNLP type planning which I am sending you in hopes of eliciting comments from you. Most of these don't worry so much about the utility of using task reduction schemas, but rather concentrate on the type of search philosophies looked at in HTN type planning vs. SNLP type planning (I think that as far as task-reduction schemas is concerned, the explanation that they are part of the problem specification, is good enough).[**should also talk about improved goal language]

It seems to me that there are differing meta-assumptions behind hierarchical planning and SNLP/tweak type planning that are not generally acknowledged, which in turn seem to lead to misunderstandings regarding the utility of various features. A lot of the confusions can be traced back to the shift from STRIPS to NOAH which can be interpreted in two different ways.

The current SNLP/TWEAK view of the shift from STRIPS to NOAH has been that NOAH wanted to avoid over-commitment to ordering decisions of STRIPS, and search in the space of plans, and specifically in the space of equivalence classes of ground operator sequences. There is however another compelling interpretation of that shift that seems little acknowledged-- NOAH wanted to splice together large, relatively stable plan fragments, and once you decide to use stored plans and splice them, you may as well have the flexibility of interleaving them; rather than just concatenating them (we have a paper this time in AAAI on the utility of PO planning in reuse, which demonstrates this empirically).

I believe that this second explanation of the shift from NOAH, if taken seriously, will explain the differences between the view of planning taken by the great hierarchical planners, and the view of planning sponsored by SNLP/TWEAK formalizations.

In particular, I think most of the great hierarchical planners (starting with Noah and ending in FORBIN) have made an assumption that the plan library contains plan fragments that are relatively stable and can be put together with relatively little worry about interactions. Apart from providing a rationale for partial order planning in terms of ease of splicing, this meta-assumption also explains why they almost always went for a deliberative depthfirst search regime -- attempting to avoid search as much as possible by spending more and more time on critiquing plans in a variety of ways, including look-ahead and projection.

Here’s a passage from the Conclusions of my "Regression Planning" paper that expresses the same point (substitute "HTN" for "nonlinear" and "SNLP" for "linear"):

"There is not much interest in linear planners these days, but it is hard to tell from the published literature why this should be the case. A glance at the research in planning will show that much of it is still concerned with worlds as simple as the blocks world, but that nonlinear planning is the usual paradigm. It is widely believed that the reason for this focus is that nonlinear planning provides a way to cut down on the search that linear planners do. In a sense, that is true, but the blocks world is a terrible domain to demonstrate it.

The truth is that linear and nonlinear planners are not competing. The spaces searched by nonlinear planners are quite different from those searched by linear ones. A nonlinear planner pastes together big canned plans, postponing decisions about how those plans will interact. That approach makes no sense unless each of the plans is written in a robust way that will allow it to succeed when other things are happening. That gives the planner the freedom to ignore most interactions. In other words, the planner is not avoiding interactions by means other than search; instead, it is presupposing that plans have been written so that fatal interactions are improbable. This presupposition is false in the blocks world, where all the difficulties are due to intricate combinatorics in stringing together tiny pieces of plan."

This was written pre-SNLP, when I thought of nonlinear systems as being NOAH- or SIPE-like, hence my terminology.

The funny part seems to be the way this particular meta-assumption behind NOAH/Nonlin type hierarchical planning has been sidetracked from the beginning. By the time of TWEAK, the only thing one remembers about NOAH is that it didn’t do search, and that Nonlin added search to NOAH. The idea that NOAH was attempting to do deliberative DFS, and that Nonlin had trouble exactly those times when it had to backtrack, is largely forgotten.

In fact the development of great hierarchical planners is very much at odds with the SNLP/TWEAK explanation of PO planning. For example, one of the points that is often made is that STRIPS confounded planning and execution order, and NOAH set it right by taking the search to plan space. However, by the time of FORBIN and SIPE, the considerations of integrating planning and execution got us back to being interested in deliberately doing planning in an order closer to execution order! Similarly, NOAH is criticized for not doing search, and yet by the time of FORBIN, search is deliberately _avoided_.

These two different ways of rationalizing the shift to PO planning, in my opinion, also explain why when looking at NONLIN/SIPE/FORBIN through SNLP/TWEAK glasses, one has trouble appreciating many of the features of HTN planning. Consider for example, the work on typed preconditions and in particular what have been called filter conditions/reduction assumptions. Collins & Pryor [aaai-92] argue, and rightly so, that a best-first search with a reasonable heuristic can get the same functionality as filter conditions eventually. Of course! In fact, I will go one step further and say that any sort of deliberative pruning strategy -- be it filter conditions, loop-pruning or critics or projection, will be hard to justify when we expect to do a best-first search anyway. Afterall, the erroneous branches will die out eventually, or become bad enough that the heuristic will black-list them. There isn’t that much currency in removing the unpromising branch right away.

Looking at the same situation from the Hierarchical planning point of view, which makes the meta-assumption of stability and expects to be able to get by with deliberative depth-first search, pruning strategies and critics are extremely important. You are NOT interested in maintaining a full search tree, and do a best-first anyway. So, you might as well spend as much time as possible critiquing the current plan and the pending choices before refining it further. From this, point of view, filter conditions/reduction assumptions do provide valuable guidance.

To be fair, the competing rationales for PO planning seem to have confused HTN planning just as much as they are confusing SNLP planners. In particular, it seems that the idea of doing chronological backtracking in Nonlin or other HTN planners is kind of misdirected. If you are making the assumption that plan fragments can be put together reasonably easily, and are thus doing deliberative depth-first search, then doing undirected chronological backtracking at the first sign of trouble is unjustifiable. It is more reasonable to repair the failing plan to avoid deadend, and proceed from there.

What is really required, it seems to me, is a sophisticated replanning strategy, which will allow the planner to retract the "wrong decisions" and continue, when it gets into a deadend. Combined with a good deliberative depth first exploration of the search tree, this strategy makes much more sense than chronological (or even dependency directed) backtracking. (I think PRIAR reuse framework can be used for this purpose -- to do intra-plan reuse, as it were; and we are checking the effectiveness of this currently).

Now this sort of retraction is not exactly a complete taboo for HTN planners; Nonlin's dephantomization decision is really a retraction (in contrast SNLP will put simple establishment in one branch and establishement through step addition in another branch). The problem of course, is that Nonlin stops at dephantomization, and delegates other types of retraction to (chronological) backtracking. The strategy proposed above essentially makes a clean break with refinement planning.

What about O-Plan-2? Does it backtrack at all, or just keep removing "flaws"?

Comments?

I agree with you completely!

Well, I guess I wouldn't be an academic if I agreed with anyone completely. Note that my quote above is aimed in a slightly different direction from your observation. It seems to me that hierarchical plans ought to be written in such a way that even with no further planning at all they can be conjoined with other plans (although the result may be suboptimal). The purpose of planning is then to remove the resulting inefficiencies. Not surprisingly, that's how my current system works.

-- Drew

------- End of forwarded message -------

------Forwarded message from Austin Tate:

>From: Austin Tate <bat@aiai.edinburgh.ac.uk>
To: rao@asuvax.asu.edu
Subject: Re: comments requested: hierachical planning, critics, transformational planning etc.
Date: Mon, 19 Apr 93 12:44:18 BST

I am just back from holiday Rao and am having a quick look at your message.

Immeditately, as you nite, I can say though that I always viewed Nonlin (and NOAH) as wanting to splice together well worked out plan fragments as you note. The drive for the design of Nonlin was a large electricity turbine overhaul project planning domain. Here there were plan framgements in the form of PERT plans already in exsistence. In some cases there were almost no additional "goals" or conditions, though some optional tasks could be included. We were working on this and the Interplan (linear planner for my thesis work on interacting goals and "ticklist" critics on goal structure of a plan) when Earl Sacerdoti spent some time with us at Edinburgh after his ABSTRIPS work.

I consider the NOAH/Nonlin type of planner uses "action expansion" as the main basis of its work only doing "goal achievement" as one of its tactics to satisfy unsatisfied conditions. hence my concentration on "typed" conditions to try to get domain knowledge to AVOID this very expensive tactic wherever posible!

Nonlin (and NOAH) work by action decomposition to lower levels primarily. The top level "task" even in Nonlin/O-Plan can bring in a large sub-plan already well ordered in advance and well tailored to the environment using trigger condition checks to select the right sub-plan.

O-Plan even removesd the idea of "goals" as nodes in the plan, just treating them as another type of condition on the ACTIONS or time points in a plan. Block stacking is hardly the type of domain these systems are designed for, though they can cope on those too for small problems. They come into their own for larger wel constarined knowledge rich domains (such as those first started to be looke at in DEVISER).

I also agree that Nonlin/O-Plan is about AVOIDING search by building up constraints and doing forward projections, etc.

Typed conditions are important to let a domain writer provide the knowledge he has of the domain. They can then be used by the planner to avoid search that is known by the domain writer to be pointless.

I agree that dependency directed plan repair is the only sensible strategy for the Nonlin/O-Plan type of planner. Nonlin had a decision graph in its 1977 version which allowed this (see daniels, 1982 in an article in a book called AI: Tools, Techniques ansd Applications). So this went much further than the other simple non-monotonic types of reversal of decision which the 1975/6 Nonline (as published in 1977 in IJCAI) could do. We have also experimented with decision graph based dependency directed search in O-Plan1. Its just a question of the maturity of the implementation there, not philosophy.

Hope these notes help in your thinkling Rao. I attach here a paper I am writing about condition types for the DARPA planning initiative work we are doing. I quickly took out our local DARPA paper style information, so it may not go through latex without editing out some of the local specials. Austin

```
-----
%**
%** Paper on Condition Types in O-Plan2
%**
\documentstyle [11pt,]{article}
\projectdocument{1}{The Use of Condition Types in O-Plan2}
                {Technical Report}{TR/7}
\created{Austin Tate}{September 12, 1992}
\lastaltered{Austin Tate}{March 23, 1993}{11:36}

\parskip=6pt
\vfuzz=30pt
\hfuzz=30pt

\begin{document}

\section*{Austin Tate -- The Use of Condition Types in O-Plan2}

\section*{Abstract}

The aim of this document is to give a description of the use
```

```
of condition type information to restrict search in O-Plan2.
This information is provided via the domain description language
Task Formalism ({\sc tf}) to O-Plan2.  A definition of each
condition type used in O-Plan2 is given in domain writer terms.
The way in which each condition type is handled by the planner
is documented.

\section{Introduction to Condition Typing}

One powerful means of restricting search in a planner is to recognise explicit
precondition types, as introduced into Nonlin \cite{tate77}
and subsequently used in other systems \cite{vere81,sipe88}.
One main form of search reduction in O-Plan is through the use of such
condition typing.

This technique allows domain knowledge to be used to prune the search.  It is
fed into the system via the Task Formalism ({\sc tf}) domain description
language.  The domain writer takes the responsibility for a deliberate pruning
of the space.  This caused us to
adopt the term {\em knowledge based planning} to describe our work.

A more general approach, not using such domain knowledge, is manifest in the
{\sc tweak} type formal approach (\cite{tweak}) which necessarily includes no
search control issues.  Chapman's work therefore provides a description of the
search space, but not a specification of how to control or prune search in
that space.

Condition typing can be very successful but there is work to be done on how
far this technique can be developed.  It is often difficult for a domain
writer to choose the correct type for a condition to most effectively restrict
the search space while not over-indulging and throwing away plans which should
be considered valid in the domain.  In practice, condition typing is essential
on realistic problems in order to reduce search spaces to a manageable level,
and this can be done effectively by a domain writer providing instructions to
the system about how to satisfy and maintain conditions required in the plan.

\section{Condition Types in O-Plan}

Conditions play a greater role in O-Plan than in previous systems since there
is no {\em special} notion of {\em goal}.  Nonlin style goals become simply
{\tt achieve} conditions in O-Plan.  Conditions are one of the most elaborate
of all {\sc tf} statements due to the variety of condition types identified as
being necessary in O-Plan.  The main condition types are:

\begin{itemize}

\item {\bf only\_use\_if}
This is a filter or applicability check on the use of the schema.

\item {\bf only\_use\_for\_query}
conditions are used to make queries at a point in the plan (to instantiate or
restrict a variable in a schema).  For instance to find out which block is on
top of another as in the case \{on ?x block1\} = true.  The answers returned
are context dependent and have to be treated as re-establishable at some later
point during planning if they become no longer appropriate.  Due to this
re-establishment property, it is important to write schemas including {\bf
only\_use\_for\_query} so that they do not depend only on the initial
bindings.

\item {\bf supervised}
A condition is satisfied directly within the schema containing it by the
introduction of a suitable effect (or alternative effects) at an earlier point
or by the direct inclusion of an action known to achieve the necessary effect
(at some level in the schema's decomposition).

\item {\bf unsupervised}
This describes a condition which must be satisfied at the required point, but
it is assumed that, in circumstances in which the schema introducing such a
condition is used, that the condition will have been satisfied elsewhere.

\item {\bf achieve}
A condition which can be satisfied by any means available to the
planner (including the addition of new actions).

\item {\bf compute}
conditions provide the {\em O-Plan2 External System Interface}.  They
are not conditions satisfiable directly from effects within a plan.  A {\bf
compute} condition describes a requirement which can be satisfied using
information from an external system (or database or user).

\end{itemize}

Other condition types can be identified but the ones above have been found to
be useful ways to extract knowledge from a domain writer in a form that can be
used to restrict search in an AI planner.  The control of planner search via
condition types is worthy of a serious study in its own right, and could form
an ideal Ph.D. topic.

Condition typing allows information to be kept about when, how and why a
condition present in the plan has been satisfied and the way it is to be
treated if the condition cannot be maintained.  However, use of this
information itself will almost certainly commit the planner to prune some of
the potential search space thereby losing completeness of search if the {\sc
tf} writer uses an inappropriate condition type.  Unfortunately this puts a
burden on the domain writer and can make domain writing a difficult job.

Condition typing helps direct the planning process, but it also requires the
domain {\sc tf} writer to structure the hierarchy of the tasks or actions more
clearly.  It forces checks to be made on processes or actions which should
communicate with others ensuring they actually do advertise their results
through a common vocabulary.

\section{Condition Types for the TF Writer}

This section presents an attempt to give definitions of condition types in
terms of what information a domain writer providing a library of action or
plan components can state, without knowledge of how the AI planner would go
about using this in detail.

\subsection*{Definitions}

\begin{description}

\item[the environment]
a plan within which a schema containing the given condition may be used.

\item[condition satisfaction]
ensuring a condition is satisfied.

\item[condition achievement]
the special case of satisfying a condition by adding new actions or expanding
a schema.
```

```
\end{description}

\subsection*{Condition Types}

\begin{description}

\item[only\_use\_if]
A filter on the relevance of the schema based on a statement
in the environment which it is not anticipated will be altered
during the required range.

Alternative wording: a necessary condition on the applicability of the schema
and one anticipated as not being refuted over the required range.

Normally used to filter out non-applicable schemas.

\item[only\_use\_for\_query]
A condition anticipated as being satisfied in the environment.

Normally used to bind variables appearing in the condition.

\item[supervised]
A condition established by (one or more alternative nominated)
substep(s) of the schema's decomposition.

Normally used to protect conditions across time intervals within
a schema.

\item[unsupervised]
A condition which is anticipated as being established elsewhere in
the environment in which this schema is used.

Normally used to order steps in a plan to meet sequencing requirements.

\item[achieve]
A condition which may be satisfied by any means available to
the planner (including adding new plan structure).

\item[achieve after $<$time point$>$]
As {\bf achieve} but subject to temporal restriction if new plan structure
is added.

\end{description}

\section{Triggers on when to attempt to satisfy a condition}

To be added in a later revision of this technical report.

\section{Planner tactics to satisfy, maintain and re-satisfy conditions}

\subsection{O-Plan2 Condition Satisfaction and Re-satisfaction Tactics}

The Question Answerer ({\sc qa}) procedure is used in O-Plan2 to establish
whether conditions are satisfied at a point in the plan or to propose plan
state changes that may allow the condition to be satisfied at that point.
The interface to {\sc qa} is to ask:

\begin{verbatim}
  <pattern spec.> = <value spec.> at <node end> using <tactics>
  or (P=V at N using Tactics) for short
\end{verbatim}

Given a particular condition type the {\sc qa} can use any of the permitted
tactics available to satisfy it and (where permitted) to re-satisfy it should
it be broken by the addition of a new effect.

The following table represents the tactics to be used by {\sc qa}
in initially satisfying a particular condition type.  ``Deeper'' lever tactics
typically have greater impact on the plan -- making more changes to
it.  O-Plan2 currently assumes it is best to make no changes, then
better to only cause bindings of plan state variables, then next best to make
temporal orderings or links, and then it assumes it is worst to satisfy a
condition by expansion -- thus introducing new plan actions, etc.  In actual
fact, this is a simplification.  A more comprehensive analysis of which option
is best against a plan utility measure is really required in due course.

\begin{verbatim}
Tactics Available                    Condition Types

None                |             |                    |               |
------------------- |Only-use-if  |Only-use-for-query  |Unsupervised   |Achieve
Always              |Supervised   |                    |               |
Already-Satisfied   |             ?                    |               ?
------------------- |             |                    |               |
Always-with-binding |             |                    |               |
By-binding          *             ?                    |               |
-------------------                                    |               |
Link-no-binding                                        |               |
Link-with-binding                                      ?               ?
-------------------                                                    |
Expand                                                                 *
\end{verbatim}

\begin{description}
\item[*] at the end of a band means that the tactics above it in the band
cannot be repeated (the condition once satisfied must be maintained).

\item[?] in a band means the tactic can be repeated to re-satisfy
a condition.  If the {\bf ?} appears only at the end of a band, this means that
all tactics are used together to get all the alternatives at once, and if
multiple answers are possible these are noted at that time.  If the {\bf ?}
appears in the middle of a band (possibly at several different points), this
means that the ``deeper'' tactics are only used if the earlier ones were
attempted and did not produce any result.  This allows, for example, an
achieve condition to be noted as already satisfied at first (without
generating {\em any} of the other possibilities which may be valid using the
other deeper tactics which can come into play later if necessary if the
condition range is violated and cannot be re-satisfied with the same
tactic).

\item[Expand] note that the Expand tactic is not part of the {\sc qa} system in
O-Plan2 - it is sanctioned by the {\sc ks\_achieve} knowledge source if
required (due to failure of the other available tactics).

\end{description}

``Deeper'' level tactics within the allowed band can be used if the earlier
tactics prove unsuccessful at satisfying a condition.  These can also be used
to re-satisfy a condition previously satisfied by a simpler tactic but which
are subsequently violated (where the condition type permits this).  The tactic
```

```
‘‘none’’ is used to indicate that no tactic has yet been applied (and thus the
condition is not yet satisfied), and is useful for condition maintenance
within the planner.

\subsection*{O-Plan2 Current Tactics}

At present, the tactics used for condition satisfaction in O-Plan2 version 1.2
differ from the intended tactics shown in the table above.

\begin{itemize}

\item {\bf only\_use\_if} and {\bf only\_use\_for\_query} condition types
use deeper level tactics than in the table above -- allowing linking as well.
This is due to potential limitations of the triggering of the time at which
condition satisfaction is attempted in the current release.

\item {\bf unsupervised} does not repeat its tactic as allowed for above --
therefore not allowing it to be violated and re-satisfied (as the tactics
available to satisfy unsupervised conditions is used late in planning in
O-Plan2 using the default agenda priority function).

\item {\bf supervised} condition ranges are protected as stated in the schema
which introduced them, but O-Plan2 does not at present ensure that the actual
contributor is identified and tied into the protected range.  This can lead to
‘‘holes’’ in the Goal Structure if the TF domain writer does not
ensure the correct behaviour.

\item {\bf only\_use\_for\_query} tries all its tactics at the same time,
it does not try to find satisfying contributors which do not require
bindings first (as the table above allows).

\end{itemize}

\subsection{Maintenance Requirements for Conditions}

The condition type gives the planner an indication of how important the
protected range for that condition is, i.e., how much commitment the planner
has to preserve it.  It also indicates the tactics to be adopted
if the satisfied condition protected range is violated.

\begin{description}

\item[only\_use\_if] must be maintained (it is expected to be invariant
over the required range).  The schema’s inclusion would not have been
sanctioned by the filter had the condition not been satisfied when chosen.

\item[only\_use\_for\_query] can be undone and re-satisfied at any time.

\item[supervised] must be maintained (for a minimum of one contributor)
over the required range.  The schema specifically included sub-activities to
ensure the satisfaction of the condition, if these internal intentions are
broken, the schema should be considered inappropriate (unless re-sanctioned in
the changed environment).

\item[unsupervised] must be satisfied by the end of planning.  It
may be undone and re-satisfied if required.

\item[achieve] can be satisfied by any means available to the planner.
It can be undone in all cases except where new actions were included to
satisfy the condition by expansion.  In this case, if the condition is undone,
the initial use of the expansion must be considered in-appropriate (unless
re-sanctioned in the changed environment).

\item[compute] ‘‘condition’’ maintenance requirements (if any)
will be stated in the {\sc tf} {\tt compute\_conditions} statement.

\end{description}

In all cases where a condition type which {\em cannot} be undone is violated
after it is initially satisfied, the current plan state should be poisoned.

\section{Condition Type Correspondence to Nonlin, \Sipe and ACT}

Nonlin was the first Edinburgh planner to use the Task Formalism ({\sc tf})
language.

The {\sc sri} \Sipe planner also includes support for a number of condition
types.  A development of the \Sipe domain description language to link to work
on the {\sc sri} {\sc prs} Procedural Reasoning System reactive execution
support system is now underway to create a shared domain description language
{\sc act}.

This section shows the correspondence between Nonlin, \Sipe and {\sc act}
condition types and those used in O-Plan2 {\sc tf}.

\begin{description}

\item[only\_use\_if]
This is the same as Nonlin’s {\bf usewhen} (originally called {\bf holds}).
It is also the same as a {\bf precondition} in either \Sipe or {\sc act}

\item[only\_use\_for\_query]
In Nonlin and \Sipe, such conditions were part of the {\bf usewhen} or {\bf
precondition} respectively and not treated separately.

The re-establishment of the condition if it could not be maintained with the
original contributor(s) was not possible in Nonlin or \Sipe.  That is, they
treated a query condition in the same way as an {\bf only\_use\_if}, except
that it was assumed that variables would be bound by satisfying it.

As in O-Plan2, in {\sc act} {\sc sri} have chosen to separate these for
clarity -- it is called the {\bf setting} in {\sc act}.

\item[supervised]
The same as a {\bf protect-until} in \Sipe and {\bf require-until} in {\sc
act}.

\item[unsupervised]
\Sipe introduced an {\bf external} condition to give this capability.
In {\sc act}, this is called {\bf wait-until}.

\item[achieve at N]
There is no equivalent in Nonlin, \Sipe or {\sc act}.

\item[achieve at N after $<$time point$>$]
The same as a {\bf goal} in the expansion/decomposition part
of a schema in Nonlin and \Sipe. In {\sc act} it is called {\bf achieve}.

\end{description}

\section{Condition Types -- KRSL Considerations}
```

```
The following notes were prepared in discussions between Nancy Lehrer (ISX),
Drew McDermott (Yale University) and Austin Tate at the San Antonio workshop
of the DARPA-Rome Laboratory Planning Initiative in February 1993.  They show
how condition types can be characterised with respect to KRSL.

\begin{itemize}

\item Filters:
\begin{itemize}
\item Applicability information.
\item Must be satisfied before this point in the plan.
\item Cannot cause reordering links.
\item Cannot cause plan expansion.
\item Cannot be undone at point where satisfied.
(usually cannot be altered at all -- invariant).
\item {\bf only\_use\_if} in O-Plan2.
\item Appears in high level KRSL:plan definition.
\end{itemize}

\item Preferred (‘‘Filters’’) Conditions:
\begin{itemize}
\item Preferred applicability information -- condition/utility pair.
\item Same as filters but conditions only "preferred" to be true.
\item Used for finer-grained sub-plan applicability.
\item no direct analogue in O-Plan2 (except via {\bf prefer\_schemas} {\sc tf}
statement and ordering of schemas in TF)\footnote{O-Plan2 {\sc tf} has
preference information appear in separate TF forms, not mixed up with filter
conditions.}.
\item Appears in high level KRSL:plan definition.
\end{itemize}

\item Parameter Binding Constraints:
\begin{itemize}
\item Parameter binding constraints.
\item Cannot cause reordering links.
\item Cannot cause plan expansion (new goals posted).
\item {\bf only\_use\_for\_query} in O-Plan2.
\item Appears in high level KRSL:plan definition.
\end{itemize}

\item Supervised Conditions:
\begin{itemize}
\item Must be achieved as an effect of this sub-plan or its lower
level decompositions.
\item Cannot cause new temporal reorderings.
\item May be achieved by plan expansion within the sub-plan.
\item Appears as a decomposition node of a KRSL:plan definition.
\item Temporal ordering is triggered by including the node in the
temporal graph.
\end{itemize}

\item Unsupervised Conditions:
\begin{itemize}
\item Must be satisfied by an effect of a different sub-plan.
\item Can cause temporal reorderings.
\item Cannot cause plan expansion (i.e., cannot post a goal).
\item Temporal ordering is triggered by including the node in the
temporal graph.
\item Appears in decomposition graph of KRSL:plan definition.
\end{itemize}
```

```
\item Achieve:
\begin{itemize}
\item Can be achieved by any means:
\begin{itemize}
\item effect of this sub-plan or any other sub-plan.
\item ordering link within this sub-plan or between sub-plans.
\item posted as new goal.
\end{itemize}
\item Appear as decomposition node in a KRSL:plan definition.
\item Temporal ordering is triggered by including the node in the
temporal graph.
\end{itemize}

\end{itemize}

Suggested usage in KRSL:plan (provided by Nancy Lehrer 20-Mar-93):

\begin{verbatim}
(ks:define (plan <plan-name>)
   ...

  :filter <condition>
  :preferred-filters ((<condition> <utility>)*)
  :parameter-binding <condition>

  ...
  :decomposition
    (:nodes ((:label <node label>
              {:supervised | :unsupervised | :achieve} <condition>
             ...)
            ...)
     :graph (:and
             <temporal relation>*
    ))

\end{verbatim}

Example (provided by Nancy Lehrer 20-Mar-93): a plan to put two specified
blocks on any red block.  ‘‘AT Notes’’ below show cases where the usage does
not correspond to the framework for condition types and their intended
definition as established in this paper.

\begin{verbatim}
(ks:define (plan red-three-block-tower)
   :parameters ((x :type-restriction block :properties (:input))
                (y :type-restriction block :properties (:input))
                (z :type-restriction block)
   :filter (:and (clear ?x) (clear ?y))
                 ;;; (clear ??) cannot be a filter - AT Note
   :parameter-bindings (color ?z ’red)
   :decomposition
   (:nodes ((:label clear-z
             :achieve (clear ?z))
            (:label move-x
             :supervised (move ?x ?z))
                 ;;; action name cannot be supervised condition - AT Note
                 ;;; must be a world statement.
            (:label move-y
             :supervised (move ?y ?x)))
                 ;;; action name cannot be supervised condition - AT Note
    :graph (:and
```

```
            (interval-before (time-of clear-z) (time-of move-x))
            (interval-before (time-of move-x) (time-of move-y)))))

\end{verbatim}
```

\begin{thebibliography}{99}

\bibitem{tweak} Chapman, D. Planning for conjunctive goals.
{\em Artificial Intelligence Vol. 32, pp. 333-377, 1987}.

\bibitem{tate77} Tate, A. Generating project networks. {\em In procs. {\sc ijcai-77}, 1977}.

\bibitem{vere81} Vere, S. Planning in time: windows and durations for activities and goals. {\em {\sc ieee} Transactions on Pattern Analysis and Machine Intelligence Vol. 5, 1981}.

\bibitem{sipe88} Wilkins, D. Practical Planning. {\em Morgan Kaufman, 1988}.

\end{thebibliography}

\end{document}

```
>From ihrig@enws318.eas.asu.edu  Wed Apr 28 15:15:17 1993
Return-Path: <ihrig@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA26597; Wed, 28 Apr 93 15:15:17 MST
Received: from enws323.eas.asu.edu by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA05329; Wed, 28 Apr 93 15:11:45 MST
Date: Wed, 28 Apr 93 15:11:44 MST
From: ihrig@enws318.eas.asu.edu (Laurie Ihrig)
Message-Id: <9304282211.AA05329@enws318.eas.asu.edu>
To: plan-class@parikalpik
```

Notes for April 22 Laurie H. Ihrig

Two Types of Learning:
- Analytical Learning (Speedup Learning)
  - -learned knowledge is in deductive closure of what is already known
  - -associated with an improvement in performance eg. plan reuse
- Inductive Learning
  - -learned knowledge is not in the deductive closure of what is known
  - -very hard in general

Abstraction: Methods of Automatically Constructing Predicate Heirarchies

1.a Predicate Relaxation
- -successive regressions of predicate over all actions in domain
- -purpose is to assign an importance to a predicate
- -determines the length of the plan to attain a condition
  ie if nth level of relaxation is a tautology
  then the length of the plan is n

$$P^0_{rel} = P$$

$$P^1_{rel} = P \vee \bigvee_{\text{for all operators } O_i} Reg(P, O_i)$$

$$P^n_{rel} = P^{n-1}_{rel} \vee \bigvee^{i}_{\text{for all operators } O_i} Reg(P^{n-1}_{rel}, O_i)$$

- -the number of disjuncts grows with the level
- -if you have ADL operators you have tidy regression operators since

  Reg (PVQ) = Reg(P) V Reg(Q)

1.b An example of Predicate Relaxation

$$he^0_{rel} = he$$

$$he^1 = he \vee Reg(he, putdown) \vee Reg(he, stack)$$

(the regression of he over pickup is F, therefore ignore it
-also ignore unstack)

Since the precondition of putdown(x) is holding(x), and the precondition of stack(x,y) is holding(x) ^ clear (y), the above simplifies to:

There exist x,y such that

$$he^1_{rel} = he \vee holding(x) \vee (holding(x) \wedge clear(y))$$

Since he V holding(x) is a tautology, the length of a plan for he is 1.

On the other hand, on(P,Q) never becomes a tautology-this means there are infinitely long plans for on(P,Q).

1.c Thoughts on Predicate Relaxation

Hard to achieve predicates (ones with longer plans) are considered more important (and should be worked on first).

The method above gives the longest plan that will be needed to achieve a condition. However, this analysis does not take into account the interaction between subgoals. This means that the actual plan may take longer. It also reflects only the length of the search space, not the breadth.

It might be argued that the length of the plan is not what abstraction should be about. For example, it could be the interaction between subgoals should be taken into account.

2.a Example from machine-shop domain:

Operator	Shape(x)
Preconditions	Object(x), not Fastened(x)
Effects	Shaped(x), not Drilled(x), not Painted(x)
Operator	Drill(x)

```
  Preconditions             Object(x), not Fastened(x)
  Effects                   Drilled(x), not Painted(x)

Operator                    Fasten(x)
  Preconditions             Object(x), Drilled(x)
  Effects                   Fastened(x)

Operator                    Paint(x)
  Preconditions             Object(x), Steel(x)
  Effects                   Painted(x)
```

Shaping removes paint or drilled holes and requires that the object be unfastened. Drilling also removes paint and also requires unfastened. Fastening requires drilled. Painting is possible only for steel objects.

2.b Knoblock’s Mechanism for Predicate Classification

According to Knoblock’s technique, we construct a graph where nodes are clauses (conditions) and the edges are added as follows:

Take the goal of the planning problem. Suppose that a goal clause matches the effect e1 of the operator O. Then add a directed arc from e1 to each of the other effects of the action and to each of its preconditions. Repeat for each goal and each precondition.

The strongly connected components of the graph (SCCs) define groups of clauses that are assigned the same level in the heirarchy.

Suppose the operators are as above, and the goal is:
   Shaped(x), Fastened(x), Painted(x).
The resulting graph is:

```
        ---------Shaped(x)---------
     /          /          \        \
               V      ->    V
    /     Drilled(x) <-   Fastened(x) \
             |   \          /
    \        V    \        /         /
      > Painted(x) |      |         /
            /  \   V      V        /
            V   >  Object(x)     <
        Steel(x)
```

According to Knoblock, the heirarchy is:

Shaped(x)
Drilled(x) Fastened(x)
Painted(x)

and Shaped should be worked on first and Painted last.

Thoughts on Knoblock’s Method:
In the situation where only one of many objects is steel, Painted(x) is the most difficult goal to satisfy, and to be efficient the planner should work on it first. Otherwise x will be bound prematurely to objects which are not steel resulting in needless backtracking.

Knoblock’s heirarchy is really only good for linear planning, since it lessens the amount of backtracking caused by subgoal interaction, but is probably useless for PO planning.

In general, we should work on goals that result in lower branching factor first, since this means less backtracking.

Also, we should work first on goals for which there is only a small number of ways to achieve them (eg. only a few objects are steel and can be painted). this results in early pruning of paths.

It becomes obvious from the above analysis that the generation of a precondition abstraction is particular to the domain and would have to be prespecified or learned for each domain.

```
From gopi@enws318.eas.asu.edu  Thu Apr 29 16:47:35 1993
Return-Path: <gopi@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA27829; Thu, 29 Apr 93 16:47:35 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA06249; Thu, 29 Apr 93 16:44:03 MST
Date: Thu, 29 Apr 93 16:44:03 MST
From: gopi@enws318.eas.asu.edu (Bulusu Gopi Kumar)
Message-Id: <9304292344.AA06249@enws318.eas.asu.edu>
To: plan-class@parikalpik
```

AGENDA

U N C H A R T E D   T E R R I T O R Y
----------------- -----------------

HTN PLNNING THE REAL STORY
----

1) TASKS

2) TIME MAPS

3) PROJECTION PROBLEM

Classical View Of Planning

Classical view of planning is very limited ! A more general planning problem should be looked at. Give a model of planning :

-> Can you pose a given problem to your planner ?

E P I S T E M O L O G I C A L   A D E Q U A C Y

-> Can you solve this problem efficiently ?

H E U R I S T I C   A D E Q U A C Y

Task based model of planning, to a certain extent, satisfies the first of these questions. It also to a certain extent, covers the kinds of plans classical planners can represent and solve.

GOAL LANGUAGE

The GOAL LANGUAGE of a planner decides the epistemological adequacy of the planner. We will now see the goal language of classical planning :

GOALS : Assertions on state descriptions

ACTIONS : State changers

The framework of classical planning views planning as described above. Planning problem is thought of as being going from initial state to the goal state. Various classical planners differ in H O W they organize the search to solve the problem and N O T in the way they look at the problem O R in what kind of problems it can solve. Based on these we have classical planners, which are organized as :

-> STATE BASED PLANNERS (STRIPS)

-> PLAN BASED PLANNERS (Refine a plan till it is correct)

The plan based planners themselves are further classified as

-> TOTAL ORDERING PLANNERS
-> PARTIAL ORDERING PLANNERS

and

-> PARTIALLY INSTANTIATED PLANNERS
-> TOTALLY INSTANTIATED PLANNERS

Having talked about goal language of classical planners, and as to why all kinds of classical planners have the same goal language, a few goals (IN ENGLISH) can be considered, and we can see whether these can be represented in the goal language of classical planners.

-> Put a red block on top of a green block.

--> ON (X,Y) ^ GREEN (Y) ^ RED (X)

-> Put A on B, remove A from B, then put A on C

--> NO, we cannot represent this in the goal language of classical planners !

The reason why we could not express the second goal in goal language of classical planners is because, the above goal in some sense talks about the behaviour of a plan, rather than asserting something about some state.

H T N P L A N N I N G

[Refer to Chapter 9 of Charniak & Mc Dermott]

HTN planning takes a different view of planning, here planning is not achieving a state, which satisfies certain assertions, but it is reducing a task into sub tasks till the plan consists of only primitive tasks.

THE HTN PLANNING ALGORITHM

* Pick a task, if it is primitive do nothing

* Pick a method from the library

* Apply it, (Reduce)

* Project the effects of the new tasks

* Consider task interactions

* Repeat from step 1

In the above procedure there is no direct truth criteria to judge the correctness of the plan ! But tasks do have effects, but these are used to check interactions. There are various kinds of interactions which can be summarized as follows :

1) Reduction Assumptions

When a task is reduced, there may be certain assumptions made, when these are violated by the reduction of some other task, we have interactions which have to be dealt with !

example : g1 Bomb airport
g2 Get out by plane

One effect of achieving g1 is Delete Airport
AND one Reduction Assumption for g2 is "Airport present"

Now there is an interaction between g1 and g2, and this has to be resolved

2) Protection Violations

If on (A, B) is achived by a task, and some other task removes this effect, we see an interaction. There is no difference between 1 & 2 from the view point of a planner like TWEAK. In HTN planning MTC can be used to validate only part of the plan, overall correctness is dependent on the task reduction (Which cannot be checked by the planner)

3) Projection Violations

In HTN planners, what assertions are present over which time is stored by a time map manager. These then can be found by using the time manager. When a task is projected a few assumptions can be made, if later these assumptions are violated, the task should be reprojected !

Finally, to the extent the methods are correct, the plan is correct if all the above interactions are taken care.

Further references : Mc Dermott : Planning & Acting

From wtsai@enws320.eas.asu.edu Tue May 4 16:35:34 1993
Return-Path: <wtsai@enws320.eas.asu.edu>
Received: from enws320.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA03184; Tue, 4 May 93 16:35:34 MST
Received: by enws320.eas.asu.edu (4.1/SMI-4.1)
        id AA25614; Tue, 4 May 93 16:28:06 MST
Date: Tue, 4 May 93 16:28:06 MST
From: wtsai@enws320.eas.asu.edu (Wan C. Tsai)
Message-Id: <9305042328.AA25614@enws320.eas.asu.edu>
To: plan-class@enws228

Notes for the Class of Planning Seminar on Apr. 29, 1993
Written by Wan-Chu Tsai

* Time dependent planning

A realistic planner has to deal with temporal information which interact with external world. The following example is a reasonable planning, but can not be solved (or cannot be solved without using specially domain-dependent knowledge) by a STRIPS or other planners we have talked about so far.

Initial state : Blocks A, B, C are on the table.
Goal state : Pick block A, put on block B, and then put on block C.

The temporal constraint is the difficult part to be modelled. Other modelling language that is suitable for representing this kind of temporal information is needed.

- Projection (Definition from McDermott & Charniak)

  (1) The inference from what is true at one time to what is true at another.
  (2) The maintenance of and retrieval from a databese of such inference.

  Projection is not used to make sure the plan is correct. It is only used to make sure some effect is true within some interval.

- Temporal database

  A predicate calculus database whose elements are of the form f(t1, t2, p). Thus, holds(t1, t2, p) means p is true from time t1 to time t2, occurs(t3, t4, e) means e occurs during time t3 and time t4. p and e could be actions or events.

  The difference between action and event is that action is initiated by the planner, i.e. it is intentional, while event could be accidental. Temporal database does not distinguish action from event. It only provides the answer to a query : whether or not some particular predicate p is true at some particular point.

- Causal theory

  Example:

    If holds(t1, t2, p) and occurs(t3, t4, e) and subsum([t1,t2],[t3,t4])
    Then clip(p’, t4+d)

    This means after t4+d, p’ is no longer to be true.

  To infer about time, we need to deal with the relation between given time intervals. There are 13 relations defined by Allen’s interval logic.

- Time map management (TMM)

  A time map is a permanent database of state and event tokens. Time map manager manages the database and answer queries. It only considers discrete time point or interval.

  To answer the query, either TMM finds the effect in the database, or it performs projection.

```
                                                       +-------+
                                                       | Golas |
  +------------------------------------+               +-------+
  |  +-------------------+             |                   ^
  |  | Temporal Database |\            |                   |
  |  +-------------------+ \ +-----+   |  ?(p, t1, t2) +---------+
  |                         >| TMM |   |<--------------| Planner |
  |  +-------------------+ / +-----+   |               +---------+
  |  |   Causal Theory   |/            |                   |
  |  +-------------------+             |                   V
  +------------------------------------+         +--------------------------+
                                                  |         Library          |
                                                  | Task Reduction Formulas  |
                                                  +--------------------------+
```

  . TMM checks to see if the query is true. When doing projection, it keeps checking and takes care of the projection violation problem.
  . TD is projected.
  . Library contains information of task reduction as well as how subtasks should be put together.
  . The interval can be constrainted so that parallelism can be done.

- Algorithm

  (1) Select a task.

  (2) Using the query, todo(what, when, how), try to find some method ’how’ for carrying out the task found in step 1.

  (3) If the query specified in step 2 fails, try adding constraints to restrict the ordering of the existing tasks. This may trigger rules permitting the ’todo’ query to succeed on the next attempt.

  (4) If the query specified in step 2 fails even after trying various additional constraints, try removing one or more of the existing tasks along with all associated protections and other constraints. Be careful to reinstate the original supertask. (Dependency has to be taken care of.)

  (5) If step 2 through step 4 fail to produce an applicable method, return to

```
    step 1 and try another task.
    (May require heuristics.)

(6) If the query succeed, mark the original task as reduced and add the new
    'how' task or plan to the database, along with any specified constraints
    and protections.

(7) Upon effecting the reduction, TEMPLOG will have updated the datebase
    using the projection and persistence clipping algorithm, and the
    projection rules that describe the effects of selected actions.  Check
    to see if any protections are violated by the addition of the new tasks.
    (NP-hard problem, the heart of TMM)

(8) If any protections are violated, resolve the violation by either
    reordering or removing one or more of the existing tasks.

(9) Go to step 1.

Notes :

- This algorithm allows removing anything.

- In the real world, planner is likely to choose the method that is
  reversable, rather than optimal.

- Assertion (By TMM to TD)

  reduction assumption(...)
  holds(...)
  occurs(...)
  protection(...)
  begin(...) <= end(...)
  reduce(t1, m1)
  projection assumption(...)

- Assertion includes future assertion.

- Clip is procedual.

  If holds(t1, p) and clips(p, t2)
  Then delete holds(t1, p) and
       add holds([t1,t2], p)

- Consistency checking is hard in temporal database.

- Temporal planning can deal with real-world problem, such as the example
  given in the beginning.

- In real-world, planning and executing are usually in parallel.
  How can the planner do such that the execution can start as early as
  possible?
  Also, the order of execution needs to be considered to improve
  performance.

- Recursive task is allowed.

  Example :

    todo(Achieve(empty Truck))
         plan((unload-item(Truck),Achieve(empty Truck)
              [end(1) <= begin(2)])) <-
              holds(end(k),~empty(Truck))

* Persistence

  - Assume persistence is a rule, if nothing else happens, then this will be
    true.  But how long?

  - In real world, persistence can not be guaranteed.

    Example : Yale shooting problem

              Gun loaded
        t0 ------------------>

                Gun pointed at Fred
             t1 -------------------->

                     Gun is fired
                 t2 ------------------>

                           t3 ---------------->

     Query : Is Fred dead at t3?

     In closed world, the answer is yes, but in real world, the answer is
     unknown.  There could be many situations happening so that Fred is not
     dead.  But by default, he is dead.

     Planning usually assumes closed world environment.

  - To apply default reasoning, add one more predicate 'nothing_is_abnormal' to
    causal theory : If ... and nothing is abnormal Then ...

    Using default reasoning can solve the real world problem.

  - In STRIPS, if p is true in the initial state, then it cannot be false,
    because the initial state is fixed.  Here, p is true is based on the current
    knowledge of the planner, if the knowledge supports p, then p is believed to
    be true.

  - Projection assumption
    When to commit?  Should it nly answer the query based on current knowledge
    or should it make assumption to commit, and complete the projection, while
    later on, may undo the committment?


From gopi@enws318.eas.asu.edu  Mon May 17 18:29:44 1993
Return-Path: <gopi@enws318.eas.asu.edu>
Received: from enws318.eas.asu.edu by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA20147; Mon, 17 May 93 18:29:44 MST
Received: by enws318.eas.asu.edu (4.1/SMI-4.1)
        id AA00569; Mon, 17 May 93 18:25:52 MST
Date: Mon, 17 May 93 18:25:52 MST
From: gopi@enws318.eas.asu.edu (Bulusu Gopi Kumar)
Message-Id: <9305180125.AA00569@enws318.eas.asu.edu>
To: plan-class@parikalpik
Subject: Execution monitoring ..., class notes
Cc: gopi@enws318.eas.asu.edu


AI PLANNING 5-4-93
```

AGENDA

INTEGRATING PLANNING AND EXECUTION

INTRODUCTION

Even though plans are planned yo be executed, it may not always be true, for eexample a plan may be made to just check the feasibility of a solution. In short planning and execution need not necessarily be integtrated.

Ramifications of integrating planning and execution
----

1 EXECUTION MONITORING

If there is a plan of action, even before reaching goals state, an assessment can be made as to the progress of the plan, by looking at current state of the world anc comparing it with soem intermediate state in the plan.

2 REPLANNING

To the extent the plans model of world does not completely reflect the real world, it is possible that a plan mat not be actually executable, and in fact may fail. Clearly, the probability of plan failure depends on the correctness of the plans model of the world. A few reasons for such a failure may be

* A precondition which was not modeled

* A conditional effect not modeled

* Presence of, extraneous unmodeled events in the world

In any case the failure can be dealt with in two ways

* Taking the new situation as a new planning problem

* Modifying the current plan, this is termed as REPLANNING

Infact the second way will be the only choice left if there are other agents, and (1) the agent with the failed plan has committed certain things to the other agents based on his current plan, or (2) if these other agents have made their own plans based on the current agents plan! Therefore replanning will be the only way to achieve inter agent efficiency. (Choice one may achieve intra agent efficiency, in the second)

3 TIME DEPENDENT PLANNING

If the world changes so fast that it just does not make sense to plan off-line (because by the time the plan is made the world changes !), then planning is to be done on-line, and is termed as reactive planning. Often the next action should be taken in a constant bounded time. Thus a question arises as to how much time the planner should spend before giving a plan, if it thinks too much, the deadline may pass and the planner may not be deliver any useful plan, if it makes a decision too fast, there is a chance it did not make the best possible solution in the available time. An interesting model of a reactive planner which avoids this problem of meta-reasoning is as follows :

```
                         GOALS

                           |
      +--------------------+---------+
      |                    |         |
      | +-----+      +-----V-----+   |
      | | P   |      |           |<--+------  ENVIRONMENT
      | | L N |      | REACTOR   |   |
      | | A E |----->|           |   |
      | | N R |      |           |---+----->  ACTION
      | +-----+      +-----------+   |
      +------------------------------+
```

The reactor decides the next step based on some strategy and is independently competent, in the sense it makes progress left to itself. The planners responsibility is to improve overall chances of success over a longer time. Thus the problem of meta reasoning is solved. The reactor always takes some decision in time, however if the planner comes up with a better solution, the reactor may consider it.

HISTORY

STRIPS

STRIPS had a data structure called TRIANGLE table. After the plan is made, if the initial state Ip of the plan is not the same as the real initial state Ir, then STRIPS could look, if Ir corresponds to some intermediate state of the original plan Ii. If this is true, then the rest of the plan starting at that state is executed. TRIANGLE tables provided an easy way of checking if Ir matches with some Ii.

(Basically Ii of a step, would be the e-conditions(step) + p-conditions(step))

SUBSUMPTION ARCHITECTURE

This is similar to the concept of an individually competent reactor. Having a goal basically constrains the choice available to an agent. An agent is built with minimal competence at the base, over which more and more complicated layers are built. Thus even if a higher layer does not know how to tackle a situation, catastrophes are avoided because the base layer is still an independently competent reactor.

REACTIVE PLANNING ARCHITECTURE

```
                     PLAN NET
      +-------------------------------+
      |                               |
      V                               V
 -----------    ---------      --------------
```

```
          |          |<---| SCR’s |<------|            |
          |Executor |    ---------       |Projector   |
          -----------                    --------------
               ^                                ^
      SENSOR   |                                |
      ---------+--------------------------------+
      DATA
```

SCR == SITUATION CONTROL RULES

The projector proects an actions, sees if some catastrophic failure occurs then provides a rule to the reactor in the form "In this situation don’t do this action". Reactor does default actuons, but if a SCR is present, it will avoid catastrophes.

Example SCR

Imagine three slots 1 2 3 and problem of placing blocks a,b,c in these slots, the goal is to place a, b, c in slots 1, 2, 3 respectively, operators are place an extreme block (1,3) place and place and sweep interior blocks from left "lplace", from right "rplace"

Av - Available

Av(a) ^ av(b) ^ free(1) ^ free(2) ===> {~ place(A,1)}

Because, if a is placed, there is no way b can be placed in slot 2

If complexity of computation is of not prime importance a network of actions and the preconditions can be constructed --> PLAN NET.

If a plan net is executed in all possible ways a finite state graph results. The edges that lead to non goal nodes, without outgoing edges are to be avoided. If there is a failure path and atleast one success path, it is a critical choice point hence rules have to added to avoid the failure paths. However it may not be possible to compute the projection of a plan net. Therefore the SCR’s have to be learnt incrementally from experience.

From AZEDC@ACVAX.INRE.ASU.EDU Sun May 16 23:12:41 1993
Return-Path: <AZEDC@ACVAX.INRE.ASU.EDU>
Received: from ACVAX.INRE.ASU.EDU by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA18891; Sun, 16 May 93 23:12:41 MST
Received: from ACVAX.INRE.ASU.EDU by ACVAX.INRE.ASU.EDU (PMDF #2382) id
 <01GY9DVCCHVK0027XF@ACVAX.INRE.ASU.EDU>; Sun, 16 May 1993 23:09:24 MST
Date: 16 May 1993 23:09:24 -0700 (MST)
From: AZEDC@ACVAX.INRE.ASU.EDU
Subject: Class notes May 6
To: plan-class@enws228.eas.asu.edu
Message-Id: <01GY9DVCCHVM0027XF@ACVAX.INRE.ASU.EDU>
X-Vms-To: IN%"plan-class@enws228.eas.asu.edu"
Mime-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
Content-Transfer-Encoding: 7BIT

Notes for the class of May 6, 1993 by Eric

(no specific agenda)

Point #1: Gopi’s question concerning the goal ordering (i.e. the order in which we attempt to address the goals):

Does it make sense to keep disjunctions of refinements together (in other words, to allow disjunctions to sit in the nodes of the search space) ?

Example:
```
        +c
     s1 -> s2

        -c
        s3
```

Here, say in SNLP, we would have either s3 < s1 or s2 < s1 (in other words, we resolve that ambiguity on the spot). Gopi’s idea is to have (s3<s1)#OR#(s2<s1).

Turns out this is not a good idea. If we allow that, we are throwing the complexity into the search space; the cost per node increases. Allowing disjuncts is like allowing a set of orderings, and in that case the cost (for instance, of checking modal truth criteria) increases).

Another point is that if we are looking for a solution rather than the optimal solution, this is probably not a good idea.

We thought of PO planning for efficiency reasons and search space considerations. These motivations still hold here; it may be the case though that for some domains we may empirically prove that there are efficiency improvements by allowing disjuncts.

Point #2: In behavior planning, MTC cannot be checked. History checking may be interesting for Decision Theoretical planning, where we check for a level of satisfaction of goals rather than a predicate that can only be true or false; for example, we could pick some goal which maximizes a given utility.

For that, we need something similar to MTC, but in this case we talk of plan history instead of truth criterion.

Things in the area of planning that have not been discussed in this seminar:
.   Plan learning, EBL, reuse, derivational analogy
.   Reactive planning, time dependent planning
.   Scheduling
.   Applied problems: motion planning, assembly plan, process plan (all useful in robotics)

Areas of interest in motion planning:
.   mobile robots
.   find path, manipulation

Point #3: mobile robots that navigate in uncharted territory; confidence increases as the distance to the goal decreases.

Sensor information is critical, and dealing with uncertainty is very important. Also critical are notions such as time to act versus time to plan, etc.

The idea is to gather as much information about the world while planning as possible, and then allocate time for information gathering and planning in a way in which the utility is maximized.

Point #4: find path, manipulation
Here we have full information about the world (no uncertainty), but we need to do a lot more manipulation.

An example is the piano mover problem, where the piano can be rotated and will only be moved into a room if rotated in a certain manner. If rotation were not allowed, there would be plan to move it into the room.

There is the notion of degrees of freedom, which is equivalent to the number of rotations. The search space increases according to degrees of freedom, that is, if d.f.=2, the search space doubles, and so on.

If a robot has a certain geometry (that is, it is not a point size), a technique called configuration space approach is used. This technique is used for arbitrary shapes. Objects are "amplified" to the amount the object to be moved is larger than the point.

Point #5: for all these problems, if there are enough people interested in this class of problems, then we can forget about worrying about using a general theory, and instead we deal with them separately. This is typically what happens with factory planning, where we have a finite and limited number of entities to be worked with.

Assembly planning is another example, where we are dealing with tasks to be performed. It is typically given an order of things that need to be done; we search in the space of assembly plans, where we are looking for plans that don't interact (i.e. don't collide) with each other.

Compliant notion: we change the situation so that interactions are removed; an example is a screw hole where we are uncertain about how the hole is made and we are not really sure how to place the screw in the hole. To solve this problem, we change the shape of the hole so that there is a bigger area where the environment will guide the screw.

Scheduling is informally defined as arranging tasks that have previously been defined. The problem is to arrange the tasks according to some constraint, say the time to start, priority, etc. There are hard constraints (examples just given) and soft constraints (e.g. best possible ordering). After scheduling is done, the tasks are then assigned to the resources.

Scheduling is one complexity level below planning (which is undecidable).

Point #6: interaction with humans is an area with its own open problems. Here we would for example allow the user to pick some refinement for a plan being expanded. There are connections between human interaction and learning.

-----------END-------------------------------From rao Wed Apr 28 01:53:23 1993

Status: RO
X-VM-v5-Data: ([nil nil nil t nil nil nil nil nil]
      [nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil])
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA25865; Wed, 28 Apr 93 01:53:22 MST
Message-Id: <9304280853.AA25865@parikalpik.eas.asu.edu>
Reply-To: rao@asuvax.asu.edu
From: rao (Subbarao Kambhampati)
To: rao
Subject: HTN Planning II-- Plans as behaviors...
Date: Wed, 28 Apr 93 01:53:22 MST

0. Get A on B
1. Get some block on top of another block
2. Get A on B , and then Get A off B and put A on B

Of these, the third cannot be expressed in STRIPS goal language.

Planning problem is a really a constraint on the behaviors (or state sequences) rather than just the goal state.

The usual goal-state problem are a _special_ class of planning problems where the constraint on the behavior is expressed soleley in term of the final state of the behavior. In general, however, we could make arbitrary

The idea that Nonlin does not actually utilize the behavior based planning framework

The idea that _some_ of the behavioral specifications can be expressed in terms of ordinary goals, when we allow time argument (zeno). E.g., the goal 2 above could have been easily represented as

Role of Projection: In real HTN planning supporting goal-based planning, the role of projection is to guide planning decisions. The correctness of plan itself transcends the truth criterion. On the otherhand, in the case of TWEAK type planning, there is no difference between projection and correctness checking-- or at any rate, correctness checking can be done in terms of projection (Qn. Can we talk about the stuff of Christer etc. in terms of this disc? Can we say that projection cannot be gotten around in behavior based plannnign since we don;t ahve a way of checking plan correcntess?)

Outstanding questions:

1. How much of behavior based planning can be done by just giving a partial plan to SNLP? [pt: snlp only works on preconditions. doesn't do reduction of tasks]

2. How much of behavior based planning can be provided truth-critierion based semantics? How about some sort of archtectural + base level semantics?

3. How much of behavior based planning can be captured in terms of

4. Given that no one really seems to use "goals" as state specifciations, how much of a "big deal" is behavior based planning?

5. How does the referential opacity ideas of McDermott's code tree

etc. relate?

6. How do we make sense of Forbin in terms of behavior-based planning?

7. Does the reconstruction of HTN planning interms of efficiency have
any meaning?

---
tell kutluhan that tasknets are no magic. The point he is making is a "used-to-be-well-known" one--- that planning is really about behviors (which are sequences of states), rather than about state changes (which is just a pair of states).

Indeed, certain types of

The tricky problem about Tasknetwork planning is one of "correctness check"-- it is not easy to reason about a plan and say "yes, this plan does do what I want it to do".

What happened traditionally is that the so-called HTN planners essentially stuck to the notion of state-change. The tasks are completely characterized by their effects -- in particular, int eh case of NONLIN, every task-reduction schema's todo is represented as an effect of at least one of the subtasks of the schema. THis means that we can actually use TWEAK truth crterion to check the correctness of the resulting plans. Contrast the more general idea of HTN planning as a support for behavior based planning-- eg. the round trip plan. Here there is no way of looking at the plan and proving correcntenss of the plan simply using MTC.

As McDermott mentions in the Formal Reasonign about commonsense paper, most interesting tasks really cannot be split into simpler tasks (e.g. the eggs stuff). One kind that can be are composite tasks that simply correspond to conjunctive goals.

By the way, I was surpised to see that your bibliography doesn't cite _any_ of the papers of McDermott. He has written extensively on the semantics of tasknetwork planning -- of particular interest are his two papers in readings in planning, and another paper in the book "Formal reasoning about common sense".

From rao Tue Jul 27 16:30:54 1993
Return-Path: <rao>
Received: by parikalpik.eas.asu.edu (4.1/SMI-4.1)
        id AA12299; Tue, 27 Jul 93 16:30:54 MST
Date: Tue, 27 Jul 93 16:30:54 MST
From: rao (Subbarao Kambhampati)
Message-Id: <9307272330.AA12299@parikalpik.eas.asu.edu>
To: ai@cs, suresh@enws318, gopi@enws318, ihrig@enws318, dchen@enws228,
        cohen@enws318, plan-class
Subject: AIPS-94 (Planning systems conference-- Early Announcement)
Reply-To: rao@asuvax.asu.edu

************* CALL FOR PAPERS *************

SECOND INTERNATIONAL CONFERENCE
ON
AI PLANNING SYSTEMS

The University of Chicago
Chicago, Illinois
June 15th -17th, 1994

****

CONFERENCE CHAIR
Austin Tate - University of Edinburgh

PROGRAM CHAIR
Kristian Hammond - University of Chicago

We are pleased to invite contributions for the Second International Conference on AI Planning Systems, to be held at The University of Chicago, June 15th - 17th, 1994.

This conference will be aimed at bringing together researchers attacking different aspects of the planning problem and related issues. In addition to AI researchers, others working on planning-related issues are also encouraged to contribute and attend. Of special interest are papers discussing the integration of differing approaches to planning or the integration of planning and other AI technologies.

Topics of Interest Include:

APPLICATIONS -Empirical studies of existing planning systems; domain-specific techniques; heuristic techniques; scheduling systems.

ARCHITECTURES - Real-time support for planning and control; mixed-initiative planning and user interfaces.

ENVIRONMENTAL AND TASK MODELS - Analyses of the dynamics of environments, tasks, and domains with regard to different models of planning and execution.

FORMAL MODELS - Reasoning about knowledge, action, and time; search methods and analysis of algorithms; formal characterization of existing planners.

INTELLIGENT AGENCY - Resource-bound reasoning; distributed problem solving; integrating reaction and deliberation.

LEARNING - Learning in the context of planning and execution; learning new plans and operators.

MEMORY-BASED APPROACHES - Case-based planning; plan and operator learning and reuse; incremental planning.

PLANNING AND PERCEPTION - Integration of planning and perceptual systems.

PSYCHOLOGICAL AND BIOLOGICAL ISSUES - Analyses of goal-directed behavior; neurophysiological studies concerning planning; connectionist planning systems.

REACTIVE SYSTEMS - Environmentally driven devices/behaviors; reactive control; behaviors in the context of minimal representations.

ROBOTICS - Motion and path planning; planning and control; planning and perception.

REQUIREMENTS FOR SUBMISSION

TIMETABLE - The conference will take place June 15th - 17th,1994, at the University of Chicago, Chicago, Illinois. Authors must submit 5 copies of their papers (no electronic or Fax transmissions) by Tuesday December 14th, 1993. Notification of receipt will be sent to the first (or designated) author soon thereafter. Notification of acceptance or rejection will be mailed by February 18, 1994. Authors will need to provide camera ready copy by March 8, 1994.

APPEARANCE - Papers should be printed on 8.5" x 11" (or, if necessary, A4) sized paper, with 12 point type. Letter quality print is required. (Normally, dot-matrix printout will be unacceptable unless truly of letter quality. Exceptions will be made for submissions from countries where high quality printers are not widely available.) LaTeX 12pt article style will be acceptable.

TITLE PAGE - Each copy of the paper must include a title page, separate from the body of the paper. This should contain (i) Title, (ii) Names, addresses, phone numbers and email addresses of all authors, and (iii) An abstract of 100-200 words.

LENGTH - Papers should be submitted in 12 point text filling roughly 5.5" x 7.5" per page (LaTeX article style with 12 point text is acceptable). Papers should be no more than 12 pages including figures, tables, diagrams, and references. Short papers (5 pages or less) may be submitted for review as posters. All papers will be included in the conference proceedings.

DEMONSTRATIONS - Participants wanting to give computer and/or video taped demonstrations should send a two page abstract describing their contribution to the same address by February 22, 1994. These abstracts should include a separate title page with a (i) Program name and (ii) Names, addresses, phone numbers and email addresses of all authors. Demonstrations will be held in concert with the conference's poster session.

PANELS - Researchers interested in organizing panels should get hold of the program chair as soon as possible.

All submissions should be sent to:

AIPS-94
c/o Kristian Hammond
Department of Computer Science
University of Chicago
1100 East 58th Street
Chicago, IL 60637

For more information, email to:
hammond@cs.uchicago.edu