DRAFT 4/15/98

Reasoning Defeasibly about Plans

John L. Pollock
Department of Philosophy
University of Arizona
Tucson, Arizona 85721
(e-mail: pollock@arizona.edu)

Abstract

This technical report describes the construction of an experimental planner that finds plans by reasoning about them defeasibly rather than by running a search algorithm. The need for such a planner is defended in the paper "The Logical Foundations of Goal-Regression Planning".

1. Planning Agents

Practical applications of AI planning theory have only occurred in narrowly circumscribed domains in wich the goals are fixed and all the relevant information can be precompiled and supplied to the planner. The planner then runs a program that searches the space of possible plans (relative to the given information) until it finds a plan whose execution is guaranteed to achieve the goals. In such "applied planning", a planner is a tool used by a human being, and in order to use the tool effectively the human must prepare the ground very carefully, being sure to give the planner all the knowledge needed to solve the planning problem.

Contemporary AI planning theory is based upon *algorithmic planners*. Given a planning problem, an algorithmic planner runs a program that systematically searches the space of possible plans until it returns one that purports to solve the problem. The sense in which the planner is algorithmic is that it executes an effective computation, i.e., the set of pairs ⟨problem,solution⟩ that characterize the planner is recursively enumerable.

One of the ideals to which AI aspires is the construction of autonomous rational agents capable of maneuvering through a complex, variable, and often uncooperative environment. A special case of this is the attempt to build a system modeling human rationality. Planning will be an essential ingredient in any such agent. However, the planning problem faced by such an agent contrasts in important ways with the kind of applied planning problem that is solved by current AI planning technology. The most obvious difference is that, in sharp contrast to applied planning, it cannot be assumed that a planning agent has exactly the knowledge it needs to solve a planning problem. An autonomous agent must build its own knowledge base. The system designer can get things started by providing background knowledge, but the agent must be provided with cognitive machinery enabling its knowledge base to grow and evolve as it gains experience of its environment, senses its immediate surroundings, and reasons about the consequences of beliefs it already holds. The more complex the environment, the more the autonomous agent will have to be self-sufficient for knowledge acquisition. I have distinguished between practical cognition and epistemic cognition. The principal function of epistemic cognition in an autonomous agent is to provide the information needed for practical cognition. As such, the course of epistemic cognition is driven by practical interests. Rather than coming to the

This work was supported by NSF grant no. IRI-9634106.

planning problem equipped with all the knowledge required for its solution, the planning problem itself directs epistemic cognition, focusing epistemic endeavors on the pursuit of information that will be helpful in solving current planning problems.

Paramount among this information is knowledge about what will happen if certain actions are taken under certain circumstances. Sometimes the agent already knows what will happen, but often it has to figure it out. At the very least this will require reasoning from current knowledge. In many cases it will require the empirical acquisition of new knowledge that cannot be obtained just by reasoning from what is already known. For example, in order to construct a plan the planning agent may have to find out what time it is, and it may be able to do that only by examining the world in some way (e.g., it may have to go into the next room and look at the clock). In general, such empirical investigations are carried out by performing actions (not just by reasoning). Figuring out what actions to perform is a matter of engaging in further planning. The agent acquires the epistemic goal of acquiring certain information, and then plans for how to accomplish that. So planning drives epistemic investigation which may in turn drive further planning. It follows that an essential characteristic of planning agents is that planning and epistemic cognition are interleaved. Unlike applied planning, it is impossible to require of a planning agent capable of functioning in realistically complex environments that it acquire all the requisite knowledge before beginning the plan search.

Now let us apply this to the question whether human beings (and other rational agents) can perform their planning by implementing planning algorithms. Goal-regression planners construct separate plans for the conjuncts of a conjunctive goal, and then merge them into a single plan. In order to do that, they must first determine that the separate plans do not interfere with each other destructively. Such destructive interference consists of some steps of the merged plan making a subgoal of one of the plans false before it can be used for achieving its purpose. It will only be possible to build an algorithmic planner will only be possible if the set of destructive interferences is effectively computable, i.e., recursive. If the set of destructive interferences is not effectively computable, the planner will not be able to determine whether two plans can be merged or, when there is destructive interference, whether it can be repaired.

In order for destructive interference to be computable, it must be computable whether a particular condition (the negation of a precondition of one of the plan steps) is a consequence of an action under specifiable circumstances. Standard AI planning systems accomplish this by assuming that all relevant planning-conditionals are contained in a database at the time planning begins, and hence the consequences of actions can be determined by simply looking them up in a table (using unification). Such planners do no reasoning or very little reasoning about the consequences of actions, relying instead on precompiled knowledge.[1]

By contrast, autonomous planning agents cannot rely on precompiled knowledge. They must engage in genuine reasoning about the consequences of actions, and we should not expect that reasoning to be any simpler than general epistemic reasoning. Realistically, epistemic reasoning must be defeasible, which makes the set of conclusions at best Δ_2.[2] But even if we could construct an agent that did only first-order deductive reasoning, the set of conclusions is not effectively computable—it is recursively enumerable. Even for such an unrealistically oversimplified planner, destructive interference will not be computable—the set of destructive interferences will be only r.e. This means that when the planning algorithm computes plans for the conjuncts of a conjunctive goal and then considers whether they can be merged without destructive interference, the reasoning required to find any particular destructive interference may take indefinitely long, and if there is no destructive interference, there will be no point at which the planner can draw the conclusion that there is none simply on the grounds that none has been found. Thus the planning algorithm will bog down at this point and will never be able

---

[1] This originated with STRIPS (Fikes and Nilsson 1971), which built the requisite planning-conditionals into the plan operators themselves. Subsequent AI planners have followed suite.

[2] For a discussion of this, see Pollock (1995), chapter three.

to produce the merged plan for the conjunctive goal.[3]

If destructive interference is not computable, how can a planner get away with dividing conjunctive goals into separate conjuncts and planning for each conjunct separately? The key to this problem emerges from considering how human beings solve it. Humans assume defeasibly that the separate plans do not destructively interfere with one another, and so infer defeasibly that the merged plan is a good plan for the conjunctive goal. Having made this defeasible inference, human planners then look for destructive interference that would defeat it, but they do not regard it as essential to establish that there is no destructive interference before they make the inference. And if, at the time plan execution is to begin, no destructive interference has been discovered, then we humans go ahead and execute the plan despite the fact that we have not *proven conclusively* that there is no destructive interference.

One may be tempted to suppose that human beings are making an unreasonable leap of faith here, and that a more rational agent would postpone plan execution until it has been established that there is no destructive interference. However, the logic of the epistemic search for destructive interference makes that logically impossible. Given a logically complex knowledge base, there will not, in general, be a point at which an agent can conclude with certainty that there is no destructive interference within a plan, so an agent that required such certainty would be unable to execute any of its plans.

The upshot of this is that a rational agent operating in a realistically complex environment must make defeasible assumptions in the course of its planning, and then be prepared to change its planning decisions later if subsequent epistemic reasoning defeats those defeasible assumptions. In other words, the reasoning involved in planning must be a species of defeasible reasoning. *Planning in autonomous agents cannot be done algorithmically.*

The general way goal-regression planning must work is by splitting conjunctive goals into their conjuncts and planning for them separately, and then merging the plans for the individual conjuncts into a combined plan for the conjunctive goal. The planning agent will infer defeasibly that the merged plan is a solution to the planning problem. A defeater for this defeasible inference consists of discovering that the plan contains destructive interference. Whenever a defeasible reasoner makes a defeasible inference, it must adopt interest in finding defeaters, so in this case the agent will adopt interest in finding destructive interference. Finding such interference should lead the agent to try various ways of repairing the plan to eliminate the interference, and then lead to a defeasible inference that the repaired plan is a solution to the planning problem. The tentative conclusion being adopted is that the plan will achieve its goal. Goal-regression planning becomes a form of epistemic reasoning to the effect that if a plan is executed then it is defeasibly reasonable to expect the goal to be achieved.

The theoretical foundations of such a defeasible goal-regression planner are developed in my [1998a]. The purpose of this paper is to document the construction of a planner implementing this theory within the OSCAR architecture for autonomous rational agents. For the complete justification of the defeasible inference-schemes discussed in this paper, the reader is referred to my [1998a].

2. The OSCAR Architecture

OSCAR is an architecture for rational agents, based upon an implemented general-purpose defeasible reasoner.[4] The architecture divides cognition into epistemic cognition (cognition about

---

[3] Notice that a similar problem arises in applying PROPOSE-NULL-PLAN, which may require an indeterminate amount of reasoning to determine that the subgoal is already true. If the requisite reasoning is not at least r.e., then the planning cannot be algorithmic.

[4] The theory underlying OSCAR is described in detail in my [1995]. The implementation is described in *The OSCAR Manual.* Both the manual and the implemented system can be downloaded from

what to believe) and practical cognition (cognition about what to do). Epistemic cognition is interest-driven in the sense that it proceeds bidirectionally. The reasoner reasons forwards from perceptual input, and backwards from queries it is trying to answer. Initially, queries are posed by practical cognition, and then as reasoning proceeds, new epistemic interests are generated by reasoning backwards from the initial ones.

The architecture for epistemic cognition is diagramed in figure 1. Reasoning begins with perceptual input, and the ultimate-epistemic-interests. The latter are queries that are passed to epistemic cognition from practical cognition, without being derived from other interests by backwards reasoning. Conclusions are stored as nodes ("inference-nodes") in the *inference-graph*, which records both inference-relations between conclusions and defeat-relations between conclusions. Interests are stored in the *interest-graph*, which records their inference-relations. Backwards reasoning from an interest creates *interest-links*, which link a set of interests to the original interest. The significance of an interest-link is that if conclusions are drawn discharging the link-interests, then a conclusion discharging the original interest can be inferred from those conclusions.

Reasoning is mediated by reasons. Forwards-directed reasons mediate forwards-reasoning (reasoning from conclusions to conclusions), and backwards-directed reasons mediate backwards-reasoning (reasoning from interests to interests). OSCAR appears to be unique in that its forwards-directed reasons are entirely distinct from its backwards-directed reasons. For example, ADJUNCTION is a backwards-directed reason, instructing the reasoner that if it is interested in (P & Q) then it should become interested in P and in Q. SIMPLIFICATION is a forwards-directed reason instructing the reasoner that if it has concluded (P & Q) then it should infer P and Q from that conclusion. This structure of reasoning makes OSCAR very efficient.[5]

Reasoning proceeds in terms of reasons. *Backwards-reasons* are used in reasoning backwards, and *forwards-reasons* are used in reasoning forwards. Forwards-reasons are data-structures with the following fields:

- reason-name.
- forwards-premises — a list of forwards-premises.
- backwards-premises — a list of backwards-premises.
- reason-conclusions — a list of formulas.

---

http://www.u.arizona.edu/~pollock/.

[5] In a recent comparison of OSCAR with more standard resolution-based theorem provers, OSCAR proved to be twelve times faster than Otter, a highly regarded state-of-the-art resolution-based theorem prover. The comparison was arranged by Geoff Sutcliffe, librarian of the TPTP library of theorem proving problems. It is noteworthy that OSCAR did so well despite the fact that OSCAR only does theorem proving as a side effect of its general reasoning, and is burdened with a great deal of extra machinery that is required for defeasible reasoning. In addition, Otter is written in C while OSCAR is written in LISP. This should give Otter an advantage of as much as an order of magnitude .

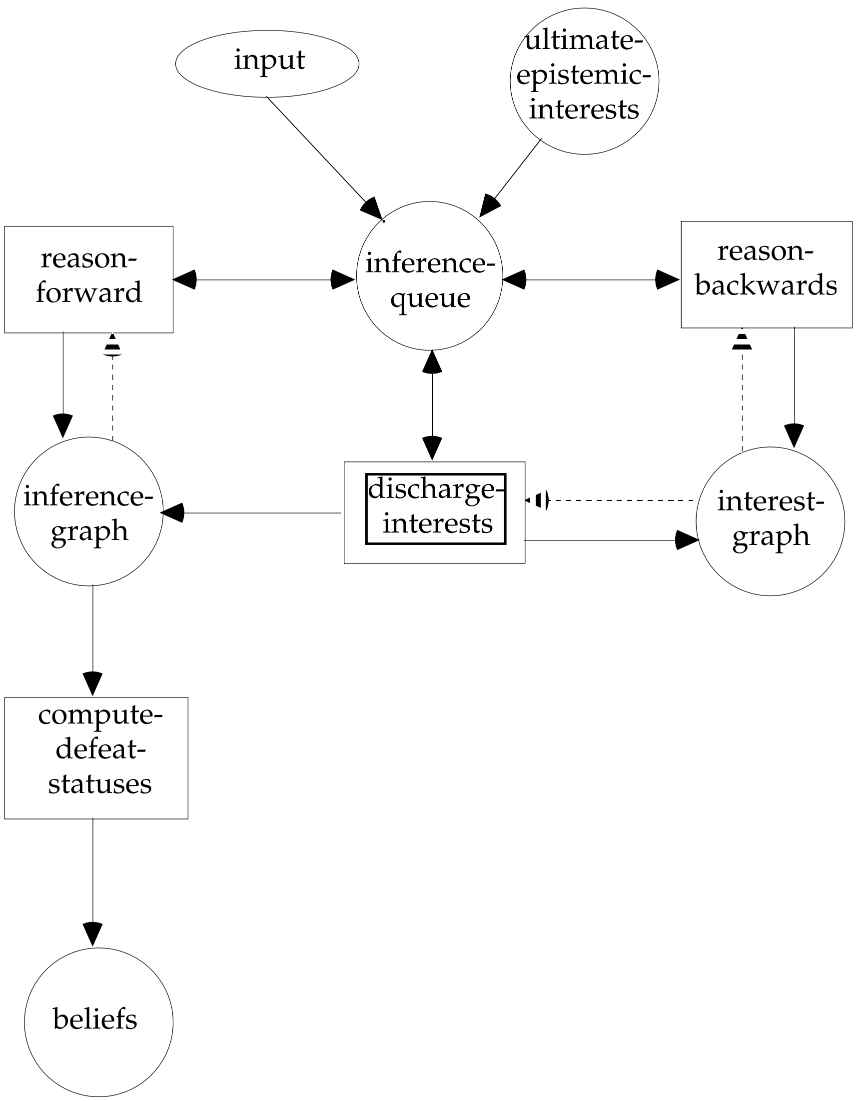

Figure 1. The defeasible reasoner

- conclusions-function — an optional function used for computing the conclusions of an inference made in accordance with the reason. If the function is not supplied, the conclusions are computed by instantiating the reason-conclusions.
- defeasible-rule — T if the reason is a defeasible reason, NIL otherwise.
- reason-variables — variables used in pattern-matching to find instances of the reason-premises.
- reason-strength — a real number between 0 and 1, or an expression containing some of the reason-variables and evaluating to a number.
- reason-description — an optional string describing the reason.

Forwards-premises are data-structures encoding the following information:

- fp-formula — a formula.

- fp-kind — :inference, :percept, or :desire (the default is :inference)
- fp-condition — an optional constraint that must be satisfied by an inference-node for it to instantiate this premise.
- clue? — explained below.

Similarly, *backwards-premises* are data-structures encoding the following information:
- bp-formula
- bp-kind

Backwards-reasons will be data-structures encoding the following information:
- reason-name.
- forwards-premises.
- backwards-premises.
- reason-conclusions — a list of formulas.
- conclusions-function.
- reason-variables — variables used in pattern-matching to find instances of the reason-premises.
- strength — a real number between 0 and 1, or an expression containing some of the reason-variables and evaluating to a number.
- defeasible-rule — T if the reason is a defeasible reason, NIL otherwise.
- reason-condition — a condition that must be satisfied by the sequent of interest before the reason is to be deployed.

Simple forwards-reasons have no backwards-premises, and *simple backwards-reasons* have no forwards-premises. Given inference-nodes that instantiate the premises of a simple forwards-reason, the reasoner infers the corresponding instance of the conclusions. Similarly, given an interest that instantiates the first conclusion of a simple backwards-reason, the reasoner adopts interest in the corresponding instances of the backwards-premises. Given inference-nodes that discharge those interests, an inference is made to the conclusions from those inference-nodes.

In deductive reasoning, with the exception of a rule of reductio ad absurdum, we are unlikely to encounter any but simple forwards- and backwards-reasons.[6] However, the use of backwards-premises in forwards-reasons and the use of forwards-premises in backwards-reasons provides an invaluable form of control over the way reasoning progresses. This will be illustrated below. *Mixed* forwards- and backwards-reasons are those having both forwards- and backwards-premises. Given inference-nodes that instantiate the forwards-premises of a mixed forwards-reason, the reasoner does not immediately infer the conclusion. Instead the reasoner adopts interest in the corresponding instances of the backwards-premises, and an inference is made only when those interests are discharged. Similarly, given an interest instantiating the first conclusion of a mixed backwards-reason, interests are not immediately adopted in the backwards-premises. Interests in the backwards-premises are adopted only when inference-nodes are constructed that instantiate the forwards-premises.

Reasons are most easily defined in OSCAR using the macros DEF-FORWARDS-REASON and DEF-BACKWARDS-REASON:

(def-forwards-reason *symbol*
  :forwards-premises *list of formulas optionally interspersed with expressions of the form (:kind ...) or (:condition ...)*
  :backwards-premises *list of formulas optionally interspersed with expressions of the form (:kind ...) or (:condition ...)*
  :conclusions *list of formulas*
  :conclusions-function
  :strength *number or a an expression containing some of the reason-variables and evaluating to a number.*

[6] This is discussed at greater length in Chapter Two of my [1995a].

```
   :variables list of symbols
   :defeasible? T or NIL (NIL is the default)
   :description an optional string (quoted) describing the reason)

(def-backwards-reason symbol
   :conclusions  list of formulas
   :conclusions-function
   :forwards-premises list of formulas optionally interspersed with expressions of the form (:kind ...) or (:condition ...)
   :backwards-premises  list of formulas optionally interspersed with expressions of the form (:kind ...) or (:condition ...)
   :condition  this is a predicate applied to the binding produced by the target sequent
   :strength number or an expression containing some of the reason-variables and evaluating to a number.
   :variables list of symbols
   :defeasible? T or NIL (NIL is the default)
   :description an optional string (quoted) describing the reason)
```

The use of these macros will be illustrated below.

3. Goal-Regression Planning

The objective is to build an agent that can plan by reasoning about plans. This involves reasoning about the consequences of actions under various circumstances. I assume that the requisite causal information is given in *planning-conditionals* having the form "(C & A) $\Rightarrow$ G" and meaning "Doing A under circumstances in which C is true would result in G being true". In this paper I will make the standard assumption that C and G are either literals or conjunctions of literals. The theory of goal-regression planning developed in my [1998a] is based upon the following definition of the "expectable-results" of a sequence of actions $\langle A_1,...,A_n\rangle$:

> Where *start-state* is a state of affairs and *conditionals* is a set of planning-conditionals, P is an **expectable-result** of $\langle A_1,...,A_n\rangle$ relative to *start-state* and *conditionals* iff either:
> (i) $n = 0$ and P is true in *start-state*; or
> (ii) $n > 0$ and *conditionals* contains a conditional (A_n & C) $\Rightarrow$ P such that C is a temporally-projectible expectable-result of $\langle A_1,...,A_{n-1}\rangle$; or
> (iii) $n > 0$, P is a temporally-projectible expectable-result of $\langle A_1,...,A_{n-1}\rangle$, and *conditionals* does not contain a conditional of the form (A_n & C) $\Rightarrow$ ~Q such that Q is either P or a conjunct of P and C is a temporally-projectible expectable-result of $\langle A_1,...,A_{n-1}\rangle$; or
> (iv) $n > 0$ and P is a conjunction whose conjuncts are expectable-results of $\langle A_1,...,A_n\rangle$.

Plans will be nonlinear, in the sense that the plan-steps may be only partially ordered, so we must augment the above definition as follows:

> P is an expectable-result of a partial-order plan iff it is an expectable-result of every linearization of the plan.

A linearization of a plan is a total ordering of the plan-steps consistent with the ordering-constraints built into the plan. The objective of plan reasoning is to construct a plan whose execution has the goal as an expectable-result. A sound and complete set of reasoning-schemes was developed in my [1998a], and will be implemented here.

4. Reasoning about Plans

The objective is to build an agent that can plan by reasoning about plans. This involves reasoning about the consequences of actions under various circumstances. The requisite causal information is given in the form of planning-conditionals. A minimal amount of reasoning about these conditionals can be accommodated by giving OSCAR the following reason-schemas:

```
(def-backwards-reason =>-neg1
   :conclusions  "(P => ~(Q & R))"
   :condition (and (not (interest-variable Q)) (not (interest-variable R)))
   :backwards-premises
      "(define -Q (neg Q))"
     "(P => -Q)"
   :variables  P Q -Q R)

(def-backwards-reason =>-neg2
   :conclusions  "(P => ~(Q & R))"
   :condition (and (not (interest-variable Q)) (not (interest-variable R)))
   :backwards-premises
      "(define -R (neg R))"
     "(P => -R)"
   :variables  P Q R -R)

(def-forwards-reason SIMPLIFY-=>
   :forwards-premises "(P => (Q & R))"
   :conclusions  "(P => Q)" "(P => R)"
   :variables P Q R)

(def-backwards-reason =>-ADJUNCTION
   :conclusions "(P => (Q & R))"
   :backwards-premises  "(P => Q)" "(P => R)"
   :variables P Q R)
```

Notice the "define" premises in ⇒-neg1 and ⇒-neg2. Such a premise is used to set the value of one of the variables. If *Q* has the form ~*R* then (NEG *Q*) is *R*, otherwise (NEG *Q*) is ~*Q*.

I will describe the construction of a goal-regression planner within OSCAR. The plans produced by this planner are modelled on those produced by SNLP and UCPOP.[7] Plans will be nonlinear, in the sense that the plan-steps may be only partially ordered. The only ordering imposed is that required by causal-links and that contained in the explicit ordering-constraints. The interpretation of a nonlinear plan is that every linearization of the plan (i.e., every total ordering of the plan-steps consistent with the explicit ordering-constraints and causal-links) is expected to achieve the goals of the plan.

Plans contain two kinds of ordering-constraints. It can be required that one plan-node is executed before another, or it can be required that one plan-node is not executed between two other plan-nodes.[8] *Plans* will be data-structures with the following fields:

- plan-number — *a number used for identification*
- plan-steps — *a list of plan-nodes*
- plan-goal — *a formula*
- causal-links — *a list of causal-links*
- before-nodes — *pairs of plan-nodes, where the first is required to be executed before the second*
- not-between — *triples of plan-nodes, where the first is required to not be executed between the second and third.*

Algorithmic planners typically also store variable-bindings in plans. However, this is not necessary

[7] SNLP is described in McAllester and Rosenblitt [1991]. UCPOP is described in Penberthy and Weld [1992] and Weld [1994].

[8] The latter is a slight generalization of UCPOP's ordering constraints.

when the planning is done by reasoning, because the reasoner automatically handles the variable-bindings for us.

Plan-nodes will be data-structures with the following fields:

- plan-node-number
- plan-node-action

A plan can contain multiple plan-nodes with the same action, signifying that the action must be repeated.

Finally, *causal-links* will be data-structures with the following fields:

- causal-link-number
- causal-link-root—*a plan-node*
- causal-link-goal—*a formula*
- causal-link-target—*a plan-node.*

The causal-link "*n --g--> n**" signifies that the root *n* aims to achieve the goal *g* which is a precondition for the target *n** to achieve whatever goal it aims to achieve. (call-set *node plan*), the call-set of *node* relative to *plan*, is the set of causal-links in *plan* having *node* as their target.

In order to construct causal-links for preconditions that are satisfied simply by being true initially, it is convenient to have a dummy plan-node corresponding to the start-state. This is called **start**, and is plan-node 0. Similarly, each plan will terminate with the dummy node **finish**.

To illustrate, given the following information:

```
(at-library Horatio)
(all x)(((at-library x) & (ask-librarian x)) ⇒ (know-beethoven-birthday x))
(all x)(((at-clock x) & (read-clock x)) ⇒ (know-time x))
(all x)(((at-library x) & (go-to-clock x)) ⇒ ((at-clock x) & ~(at-library x)))
```

and the goal "((know-beethoven-birthday Horatio) & (know-time Horatio))", we would like the planner to find the following plan:

```
PLAN-STEPS:
  (1) ( ask-librarian Horatio)
      causal-links:
        0 --( at-library Horatio)--> 1
  (2) ( go-to-clock Horatio)
      causal-links:
        0 --( at-library Horatio)--> 2
      ordering-constraints:
        2 > 1
  (3) ( read-clock Horatio)
      causal-links:
        2 --( at-clock Horatio)--> 3
      ordering-constraints:
        3 > 2
  GOAL: (( know-beethoven-birthday Horatio) & ( know-time Horatio))
      established by:
        1 --> ( know-beethoven-birthday Horatio)
        3 --> ( know-time Horatio)
```

This plan can be represented graphically as in figure 2. The solid arrows indicate causal-links, and the dotted arrow indicates an ordering-constraint not generated by a causal-link.

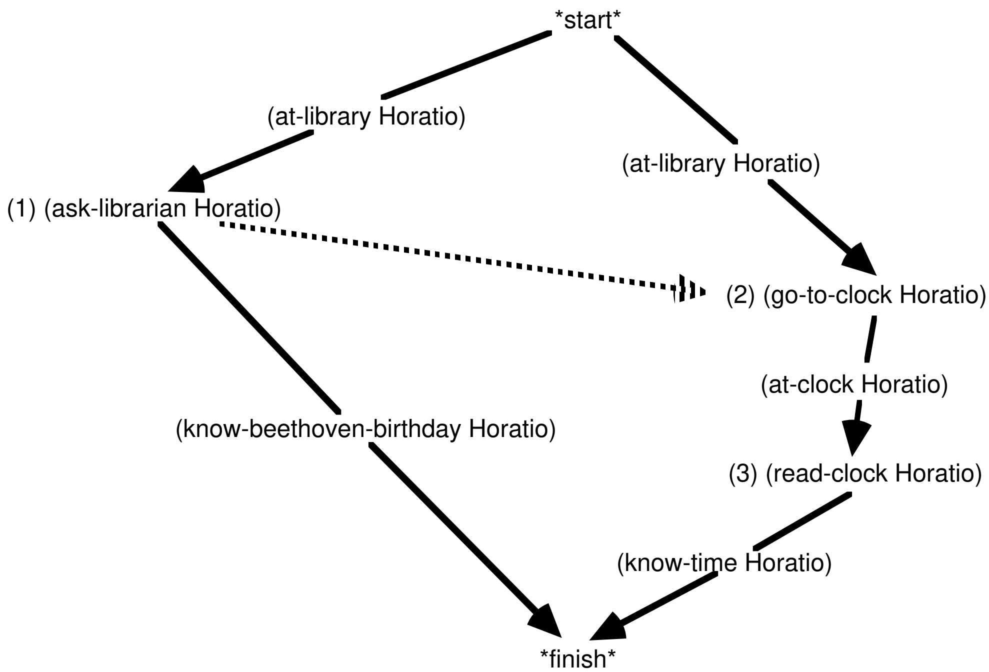

Figure 2. A nonlinear plan.

A *linearization* of a plan is a total order of its plan-steps consistent with its ordering-constraints. The *preceding-nodes* or *succeeding-nodes* of a plan-node are those that precede or succeed it in every linearization. The *possibly-preceding-nodes* or *possibly-succeeding-nodes*, of a node are those for which there is a linearization in which they precede it or succeed it. The *possibly-intermediate-nodes* of a pair of plan-nodes are those that are intermediate between them in some linearization.

Goal Regression

OSCAR will find plans by attempting to answer queries of the form (?*p*)(plan-for *p goal goals nodes nodes-used links*), meaning "find a *p* such that *p* is a plan for *goal* relative to the set of superordinate goals *goals* and the used nodes *nodes-used* and defeated links *links*". (The *goals*, *nodes-used* and *links* variables are used to make plan-search more efficient, and will be discussed in section seven.)

In goal-regression planning, the planner adopts interest in finding a plan for a goal. Given a conditional of the form "((*precondition* & *action*) ⇒ *goal*)", the reasoner adopts interest in finding a "subplan" for the precondition, and if such a subplan is found, a plan is constructed for the original goal by adding a plan-node whose action is *action* to the end of the subplan. If the precondition is already satisfied, then a null-plan (a plan with no plan-steps) is taken to achieve the precondition. There can be many null-plans, because although they have no plan-steps they may have different goals. Reasoning leading to the conclusion that the subgoal is achieved by a null-plan is accomplished by the following reasoning-schema:

PROPOSE-NULL-PLAN

Given an interest in finding a plan for achieving *goal*, if *goal* is already true, infer defeasibly that a null-plan will achieve *goal*.

This can be implemented as the following backwards-reason:

```
(def-backwards-reason NULL-PLAN
```

```
  :conclusions "(plan-for plan goal goals nodes nodes-used links)"
  :condition (interest-variable plan)
  :backwards-premises
     "goal"
     "(define plan (null-plan goal links))"
  :variables goal plan goals nodes nodes-used links)
```

The variables *nodes*, *nodes-used*, and *links* will be explained in section seven. As remarked above, the "define" premise works like a "let", temporarily binding the variable *plan* to the value of (*null-plan goal*), which is a plan with with no plan-steps and with a goal-node that is a plan-node having *goal* as its plan-node-action, and having a single causal-link **start* --goal--> goal-node*.

The fundamental operation of goal-regression is accomplished by the following reasoning-schema:

GOAL-REGRESSION

Given an interest in finding a plan for achieving G, adopt interest in finding planning-conditionals $(C \,\&\, A) \Rightarrow G$ having G as their consequent. Given such a conditional, adopt an interest in finding a plan for achieving C. If a plan *subplan* is proposed for achieving C, construct a plan by (1) adding a new step to the end of *subplan* where the new step prescribes the action A, (2) ordering the new step after all steps of *subplan*, and (3) adjusting the causal-links appropriately. Infer nondefeasibly that the new plan will achieve G.

This can be implemented as the following backwards-reason:

```
(def-backwards-reason GOAL-REGRESSION
  :conclusions "(plan-for plan goal goals nodes nodes-used links)"
  :condition (and (interest-variable plan) (not (mem goal goals)) (null nodes-used))
  :backwards-premises
     "((precondition & action) => goal)"
     (:condition (not (mem precondition goals)))
     "(define new-goals (cons goal goals))"
     "(plan-for subplan precondition new-goals nodes nodes-used links)"
     "(define plan (extend-plan action goal subplan links))"
     (:condition (not (null plan)))
  :variables precondition action goal plan subplan goals new-goals nodes nodes-used links)
```

The function (EXTEND-PLAN *action goal subplan links*) constructs a new plan-node *node* having *action* as its action and *precondition* as its goal, and then constructs a new plan by adding this plan-node to *subplan*. The causal-links for the new node are constructed by taking the causal-links in the call-set of **finish** in *subplan*, and replacing their causal-link-targets by *node*. Then a new causal-link is added linking *node* to **finish** and having *goal* as its causal-link-goal.

To avoid infinite regresses in the backwards-reasoning, the condition is imposed that *precondition* is not already a member of *goals*. This condition will be discussed at length in section seven, along with the variables *nodes, nodes-used*, and *links*.

To illustrate the reasoning with a very simple example, consider the following:

```
======================================================================
Backwards-substantive-reasons:
        null-plan
        goal-regression

Goal-state:
        know-time

Given:
        at-library  :  with justification = 0.99
        ((at-clock & read-clock) ⟹ know-time)  :  with justification = 0.99
        ((at-library & go-to-clock) ⟹ at-clock)  :  with justification = 0.99
```

```
===================================================================
THE FOLLOWING IS THE REASONING INVOLVED IN THE SOLUTION

 # 1
 at-library
 given
 This discharges interest 9
 # 2
 ((at-clock & read-clock) ⟹ know-time)
 given
 # 3
 ((at-library & go-to-clock) ⟹ at-clock)
 given
                                    # 4
                                    interest: (plan-for ^@y0 know-time)
                                    This is of ultimate interest
                                    # 6
                                    interest: (plan-for ^@y1 at-clock)
                                    For interest 4 by goal-regression
                                    This interest is discharged by node 5
                                    # 7
                                    interest: (plan-for ^@y2 at-library)
                                    For interest 6 by goal-regression
                                    This interest is discharged by node 4
                                    # 9
                                    interest: at-library
                                    For interest 7 by null-plan
                                    This interest is discharged by node 1
 # 4
 (plan-for <plan 1> at-library)
 Inferred by:
              support-link #4 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #7

Plan #1 has been constructed:
    GOAL:   at-library
           established by:
             0 --> at-library

 # 5
 (plan-for <plan 2> at-clock)
 Inferred by:
              support-link #5 from { 3 , 4 } by goal-regression
 This node is inferred by discharging a link to interest #6

Plan #2 has been constructed:
    PLAN-STEPS:
      (2) go-to-clock
           causal-links:
             0 --at-library--> 2
    GOAL:   at-clock
           established by:
             2 --> at-clock

 # 6
 (plan-for <plan 3> know-time)
 Inferred by:
              support-link #6 from { 2 , 5 } by goal-regression
 This node is inferred by discharging a link to interest #4

Plan #3 has been constructed:
    PLAN-STEPS:
      (2) go-to-clock
           causal-links:
             0 --at-library--> 2
      (3) read-clock
           causal-links:
             2 --at-clock--> 33
             3 > 2
    GOAL:   know-time
           established by:
             3 --> know-time

             ==========================================
             Justified belief in (plan-for <plan 3> know-time)
             with undefeated-degree-of-support 0.99
             answers #<Query 1: (? plan)(plan-for plan know-time)>
             ==========================================
```

The plan found is the right half of the plan diagrammed in figure 2.

Splitting Conjunctive Goals

The operations PROPOSE-NULL-PLAN and GOAL-REGRESSION do not by themselves constitute a complete description of goal-regression planning. The subgoals generated by GOAL-REGRESSION will usually be conjunctions. For example, if my goal is to light a fire, I may observe that I could do so by lighting a match provided I have a match and I have tinder. GOAL-REGRESSION will thus generate the conjunctive subgoal *I have a match and I have tinder*. We will generally be unable to make further progress in our plan construction by applying GOAL-REGRESSION once more to such a conjunctive subgoal (C_1 & C_2). To do so would require our having a planning-conditional of the form (A & C) $\Rightarrow$ (C_1 & C_2). But it is rare that we will have a single planning-conditional like this that will achieve both conjuncts of a conjunctive subgoal. The most we can generally hope for is to have two separate planning-conditionals (A & C) $\Rightarrow C_1$ and (A^* & C^*) $\Rightarrow C_2$, which will allow us to construct separate subplans for the individual conjuncts. Given subplans for achieving each conjunct, we can then attempt to construct a plan for achieving the conjunction by merging the plans for the conjuncts. Given two plans $plan_1$ and $plan_2$, let $plan_1$ + $plan_2$ be the plan that results from combining the plan-steps, causal-links, and ordering-constraints of each. Then we can plan for conjunctive goals by using the following operation:

SPLIT-CONJUNCTIVE-GOAL

Given an interest in finding a plan for achieving a conjunctive goal (G_1 & G_2), adopt interest in finding plans $plan_1$ for G_1 and $plan_2$ for G_2. If such plans are proposed, infer defeasibly that $plan_1$ + $plan_2$ will achieve (G_1 & G_2).

This is implemented as follows:

```
(def-backwards-reason SPLIT-CONJUNCTIVE-GOAL
   :conclusions
   "(plan-for plan& (goal1 & goal2) goals nodes nodes-used links)"
   "(merged-plan plan& plan1 plan2 goals)"
     (:defeasible? nil)
   :condition  (interest-variable plan&)
   :backwards-premises
      "(define new-goals (cons (conj goal1 goal2) goals))"
      "(plan-for plan1 goal1 new-goals nodes nodes-used links)"
      "(plan-for plan2 goal2 new-goals nodes nodes-used links)"
      (:condition (or (plan-steps plan1) (plan-steps plan2)))
      "(define plan& (merge-plans plan1 plan2 goal1 goal2))"
      (:condition (not (null plan&)))
   :defeasible? t
   :variables  goal1 goal2 plan1 plan2 goals new-goals plan& nodes nodes-used links)
```

Note that this reason produces two conclusions. The first has the form "(plan-for plan& (goal1 & goal2) goals nodes nodes-used links)", and is drawn defeasibly. The second has the form "(merged-plan plan& plan1 plan2 goals)". This conclusion is drawn so that the reasoner has a record of how *plan&* was constructed. This conclusion should be drawn non-defeasibly. This is accomplished by following it with the qualification "(:defeasible? nil)", which has the effect of overriding the ":defeasible? t" specification for the reason-schema as a whole.

The function (MERGE-PLANS *plan1 plan2 goal1 goal2*) constructs a new plan *plan&* whose plan-steps consist of all the plan-steps of *plan1* and *plan2*. The plan-goal of *plan&* is the conjunction of *goal1* and *goal2*.

To illustrate consider a variant of the planning problem of figure 2 where going to the clock does not require leaving the library:

```
=================================================================

                              OSCAR_3.25         4/15/1998          15:28:25
                                       Non-Linear-Planner-31

Forwards-substantive-reasons:
         simplify-=>

Backwards-substantive-reasons:
         null-plan
         goal-regression
         split-conjunctive-goal

Goal-state:
         know-beethoven-birthday
         know-time

Inputs:

Given:
         at-library  :  with justification = 0.99
         ( (at-library & ask-librarian) => know-beethoven-birthday)  :  with justification = 0.99
         ( (at-clock & read-clock) => know-time)  :  with justification = 0.99
         ( (at-library & go-to-clock) => at-clock)  :  with justification = 0.99

======================================================================
THE FOLLOWING IS THE REASONING INVOLVED IN THE SOLUTION

  # 1
  at-library
  given
  This discharges interest 13
  # 2
  ( (at-library & ask-librarian) => know-beethoven-birthday)
  given
  This discharges interest 10
  # 3
  ( (at-clock & read-clock) => know-time)
  given
  This discharges interest 15
  # 4
  ( (at-library & go-to-clock) => at-clock)
  given
  This discharges interest 18
                                   # 5
                                   interest: ( plan-for ^@y0 (know-beethoven-birthday & know-time) nil nil nil nil)
                                  # 6
                                   interest: ( plan-for ^@y1 know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil nil)
                                   For interest 5 by split-conjunctive-goal
                                   This interest is discharged by node 8
                                   # 10
                                   interest: ( (^@y4 & ^@y5) => know-beethoven-birthday)
                                   For interest 6 by goal-regression
                                   This interest is discharged by node 2
                                   # 11
                                   interest: ( plan-for ^@y6 at-library ( know-beethoven-birthday (know-beethoven-birthday & know-time))
                                         nil nil nil)
                                   For interest 6 by goal-regression using node 2
                                   This interest is discharged by node 5
                                   # 13
                                   interest: at-library
                                   For interest 11 by null-plan
                                   For interest 19 by null-plan
                                   This interest is discharged by node 1
                                   This is of ultimate interest
  # 5
  ( plan-for <plan 1> at-library ( know-beethoven-birthday (know-beethoven-birthday & know-time)) nil nil nil)
  Inferred by:
               support-link #5 from { 1 } by null-plan
  This node is inferred by discharging a link to interest #11

Plan #1 has been constructed
       GOAL: at-library
           established by:
             0 --> at-library

  # 8
```

```
 ( plan-for <plan 2> know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
              support-link #8 from { 2 , 5 } by goal-regression
 This node is inferred by discharging a link to interest #6

Plan #2 has been constructed
   PLAN-STEPS:
     (1) ask-librarian
         causal-links:
           0 --at-library--> 1
     GOAL: know-beethoven-birthday
         established by:
           1 --> know-beethoven-birthday

                               # 14
                               interest: ( plan-for ^@y9 know-time ( (know-beethoven-birthday & know-time)) nil nil nil)
                               For interest 5 by split-conjunctive-goal using node 8
                               This interest is discharged by node 15
                               # 15
                               interest: ( (^@y10 & ^@y11) => know-time)
                               For interest 14 by goal-regression
                               This interest is discharged by node 3
                               # 16
                               interest: ( plan-for ^@y12 at-clock ( know-time (know-beethoven-birthday & know-time)) nil nil nil)
                               For interest 14 by goal-regression using node 3
                               This interest is discharged by node 12
                               # 18
                               interest: ( (^@y13 & ^@y14) => at-clock)
                               For interest 16 by goal-regression
                               This interest is discharged by node 4
                               # 19
                               interest: ( plan-for ^@y15 at-library ( at-clock know-time (know-beethoven-birthday & know-time)) nil nil
                                    nil)
                               For interest 16 by goal-regression using node 4
                               This interest is discharged by node 9
 # 9
 ( plan-for <plan 1> at-library ( at-clock know-time (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
              support-link #9 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #19

Plan #1 has been constructed
     GOAL: at-library
         established by:
           0 --> at-library

 # 12
 ( plan-for <plan 3> at-clock ( know-time (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
              support-link #12 from { 4 , 9 } by goal-regression
 This node is inferred by discharging a link to interest #16

Plan #3 has been constructed
   PLAN-STEPS:
     (2) go-to-clock
         causal-links:
           0 --at-library--> 2
     GOAL: at-clock
         established by:
           2 --> at-clock

 # 15
 ( plan-for <plan 4> know-time ( (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
              support-link #15 from { 3 , 12 } by goal-regression
 This node is inferred by discharging a link to interest #14

Plan #4 has been constructed
   PLAN-STEPS:
     (2) go-to-clock
         causal-links:
           0 --at-library--> 2
     (3) read-clock
         causal-links:
           2 --at-clock--> 3
         ordering-constraints:
           3 > 2
     GOAL: know-time
         established by:
```

```
        3 --> know-time

 # 17
 ( plan-for <plan 5> (know-beethoven-birthday & know-time) nil nil nil nil)
 Inferred by:
             support-link #17 from { 8 , 15 } by split-conjunctive-goal
 This node is inferred by discharging a link to interest #5

Plan #5 has been constructed
   PLAN-STEPS:
     (1) ask-librarian
          causal-links:
            0 --at-library--> 1
     (2) go-to-clock
          causal-links:
            0 --at-library--> 2
     (3) read-clock
          causal-links:
            2 --at-clock--> 3
          ordering-constraints:
            3 > 2
     GOAL: (know-beethoven-birthday & know-time)
          established by:
            1 --> know-beethoven-birthday
            3 --> know-time

            =========================================
            Justified belief in ( plan-for <plan 5> (know-beethoven-birthday & know-time) nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan (know-beethoven-birthday & know-time) nil nil nil nil)>
            =========================================
```

5. Undermining Causal-Links

Planning separately for the individual conjuncts can produce plans that destructively interfere with each other, in the sense that although the separate subplans can each be expected to achieve their goals in isolation, the merged plan cannot be expected to achieve both goals. This happens when one plan interferes with another is by *undermining one of its causal-links.* Suppose $plan_2$ contains the causal-link *n --g-->* n*. This means that *g* is a subgoal required for node *n** to do its job, and *g* is achieved in $plan_2$ by node *n*. $Plan_1$ undermines this causal-link iff $plan_1$ contains a subplan *p* consisting of nodes from $plan_1$ where the final step of *p* could (consistent with the ordering-constraints) be executed between *n* and *n**, and *p* is a plan for *~g.* More precisely,

> A plan-step *n* of a plan **undermines a causal-link** $n_1 \rightarrow$ *subgoal* n_2 **relative to** the plan iff there is a linearization of the plan in which *n* occurs between n_1 and n_2 and either *~subgoal* or the negation of a conjunct of *subgoal* is an expectable result of the sequence of actions prescribed by the plan-steps **start**,...,*n* in the linearization relative to the set of all true planning-conditionals.

I will also say that a **plan undermines** one of its causal-links if one of its plan-steps does so relative to that plan.

A plan is undermined by a subplan, consisting of a subset of the plan-steps of the plan ordered in a manner consistent with the plan. The subplan undermines the plan by achieving a goal *~g* which is the negation of some subgoal, and doing so between the time *g* is achieved and the time it is used. The subplan must be sound, i.e., it must really achieve *~g*. But more is required. It must achieve *~g* in the context of the larger plan. That is, the larger plan cannot, in turn undermine the subplan. This is captured by adding the rest of the steps of the original plan into the subplan, even though they are not used, and requiring that the resulting plan still achieves *~g*. The result of adding the unused steps to the subplan is an *embellishment* of the original plan, which is defined as follows:

> $plan_0$ is an **embellishment** of *plan* iff (1) the plan-steps of $plan_0$ are the same as the plan-

steps of *plan*, and (2) any ordering imposed by *plan* on plan-steps of $plan_0$ other than **finish** is also imposed by $plan_0$.

The intent of this definition is that **finish** need not occur at the end of $plan_0$. If it does not, then the definition of soundness given above ignores all plan-steps succeeding **finish**. The *penultimate steps* of a plan are those occurring just before **finish** in some linearization of the plan. It can then be proven:

> A plan-step *n* of a *plan* undermines a causal-link $n_1 \rightarrow subgoal \rightarrow n_2$ of *plan* iff there is a presumptively-sound embellishment $plan_0$ of *plan* whose goal is either *~subgoal* or the negation of a conjunct of *subgoal*, and (1) *n* is the single penultimate plan-step of $plan_0$, (2) there is a linearization of *plan* consistent with the ordering imposed by $plan_0$ in which *n* occurs between n_1 and n_2, and (3) $plan_0$ does not undermine any of its own causal-links.

Accordingly, we have the following defeater for SPLIT-CONJUNCTIVE-GOAL:

UNDERMINE-CAUSAL-LINKS

> Given an inference in accordance with SPLIT-CONJUNCTIVE-GOAL to the conclusion that *plan&* will achieve (G_1 & G_2), adopt interest in establishing that *plan&* undermines one of its own causal-links. If it is determined that it does undermine one of its own causal-links, take the inference to the conclusion that *plan&* will achieve (G_1 & G_2) to be defeated.

Let us say that a plan *achieves* its goal *between* two plan-steps iff it achieves its goal and all its penultimate steps are ordered between the two plan-steps. Similarly, a plan achieves its goal *before* a plan-step iff it achieves its goal and all its penultimate steps are ordered before the plan-step.

UNDERMINE-CAUSAL-LINK

> Given an interest in establishing that *plan&* undermines one of its own causal-links, for each causal-link $n_1 \rightarrow subgoal \rightarrow n_2$ of *plan&*, adopt interest in finding an embellishment $plan_0$ of *plan&* that achieves *~g* between n_1 and n_2 consistent with the ordering-constraints of *plan&*, where *g* is either $subgoal_1$ or a conjunct of $subgoal_1$. Given $plan_0$, infer nondefeasibly that *plan&* undermines one of its own causal-links.

This pair of reasoning-schemas is implemented by the following four backwards-reasons:

```
(def-backwards-undercutter UNDERMINE-CAUSAL-LINKS
   :defeatee  split-conjunctive-goal
   :backwards-premises
   "(define links (causal-links plan&))"
   "(plan-undermines-causal-links plan& links)"
   :variables plan& links)
```

DEF-BACKWARDS-UNDERCUTTER is a variant of DEF-BACKWARDS-REASON that computes the conclusions for us.

```
(def-backwards-reason   PLAN-UNDERMINES-FIRST-CAUSAL-LINK
   :conclusions  "(plan-undermines-causal-links plan links)"
   :condition  (car links)
   :backwards-premises
      "(define first-link (car links))"
      "(plan-undermines-causal-link plan R node first-link)"
   :variables  plan node links first-link R)
```

```
(def-backwards-reason   PLAN-UNDERMINES-ANOTHER-CAUSAL-LINK
```

```
    :conclusions  "(plan-undermines-causal-links plan links)"
    :condition  (cdr links)
    :backwards-premises
       "(define rest-of-links (cdr links))"
       "(plan-undermines-causal-links plan rest-of-links)"
    :variables  plan links rest-of-links)

(def-backwards-reason   PLAN-UNDERMINES-CAUSAL-LINK
    :conclusions  "(plan-undermines-causal-link plan+ R node link)"
    :backwards-premises
       "(define -goal (neg (causal-link-goal link)))"
       "(define node1 (if (not (eq *start* (causal-link-root link))) (causal-link-root link)))"
       "(define node2 (causal-link-target link))"
       "(define before (before-nodes plan+))"
       "(define not-between (not-between plan+))"
       "(embellished-plan-for plan plan+ -goal node1 node2 before not-between)"
       "(define node (penultimate-node plan))"
       "(define R (gen-conjunction (mapcar #'causal-link-goal (call-set node plan))))"
    :variables  plan plan+ link -goal node node1 node2 R before not-between)
```

The search for embellishments can be performed using analogues of the inference-schemes used in searching for plans in the first place, with the difference that the analogues use only the plan-steps of *plan&*. They start with *plan&* stripped of its causal-links, and simply add causal-links and ordering-constraints. Initially, the plan-steps are ordered as in *plan*. Each step of goal-regression adds ordering-constraints. When goal-regression terminates on a subgoal that is already true, then an "embedded null-plan" is constructed whose plan-steps consist of all the plan-steps of *plan* ordered according to the new ordering, but with **finish** preceding all of the other plan-steps. By embedded-goal-regression we then build plans for the earlier subgoals by sliding **finish** along the embellishment and adding subgoals:

EMBEDDED-GOAL-REGRESSION

Given an interest in finding an embellishment of *plan+* that achieves *G* before n_2 (and optionally after n_1) consistent with a set of ordering-constraints *order*, adopt interest in finding planning-conditionals (A & C) $\Rightarrow G$ having *G* as their consequent for which there is a plan-step *n* of *plan+* such that (1) the action prescribed by *n* is *A*, and (2) it is consistent with *order* that *n* occur before n_2 (and optionally after n_1). Given such a conditional and plan-step, let *order+* be the result of adding to *order* the constraint that *n* occur before n_2 (and optionally after n_1). Adopt an interest in finding an embellishment of *plan+* that achieves *C* before *n* consistent with *order+*. If an embellishment *subplan* is proposed for achieving *C* before *n* consistent with *order+*, construct an embellishment *plan* by adding a causal-ink to record the achievement of *G* by *n* and adjusting the ordering-constraints accordingly. Infer defeasibly that *plan* is an embellishment of *plan+* that achieves *G* before n_2 (and optionally after n_1) consistent with *order*.

Embedded goal-regression terminates with goals that are already established:

EMBEDDED-NULL-PLAN

Given an interest in finding an embellishment of *plan* that will achieve *goal* before plan-step *n* consistent with *order*, if *goal* is already true, construct $plan_0$ by (1) letting its plan-steps be the plan-steps of *plan*, (2) letting the ordering-constraints of $plan_0$ be *order*, and (3) taking the only causal-link to be **start** $\rightarrow$ *goal* $\rightarrow$ **finish**. From the truth of *goal* infer nondefeasibly that $plan_0$ is an embellishment of *plan* that will achieve *goal* before plan-step *n* consistent with *order*.

SPLIT-EMBEDDED-CONJUNCTIVE-GOAL

Given an interest in finding an embellishment of *plan* that will achieve a conjunctive goal (G_1 & G_2) before plan-step n consistent with *order*, adopt interest in finding embellishments $plan_1$ and $plan_2$ that will achieve G_1 and G_2 respectively before plan-step n consistent with *order*. If such embellishments are proposed, infer nondefeasibly that $plan_1$ + $plan_2$ is an embellishment of *plan* that will achieve a conjunctive goal (G_1 & G_2) before plan-step n consistent with *order*.

These reasoning-schemas are implemented as follows:

```
(def-backwards-reason EMBEDDED-GOAL-REGRESSION
   :conclusions "(embellished-plan-for plan plan+ goal node1 node2 before not-between)"
   :condition (interest-variable plan)
   :backwards-premises
      "(node-result new-node precondition goal)"
      (:condition
       (if node1
           (member new-node
                    (possibly-intermediate-nodes node1 node2 plan+ (plan-steps plan+) before not-between))
           (member new-node (possibly-preceding-nodes node2 plan+ (plan-steps plan+) before))));)
      "(define new-order
         (multiple-value-bind
            (before* not-between*)
            (catch 'merge-plans
              (add-befores (if node1 (list (cons node1 new-node) (cons new-node node2))
                                 (list (cons new-node node2)))
                              before not-between plan+))
            (list before* not-between*)))"
      (:condition (car new-order))
      "(define new-before (mem1 new-order))"
      "(define new-between (mem2 new-order))"
      "(embellished-plan-for subplan plan+ precondition nil new-node new-before new-between)"
      "(define plan
          (extend-embellished-plan new-node goal subplan plan+))"
      (:condition (not (null plan)))
   :defeasible? t
   :variables plan plan+ subplan goal node1 node2 new-node precondition before not-between
              new-order new-before new-between)

(def-backwards-reason PLAN-NODE-RESULT
   :conclusions  "(node-result node R Q)"
   :condition  (interest-variable node)
   :conclusions-function
   (let ((conclusions nil))
    (let ((m (match+ action (plan-node-action node*))))
      (when m (push (cons `(node-result ,node* ,(match-sublis m R) ,Q) nil) conclusions)))
    conclusions)
   :backwards-premises
     "(=> (& R action) Q)"
      "(plan-node node*)"
   :variables  action node node* R Q)

(def-backwards-reason EMBEDDED-NULL-PLAN
   :conclusions
      "(embellished-plan-for plan plan+ goal node1 node2 before not-between)"
   :condition (and (interest-variable plan) (null node1))
   :backwards-premises
      "goal"
```

```
    "(define plan (embedded-null-plan goal plan+ before not-between))"
    (:condition (not (null plan)))
  :variables plan+ goal plan node node1 node2 before not-between)

(def-backwards-reason SPLIT-EMBEDDED-CONJUNCTIVE-GOAL
  :conclusions
    "(embellished-plan-for plan& plan+ (goal1 & goal2) node1 node2 before not=between)"
    "(merged-plan plan& plan1 plan2 goals)"
    (:defeasible? nil)
  :condition (and (interest-variable plan&) (null node1))
  :backwards-premises
    "(embellished-plan-for plan1 plan+ goal1 node1 node2 before not=between)"
    "(define before1 (before-nodes plan1))"
    "(define not-between1 (not-between plan1))"
    "(embellished-plan-for plan2 plan+ goal2 node1 node2 before1 not-between1)"
    "(define plan& (merge-embellished-plans plan1 plan2 goal1 goal2))"
    (:condition (not (null plan&)))
  :variables
   plan+ plan& plan1 plan2 nodes goal1 goal2 node1 node2 before not=between before1 not-between1)
```

In searching for embellished-plans, we will occasionally be so fortunate as to have already constructed such a plan, in which case we can short-circuit the search by using the following reason-schema:

```
(def-backwards-reason EMBELLISHED-PLAN-FOR-GOAL
  :conclusions  "(embellished-plan-for plan plan+ -goal node1 node2 before not-between)"
  :condition  (interest-variable plan)
  :forwards-premises
    "(plan-for plan0 -goal goals nil nil nil)"
    (:condition (and (subplan plan0 plan+) (not (conjunctionp -goal))))
    "(define penultimate-node (penultimate-node plan0))"
    (:condition
     (if node1 (member penultimate-node
                 (possibly-intermediate-nodes node1 node2 plan+ (plan-steps plan+) before not-between))
               (member penultimate-node
                  (possibly-preceding-nodes node2 plan+ (plan-steps plan+) before))))
    "(define new-order
        (let ((before0 (remove-finish before))
              (not-between0 (remove-not-between-finish before not-between)))
          (dolist (L (causal-links plan0))
             (when (eq (causal-link-target L) *finish*) (push (cons (causal-link-root L) *finish*) before0)))
          (dolist (n (possibly-succeeding-nodes penultimate-node plan+ (plan-steps plan+) before0))
             (multiple-value-bind
                 (before-nodes* not-between*)
                 (add-before *finish* n plan+ before0 not-between0)
                 (setf before0 before-nodes* not-between0 not-between*)))
          (list before0 not-between0)))"
    (:condition (not (null new-order)))
    "(define plan
       (build-plan
        (plan-steps plan+) -goal (causal-links plan0) (car new-order) (cadr new-order)))"
  :variables  plan plan0 plan+ -goal node node1 node2 penultimate-node goals before not-between new-order)
```

EMBEDDED-GOAL-REGRESSION, unlike GOAL-REGRESSION, is defeasible. This is because in EMBEDDED-GOAL-REGRESSION, the causal-link achieving *G* may be undermined by other plan-steps in *plan* that can be consistently ordered between the causal-link root and the causal-link target. In GOAL-REGRESSION, on the other hand, there are no plan-steps in the plan so far constructed that can be consistently ordered between the causal-link root and the causal-link target. The defeat of EMBEDDED-GOAL-REGRESSION is accomplished by the following variant of UNDERMINE-CAUSAL-LINKS:

UNDERMINE-EMBEDDED-CAUSAL-LINKS

Given an inference in accordance with EMBEDDED-GOAL-REGRESSION, ADD-EMBEDDED-ORDERING-CONSTRAINT, or EMBEDDED-CONFRONTATION to the conclusion that *plan*+ is an embellishment of *plan* that achieves *G* before n_2 (and optionally after n_1) consistent with *order*, adopt interest in establishing that *plan*+ undermines one of its own causal-links. If it is determined that it does undermine one of its own causal-links, take the inference to the conclusion that *plan*+ is an embellishment of *plan* that achieves *G* before n_2 (and optionally after n_1) consistent with *order* to be defeated.

This is implemented very simply:

```
(def-backwards-undercutter UNDERMINE-EMBEDDED-CAUSAL-LINKS
   :defeatee   embedded-goal-regression
   :backwards-premises
   "(define links (set-difference (causal-links plan) (causal-links subplan)))"
   "(plan-undermines-causal-links plan links)"
   :variables plan subplan links)
```

Note that unlike SPLIT-CONJUNCTIVE-GOAL, SPLIT-EMBEDDED-CONJUNCTIVE-GOAL is not defeasible. This is because $plan_1$ + $plan_2$ has the same plan-steps as both $plan_1$ both $plan_2$, so if a causal-link of either $plan_1$ or $plan_2$ is undermined by $plan_1$ + $plan_2$, it will already be undermined in $plan_1$ or $plan_2$ itself when it is constructed by EMBEDDED-GOAL-REGRESSION.

To illustrate the undermining of causal-links, consider the full planning problem of figure 2, in which going to the clock requires leaving the library:

```
======================================================================
                    OSCAR_3.25       4/15/1998       15:58:56
                         Non-Linear-Planner-31

Problem number 3:

Forwards-substantive-reasons:
        simplify-=>

Backwards-substantive-reasons:
        null-plan
        goal-regression
        split-conjunctive-goal
        plan-node-result
        undermine-causal-links
        plan-undermines-first-causal-link
        plan-undermines-another-causal-link
        plan-undermines-causal-link
        embellished-plan-for-goal
        embedded-null-plan
        split-embedded-conjunctive-goal
        embedded-goal-regression

Goal-state:
        know-beethoven-birthday
        know-time

Inputs:

Given:
        at-library  :  with justification = 0.99
        ( (at-library & ask-librarian) => know-beethoven-birthday)  :  with justification = 0.99
        ( (at-clock & read-clock) => know-time)  :  with justification = 0.99
        ( (at-library & go-to-clock) => (at-clock & ~at-library))  :  with justification = 0.99

===========================================================================
THE FOLLOWING IS THE REASONING INVOLVED IN THE SOLUTION
```

Nodes marked DEFEATED have that status at the end of the reasoning.

```
 # 1
 at-library
 given
 This discharges interest 13
 # 2
 ( (at-library & ask-librarian) => know-beethoven-birthday)
 given
 This discharges interest 10
 # 3
 ( (at-clock & read-clock) => know-time)
 given
 This discharges interest 15
 # 4
 ( (at-library & go-to-clock) => (at-clock & ~at-library))
 given
                                   # 5
                                   interest: ( plan-for ^@y0 (know-beethoven-birthday & know-time) nil nil nil nil)
                                   This is of ultimate interest
                                   # 6
                                   interest: ( plan-for ^@y1 know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil nil)
                                   For interest 5 by split-conjunctive-goal
                                   This interest is discharged by node 10
 # 5
 ( (at-library & go-to-clock) => ~at-library)
 Inferred by:
             support-link #5 from { 4 } by simplify-=>
 This discharges interest 28
 # 6
 ( (at-library & go-to-clock) => at-clock)
 Inferred by:
             support-link #6 from { 4 } by simplify-=>
 This discharges interest 18
                                   # 10
                                   interest: ( (^@y4 & ^@y5) => know-beethoven-birthday)
                                   For interest 6 by goal-regression
                                   This interest is discharged by node 2
                                   # 11
                                   interest: ( plan-for ^@y6 at-library ( know-beethoven-birthday (know-beethoven-birthday & know-time))
                                         nil nil nil)
                                   For interest 6 by goal-regression using node 2
                                   This interest is discharged by node 7
                                   # 13
                                   interest: at-library
                                   For interest 11 by null-plan
                                   For interest 19 by null-plan
                                   For interest 30 by embedded-null-plan
                                   This interest is discharged by node 1
 # 7
 ( plan-for <plan 1> at-library ( know-beethoven-birthday (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
             support-link #7 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #11

Plan #1 has been constructed
    GOAL: at-library
        established by:
          0 --> at-library

 # 10
 ( plan-for <plan 2> know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
             support-link #10 from { 2 , 7 } by goal-regression
 This node is inferred by discharging a link to interest #6

Plan #2 has been constructed
   PLAN-STEPS:
    (1) ask-librarian
        causal-links:
          0 --at-library--> 1
    GOAL: know-beethoven-birthday
        established by:
          1 --> know-beethoven-birthday

                                   # 14
                                   interest: ( plan-for ^@y9 know-time ( (know-beethoven-birthday & know-time)) nil nil nil)
                                   For interest 5 by split-conjunctive-goal using node 10
                                   This interest is discharged by node 17
```

```
                                  # 15
                                  interest: ( (^@y10 & ^@y11) => know-time)
                                  For interest 14 by goal-regression
                                  This interest is discharged by node 3
                                  # 16
                                  interest: ( plan-for ^@y12 at-clock ( know-time (know-beethoven-birthday & know-time)) nil nil nil)
                                  For interest 14 by goal-regression using node 3
                                  This interest is discharged by node 14
                                  # 18
                                  interest: ( (^@y13 & ^@y14) => at-clock)
                                  For interest 16 by goal-regression
                                  This interest is discharged by node 6
                                  # 19
                                  interest: ( plan-for ^@y15 at-library ( at-clock know-time (know-beethoven-birthday & know-time)) nil nil
                                      nil)
                                  For interest 16 by goal-regression using node 6
                                  This interest is discharged by node 11
 # 11
 ( plan-for <plan 1> at-library ( at-clock know-time (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
         support-link #11 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #19

Plan #1 has been constructed
    GOAL: at-library
       established by:
        0 --> at-library

 # 12
 ( plan-node <pn2: go-to-clock>)
 Inferred by:
         support-link #12 from {  } by given
 This discharges interest 29
 # 14
 ( plan-for <plan 3> at-clock ( know-time (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
         support-link #14 from { 6 , 11 } by goal-regression
 This node is inferred by discharging a link to interest #16

Plan #3 has been constructed
   PLAN-STEPS:
    (2) go-to-clock
       causal-links:
        0 --at-library--> 2
    GOAL: at-clock
       established by:
        2 --> at-clock

 # 17
 ( plan-for <plan 4> know-time ( (know-beethoven-birthday & know-time)) nil nil nil)
 Inferred by:
         support-link #17 from { 3 , 14 } by goal-regression
 This node is inferred by discharging a link to interest #14

Plan #4 has been constructed
   PLAN-STEPS:
    (2) go-to-clock
       causal-links:
        0 --at-library--> 2
    (3) read-clock
       causal-links:
        2 --at-clock--> 3
       ordering-constraints:
        3 > 2
    GOAL: know-time
       established by:
        3 --> know-time

 # 19
 ( plan-for <plan 5> (know-beethoven-birthday & know-time) nil nil nil nil)              DEFEATED
 Inferred by:
         support-link #19 from { 10 , 17 } by split-conjunctive-goal  defeaters: { 26 }   DEFEATED
 This node is inferred by discharging a link to interest #5

Plan #5 has been constructed
   PLAN-STEPS:
    (1) ask-librarian
       causal-links:
        0 --at-library--> 1
```

```
      (2) go-to-clock
          causal-links:
            0 --at-library--> 2
      (3) read-clock
          causal-links:
            2 --at-clock--> 3
          ordering-constraints:
            3 > 2
      GOAL: (know-beethoven-birthday & know-time)
          established by:
            1 --> know-beethoven-birthday
            3 --> know-time

                          # 21
                          interest: ((( plan-for <plan 2> know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil
           nil) & ( plan-for <plan 4> know-time ( (know-beethoven-birthday & know-time)) nil nil nil)) @ ( plan-for <plan 5>
           (know-beethoven-birthday & know-time) nil nil nil nil))
                          Of interest as a defeater for support-link 19 for node 19
                          This interest is discharged by node 26
                          # 22
                          interest: ( plan-undermines-causal-links <plan 5> ( <0 --at-library--> 1> <1 --know-beethoven-birthday-->
           -1> <3 --know-time--> -1> <2 --at-clock--> 3> <0 --at-library--> 2>))
                          For interest 21 by undermine-causal-links
                          This interest is discharged by node 25
            ==========================================
            Justified belief in ( plan-for <plan 5> (know-beethoven-birthday & know-time) nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan (know-beethoven-birthday & know-time) nil nil nil nil)>
            ==========================================

Plan #5 has been adopted
    PLAN-STEPS:
      (1) ask-librarian
          causal-links:
            0 --at-library--> 1
      (2) go-to-clock
          causal-links:
            0 --at-library--> 2
      (3) read-clock
          causal-links:
            2 --at-clock--> 3
          ordering-constraints:
            3 > 2
      GOAL: (know-beethoven-birthday & know-time)
          established by:
            1 --> know-beethoven-birthday
            3 --> know-time

                          # 25
                          interest: ( plan-undermines-causal-link <plan 5> ^@y19 ^@y18 <0 --at-library--> 1>)
                          For interest 22 by plan-undermines-first-causal-link
                          This interest is discharged by node 24
                          # 26
                          interest: ( embellished-plan-for ^@y20 <plan 5> ~at-library nil <pn1: ask-librarian> ( (<pn3: read-clock> .
                                *finish*) (<pn2: go-to-clock> . <pn3: read-clock>) (<pn1: ask-librarian> . *finish*)) nil)
                          For interest 25 by plan-undermines-causal-link
                          This interest is discharged by node 23
                          # 27
                          interest: ( node-result ^@y21 ^@y22 ~at-library)
                          For interest 26 by embedded-goal-regression
                          This interest is discharged by node 20
                          # 28
                          interest: ( (^@y22 & ^@y23) => ~at-library)
                          For interest 27 by plan-node-result
                          This interest is discharged by node 5
                          # 29
                          interest: ( plan-node ^@y24)
                          For interest 27 by plan-node-result using node 5
                          This interest is discharged by node 12
 # 20
 ( node-result <pn2: go-to-clock> at-library ~at-library)
 Inferred by:
             support-link #20 from { 5 , 12 } by plan-node-result
 This node is inferred by discharging a link to interest #27
                          # 30
                          interest: ( embellished-plan-for ^@y25 <plan 5> at-library nil <pn2: go-to-clock> ( (<pn2: go-to-clock> .
           <pn1: ask-librarian>) (<pn3: read-clock> . *finish*) (<pn2: go-to-clock> . <pn3: read-clock>) (<pn1: ask-librarian> .
           *finish*)) nil)
                          For interest 26 by embedded-goal-regression using node 20
```

```
                    This interest is discharged by node 21
 # 21
 ( embellished-plan-for <plan 6> <plan 5> at-library nil <pn2: go-to-clock> ( (<pn2: go-to-clock> . <pn1: ask-librarian>) (<pn3:
       read-clock> . *finish*) (<pn2: go-to-clock> . <pn3: read-clock>) (<pn1: ask-librarian> . *finish*)) nil)
 Inferred by:
        support-link #21 from { 1 } by embedded-null-plan
 This node is inferred by discharging a link to interest #30

Plan #6 has been constructed
  PLAN-STEPS:
    GOAL: at-library
       established by:
        0 --> at-library
    (2) go-to-clock
       ordering-constraints:
        2 > *finish*
    (1) ask-librarian
       ordering-constraints:
        1 > 2
    (3) read-clock
       ordering-constraints:
        3 > 2

 # 23
 ( embellished-plan-for <plan 7> <plan 5> ~at-library nil <pn1: ask-librarian> ( (<pn3: read-clock> . *finish*) (<pn2: go-to-clock> .
       <pn3: read-clock>) (<pn1: ask-librarian> . *finish*)) nil)
 Inferred by:
        support-link #23 from { 20 , 21 } by embedded-goal-regression
 This node is inferred by discharging a link to interest #26

Plan #7 has been constructed
  PLAN-STEPS:
    (2) go-to-clock
       causal-links:
        0 --at-library--> 2
    GOAL: ~at-library
       established by:
        2 --> ~at-library
    (1) ask-librarian
       ordering-constraints:
        1 > *finish*
    (3) read-clock
       ordering-constraints:
        3 > *finish*

 # 24
 ( plan-undermines-causal-link <plan 5> at-library <pn2: go-to-clock> <0 --at-library--> 1>)
 Inferred by:
        support-link #24 from { 23 } by plan-undermines-causal-link
 This node is inferred by discharging a link to interest #25
 # 25
 ( plan-undermines-causal-links <plan 5> ( <0 --at-library--> 1> <1 --know-beethoven-birthday--> -1> <3 --know-time--> -1> <2
       --at-clock--> 3> <0 --at-library--> 2>))
 Inferred by:
        support-link #25 from { 24 } by plan-undermines-first-causal-link
 This node is inferred by discharging a link to interest #22
 # 26
 ((( plan-for <plan 2> know-beethoven-birthday ( (know-beethoven-birthday & know-time)) nil nil nil) & ( plan-for <plan 4>
       know-time ( (know-beethoven-birthday & know-time)) nil nil nil)) @ ( plan-for <plan 5> (know-beethoven-birthday &
       know-time) nil nil nil nil))
 Inferred by:
        support-link #26 from { 25 } by undermine-causal-links
 defeatees: { link 19 for node 19 }
 This node is inferred by discharging a link to interest #21
        =========================================
        Lowering the undefeated-degree-of-support of ( plan-for <plan 5> (know-beethoven-birthday & know-time) nil nil nil nil)
        retracts the previous answer to #<Query 1: (? plan)( plan-for plan (know-beethoven-birthday & know-time) nil nil nil nil)>
        =========================================

Plan #5 has been retracted:
  PLAN-STEPS:
    (1) ask-librarian
       causal-links:
        0 --at-library--> 1
    (2) go-to-clock
       causal-links:
        0 --at-library--> 2
    (3) read-clock
       causal-links:
```

```
      2 --at-clock--> 3
    ordering-constraints:
      3 > 2
GOAL: (know-beethoven-birthday & know-time)
    established by:
      1 --> know-beethoven-birthday
      3 --> know-time
============================================================
```

As before, the planner finds plan #5, but it undermines its own causal-link `0 --at-library--> 1`, so the conclusion that plan #5 will achieve its goal is defeated.

6. Repairing Undermining by Imposing Ordering-Constraints

Planning for conjunctive goals by combining plans for the individual conjuncts is defeasible. The reasoning is defeated by discovering undermining. Undermining arises when *plan*& is constructed by SPLIT-CONJUNCTIVE-GOAL and *plan*& undermines one of its own causal-links (where those causal-links are inherited from the plans $plan_1$ and $plan_2$ for the individual conjuncts). Sometimes *plan*& can be repaired. Two ways of repairing undermining can be found in the literature. First, *plan*& is a nonlinear plan, and imposes no ordering of the plan-steps of $plan_1$ and $plan_2$ with respect to each other. Undermining arises when one of these plans contains a causal-link $node_1$ --*goal*--> $node_2$, and the other plan contains a plan-step *node* that can, consistent with the ordering-constraints, be executed between $node_1$ and $node_2$ and has *~goal* as a node-result. One way of avoiding the the undermining is to add an ordering-constraint to *plan*& requiring that *node* not be executed between $node_1$ and $node_2$:[9]

ADD-ORDERING-CONSTRAINT

Given an interest in finding a plan for achieving a conjunctive goal (g_1 & g_2), and plans $plan_1$ for g_1 and $plan_2$ for g_2, if *plan&* is a putative plan for (g_1 & g_2) constructed by merging plans $plan_1$ and $plan_2$ (and possibly other plans), but a plan-step n of *plan&* undermines one of its own causal-links $n_1 \rightarrow subgoal \rightarrow n_2$, construct a plan *plan*+ by adding the ordering-constraint that n not occur between n_1 and n_2 (if this can be done consistently) and infer defeasibly that *plan*+ will achieve (g_1 & g_2).

This is implemented as follows:

```
(def-backwards-reason ADD-ORDERING-CONSTRAINTS
  :conclusions
  "(plan-for plan& (goal1 & goal2) goals nodes nodes-used links)"
  "(merged-plan plan& plan1 plan2 goals)"
    (:defeasible? nil)
  :condition  (interest-variable plan&)
  :forwards-premises
     "(plan-for plan (goal1 & goal2) goals nodes nodes-used links)"
     (:clue? t)
     "(plan-undermines-causal-link plan R node link)"
     (:clue? t)
     "(merged-plan plan plan1 plan2 goals)"
     "(define plan& (add-not-between node link plan))"
     (:condition (not (null plan&)))
```

[9] This strategy stems from ideas due to Sussman [1975], Waldinger [1977], and Warren [1974]. The first clean formulation for nonlinear planners was provided by McAllister and Rosenblitt [1991] and used in SNLP.

```
:defeasible? t
:variables  plan1 plan2 plan& plan node nodes nodes1 nodes2 link goal1 goal2 goals
  goals1 goals2 R nodes-used links links1 links2)
```

(ADD-NOT-BETWEEN *node link plan*) produces a plan just like *plan* but with the additional ordering-constraint that *node* not occur between the causal-link-root and causal-link-target of *link*. Depending upon how the nodes are already ordered in *plan*, this can either insert ordering-constraints into *before-nodes* or into *not-between*. Discovering the undermining *occasions* the inference to "(plan-for *plan& goal goals*)" without being a premise of the inference. This is captured in OSCAR by making two premises of the reason a *clue*. When a premise is a clue, it must be instantiated by an inference-node before a new conclusion is drawn, but the basis from which the new conclusion is inferred does not include the inference-node instantiating the clue. Clues provide a kind of control structure for the reasoning. Note also that this is a degenerate backwards-reason—it has no backwards-premises.

When (ADD-NOT-BETWEEN *node link plan plan1 plan2*) is run to produce plan&, OSCAR draws the conclusion (*add-ordering-constraint plan& plan1 plan2*). If it is impossible to consistently add the ordering-constraint that *node* not occur between the causal-link-root and causal-link-target of *link*, then NIL is returned and OSCAR draws the conclusion "(*add-ordering-constraint NIL plan1 plan2*)".

With the addition of ADD-ORDERING-CONSTRAINTS, the preceding problem is now done correctly by adding the following inference to the previous reasoning:

```
# 29
  ( plan-for <plan 8> (know-beethoven-birthday & know-time) nil nil nil nil)
  Inferred by:
            support-link #29 from { 18 } by add-ordering-constraints with clues { 24 , 19 }
  This discharges interest 5

Plan #8 has been constructed
    PLAN-STEPS:
      (1) ask-librarian
          causal-links:
            0 --at-library--> 1
      (2) go-to-clock
          causal-links:
            0 --at-library--> 2
          ordering-constraints:
            2 > 1
      (3) read-clock
          causal-links:
            2 --at-clock--> 3
          ordering-constraints:
            3 > 2
      GOAL: (know-beethoven-birthday & know-time)
          established by:
            1 --> know-beethoven-birthday
            3 --> know-time

            =========================================
            Justified belief in ( plan-for <plan 8> (know-beethoven-birthday & know-time) nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan (know-beethoven-birthday & know-time) nil nil nil nil)>
            =========================================
```

ADD-ORDERING-CONSTRAINTS is defeasible, but the defeaters should be the same as for SPLIT-CONJUNCTIVE-GOAL. Thus we can simply add ADD-ORDERING-CONSTRAINTS to the list of defeatees of UNDERMINE-CAUSAL-LINKS.

We can also try to repair undermined embellishments by adding ordering-constraints:

ADD-EMBEDDED-ORDERING-CONSTRAINT

> Given an interest in finding an embellishment of *plan* that achieves G before n_2 (and optionally after n_1) consistent with a set of ordering-constraints *order*, if *plan+* is a putative such embellishment but a plan-step n of *plan+* undermines one of its own causal-links n_1

$\rightarrow$ *subgoal*$_1$ $\rightarrow$ ... $\rightarrow$ *subgoal*$_n$ $\rightarrow$ n_2 $\rightarrow$ *goal*$_1$ $\rightarrow$ ... $\rightarrow$ *goal*$_m$, construct a plan *plan*++ by adding the ordering-constraint that n not occur between n_1 and n_2 (if this can be done consistently) and infer defeasibly that *plan*++ is an embellishment of *plan* that achieves G before n_2 (and optionally after n_1) consistent with a set of ordering-constraints *order*.

```
(def-backwards-reason ADD-EMBEDDED-ORDERING-CONSTRAINTS
  :conclusions "(embellished-plan-for plan& plan+ goal node1 node2 before not=between)"
  :condition (interest-variable plan&)
  :forwards-premises
    "(node-result new-node precondition goal)"
    "(embellished-plan-for subplan plan+ precondition nil new-node new-before new-not-between)"
    "(extended-plan plan subplan new-node)"
    (:clue? t)
    "(plan-undermines-causal-link plan R node link)"
    (:condition (member link (causal-links plan)))  ;; is this needed?
    (:clue? t)
    "(define constraints
       (multiple-value-bind
          (before* not-between*)
          (add-<> node (causal-link-root link) (causal-link-target link) plan
                        (before-nodes plan) (not-between plan))
          (list before* not-between*)))"
    (:condition (not (equal constraints (list nil nil))))
    "(define before+ (car constraints))"
    "(define not-between+ (cadr constraints))"
    "(define plan&
        (build-plan (plan-steps plan) (plan-goal plan) (causal-links plan) (car constraints) (cadr constraints)))"
    (:condition (not (null plan&)))
  :defeasible? t
  :variables  plan plan+ plan& goal node1 node2 before not=between new-before new-not-between
              new-node precondition subplan R node link constraints)
```

This becomes another defeatee for UNDERMINE-CAUSAL-LINKS.

7. The Goal-Stack

As OSCAR acquires new subgoals by goal-regression, a record of the previous goals is kept on the goal-stack. This is to avoid pointless cycling in the search for a plan. Suppose OSCAR is given

$(C \text{ \& } action_1) \Rightarrow goal$
$(goal \text{ \& } action_2) \Rightarrow C$

and the task is to find a plan for *goal*. Without the goal-stack restriction on GOAL-REGRESSION, OSCAR would adopt the subgoal C, and then to get C OSCAR would adopt the new subgoal *goal*. But the latter is pointless. It is instructing OSCAR to look for a plan for *goal* in order to get C in order to get *goal*. This leads to an infinite loop. This is avoided by the restriction on GOAL-REGRESSION requiring that the subgoal not be a member of *goals*.[10]

A similar phenomenon occurs in connection with SPLIT-CONJUNCTIVE-GOAL. The goal is added to the goal-stack on the assumption that it will ordinarily be pointless to renew the search

---

[10] As far as I can tell, most planners simply tolerate this problem, relying upon some variant of breadth-first search to avoid an infinite regress.

for a plan for a subgoal when the subgoal is already on the goal-stack. However, in the context of searching for plans for conjunctive goals, this is not always the case. For example, consider the following planning problem:

```
Goal-state:
        (at-store & have-money)

Inputs:

Given:
        ~have-money  :  with justification = 0.99
        ~have-food  :  with justification = 0.99
        ~at-store  :  with justification = 0.99
        ((~have-money & beg) ⇒ have-money)  :  with justification = 0.99
        (((at-store & have-money) & buy-food) ⇒ (have-food & ~have-money))  :  with justification = 0.99
        (((~at-store & have-money) & take-bus) ⇒ (at-store & ~have-money))  :  with justification = 0.99
```

OSCAR finds and retracts the following incorrect plan:

```
Plan #7:
   PLAN-STEPS:
     (1) beg
        causal-links:
          0 --~have-money--> 1
     (2) take-bus
        causal-links:
          1 --have-money--> 2
          0 --~at-store--> 2
        ordering-constraints:
          2 > 1
     (3) buy-food
        causal-links:
          2 --at-store--> 3
          1 --have-money--> 3
        ordering-constraints:
          3 > 2
     GOAL: have-food
        established by:
          3 --> have-food
```

However, OSCAR does not find the following correct plan:

```
Plan #14:
   PLAN-STEPS:
     (1) beg
        causal-links:
          0 --~have-money--> 1
     (2) take-bus
        causal-links:
          0 --~at-store--> 2
          1 --have-money--> 2
        ordering-constraints:
          2 > 1
     (4) beg
        causal-links:
          2 --~have-money--> 4
        ordering-constraints:
          4 > 2
     (3) buy-food
        causal-links:
          2 --at-store--> 3
          4 --have-money--> 3
```

```
        ordering-constraints:
          3 > 4
    GOAL: have-food
        established by:
          3 --> have-food
```

The difficulty is that to find plan #14, OSCAR must reason by goal-regression from plan #13:

```
Plan #13:
  PLAN-STEPS:
    (1) beg
        causal-links:
          0 --~have-money--> 1
    (2) take-bus
        causal-links:
          0 --~at-store--> 2
          1 --have-money--> 2
        ordering-constraints:
          2 > 1
    (4) beg
        causal-links:
          2 --~have-money--> 4
        ordering-constraints:
          4 > 2
    GOAL: (at-store & have-money)
        established by:
          2 --> at-store
          4 --> have-money
```

and to find plan#13, OSCAR must find the following two plans and merge them:

```
Plan #12:
  PLAN-STEPS:
    (1) beg
        causal-links:
          0 --~have-money--> 1
    (2) take-bus
        causal-links:
          0 --~at-store--> 2
          1 --have-money--> 2
        ordering-constraints:
          2 > 1
    (4) beg
        causal-links:
          2 --~have-money--> 4
        ordering-constraints:
          4 > 2
    GOAL: have-money
        established by:
          4 --> have-money

 Plan #5:
  PLAN-STEPS:
    (1) beg
        causal-links:
          0 --~have-money--> 1
    (2) take-bus
        causal-links:
          0 --~at-store--> 2
          1 --have-money--> 2
        ordering-constraints:
          2 > 1
    GOAL: at-store
```

```
      established by:
        2 --> at-store
```

Finding plan #5 is unproblematic, but OSCAR cannot find plan #12 using the reason-schemas thus far discussed. Presented with the goal of having money, OSCAR finds only the simpler plan:

```
Plan #3:
  PLAN-STEPS:
    (1) beg
      causal-links:
        0 --~have-money--> 1
    GOAL: have-money
      established by:
        1 --> have-money
```

The difficulty in finding plan #12 arises from the constraint on goal-regression that *precondition* not be a member of *goals*. To construct plan #12, OSCAR would have to reason backwards as follows:

```
                    # 9
                    interest: (plan-for ^@y3 (at-store & have-money) ((at-store & have-money) have-food) nil nil nil)
                    For interest 7 by goal-regression using node 8

 # 11  (plan-for <plan 1> ~at-store (~at-store (~at-store & have-money) at-store (at-store & have-money) have-food) nil nil nil)
 Inferred by support-link #11 from { 3 } by null-plan

                    # 20
                    interest: (plan-for ^@y15 have-money (have-money (~at-store & have-money) at-store (at-store &
                        have-money) have-food) nil nil nil)
                    For interest 14 by split-conjunctive-goal using node 11

 # 4   ((~have-money & beg) ⟹ have-money)
 given

                    # 22
                    interest: ( plan-for ^@y18 ~have-money ( have-money (~at-store & have-money) at-store (at-store &
                        have-money) have-food) nil nil nil)
                    For interest 20 by goal-regression using node 4

 # 10   (((~at-store & have-money) & take-bus) ⟹ ~have-money)
 Inferred by support-link #10 from { 6 } by simplify-⟹

                    # 23
                    interest: (plan-for ^@y7 (~at-store & have-money) (~have-money have-money (~at-store & have-
                        money) at-store (at-store & have-money) have-food) nil nil nil)
                    For interest 16 by goal-regression using node 8
```

The difficulty is that the last step, whereby interest 23 is adopted, violates the constraint on the goal-stack, because (~at-store & have-money) is already present on the goal-stack.

Looking more closely at plan #12, we see that it is not a plan we would ordinarily want OSCAR to produce given the goal of having money. It only becomes desirable for OSCAR to produce this plan in order to combine it with plan #5 to produce plan #13. This in turn is only desirable because when OSCAR merges plan #3 and plan #5 to produce:

```
Plan #6:
  PLAN-STEPS:
    (1) beg
      causal-links:
        0 --~have-money--> 1
    (2) take-bus
      causal-links:
        1 --have-money--> 2
        0 --~at-store--> 2
      ordering-constraints:
        2 > 1
    GOAL: (at-store & have-money)
```

```
established by:
  2 --> at-store
  1 --> have-money
```

the resulting plan undermines the causal-link 1 ➞ have-money ➞ *finish* that it inherits from plan #3. What is happening here is that, in accordance with plan #3, plan-node 1 achieves one of the goals of the plan, but when it is merged with plan #5, the goal achieved by plan-node 1 is "consumed" by plan-node 2.

To find plan #12, OSCAR must be able to renew the search for the subgoal (~at-store & have-money) when it is already on the goal-stack. If that were the ultimate goal, renewing the search would be pointless. But it can acquire a point when the plan is combined with plan #5, because plan #5 contains steps that consume the effects of plan-node 1 before it can be used in a plan for the conjunctive goal "have-money & at-store". The "consuming" consists of plan #5 undermining a causal-link of plan #3 in a way that cannot be fixed by adding ordering-constraints (because plan #5 requires plan-node 1 to precede plan-node 2). In this case we want to override the subgoal constraint.

When searching for separate plans for the conjuncts of a conjunctive goal, the goal-stack constraint can become inappropriate because, in effect, the execution of the initial steps of the plan for one conjunct can alter the start-state for the plan for the other conjunct. In the current example, executing steps (1) and (2) of plan #5 puts the agent in a state in which plan #3 is no longer a correct plan for the goal "have-money". In fact, plan #3 cannot even be executed at that point because it would require performing plan-step (2) *after* plan-step (2) has already been performed. To find plan #12, OSCAR must reopen the search for a plan for "have-money" with a new start-state consisting of the state that results from executing the first two steps (in fact all of) plan #5, and then attach the resulting plan to the steps ((1) and (3)) leading to that start-state, and merge the result with plan #5. This is accomplished by REUSE-NODES-1 and REUSE-NODES-2 (the first covers the case in which plan1 is replaced by a plan that reuses some of the nodes of plan2, and the second is the converse):

```
(def-backwards-reason REUSE-NODES-1
   :conclusions
   (plan-for plan& goal goals nodes nodes-used links)
   "(merged-plan plan& plan1 plan2 goals)"
   (:defeasible? nil)
   :condition (interest-variable plan&)
   :forwards-premises
   "(plan-undermines-causal-link plan+ R node link)"
   (:condition (and (not (member node nodes))
                         (not (member node nodes-used))
                         (equal (plan-goal plan+) goal)))
   (:clue? t)
   "(adds-ordering-constraints nil plan+ node link)"
   (:clue? t)
   "(merged-plan plan+ plan1- plan2 goals)"
   (:condition
    (and
     (or (equal (causal-link-root link) *start*)
         (and (member (causal-link-root link) (plan-steps plan1-))
              (member (causal-link-root link) (plan-steps plan2))))
     (member link (causal-links plan1-))))
   (:clue? t)
   "(define new-goals (cons goal goals))"
   "(define goal2 (conjunct2 goal))"
   "(plan-for plan2 goal2 new-goals nodes nodes-used links)"
   "(define goal0 (causal-link-goal link))"
   "(plan-for plan0 goal0 goals0 nodes nodes-used links)"
   (:condition (and (subplan plan0 plan1-)
```

```
                         (some #'(lambda (L)
                                        (and (eq (causal-link-target L) *finish*)
                                                 (eq (causal-link-root L) (causal-link-root link))
                                                 (equal (causal-link-goal L) goal0)))
                                    (causal-links plan0))
                         (goal-stack-extension plan+ (cons goal0 goals0) goals link)))
   (:clue? t)
   "(define new-nodes
         (union (cons node (possibly-preceding-nodes node plan2 (plan-steps plan2) (before-nodes plan2)))
                   nodes))"
   :backwards-premises
   "(define goal1 (conjunct1 goal))"
   "(define new-links (if (eq (causal-link-target link) *finish*) links (cons link links)))"
   "(plan-for plan1 goal1 new-goals new-nodes nodes-used new-links)"
   (:condition (and (some #'(lambda (n) (member n new-nodes)) (plan-steps plan2))
                         (not (some #'(lambda (L) (member L (causal-links plan1))) new-links))))
   "(define plan& (merge-plans plan1 plan2 goal1 goal2))"
   (:condition (not (null plan&)))
   :defeasible? t
   :variables
   plan& goal goal1 goal2 goal0 goals new-goals nodes plan0 plan+ R node link
   plan1 plan1- plan2 goals0 new-nodes nodes-used links new-links)

(def-backwards-reason REUSE-NODES-2
   :conclusions
   (plan-for plan& goal goals nodes nodes-used links)
   "(merged-plan plan& plan1 plan2 goals)"
   (:defeasible? nil)
   :condition (interest-variable plan&)
   :forwards-premises
   "(plan-undermines-causal-link plan+ R node link)"
   (:condition (and (not (member node nodes))
                         (not (member node nodes-used))
                         (equal (plan-goal plan+) goal)))
   (:clue? t)
   "(adds-ordering-constraints nil plan+ node link)"
   (:clue? t)
   "(merged-plan plan+ plan1 plan2- goals)"
   (:condition
     (and
       (or (equal (causal-link-root link) *start*)
            (and (member (causal-link-root link) (plan-steps plan1))
                    (member (causal-link-root link) (plan-steps plan2-))))
       (member link (causal-links plan2-))))
   (:clue? t)
   "(define new-goals (cons goal goals))"
   "(define goal1 (conjunct1 goal))"
   "(plan-for plan1 goal1 new-goals nodes nodes-used links)"
   "(define goal0 (causal-link-goal link))"
   "(plan-for plan0 goal0 goals0 nodes nodes-used links)"
   (:condition (and (subplan plan0 plan2-)
                         (some #'(lambda (L)
                                        (and (eq (causal-link-target L) *finish*)
                                                 (eq (causal-link-root L) (causal-link-root link))
                                                 (equal (causal-link-goal L) goal0)))
                                    (causal-links plan0))
                         (goal-stack-extension plan+ (cons goal0 goals0) goals link)))
   (:clue? t)
   "(define new-nodes
```

```
        (union (cons node (possibly-preceding-nodes node plan1 (plan-steps plan1) (before-nodes plan1)))
                nodes))"
   :backwards-premises
   "(define goal2 (conjunct2 goal))"
   "(define new-links (if (eq (causal-link-target link) *finish*) links (cons link links)))"
   "(plan-for plan2 goal2 new-goals new-nodes nodes-used new-links)"
   (:condition (and (some #'(lambda (n) (member n new-nodes)) (plan-steps plan2))
                    (not (some #'(lambda (L) (member L (causal-links plan2))) new-links))))
   "(define plan& (merge-plans plan1 plan2 goal1 goal2))"
   (:condition (not (null plan&)))
   :defeasible? t
   :variables
   plan& goal goal1 goal2 goal0 goals new-goals nodes plan0 plan+ R node link
   plan1 plan2 plan2- goals0 new-nodes nodes-used links new-links)
```

To explain, suppose we merge *plan1*- and *plan2* to form *plan+*, and then find that a plan-step *node* in *plan+* undermines the causal-link *link* of *plan1-*, and the undermining cannot be avoided by adding ordering-constraints. Then we reopen the search for a plan for *goal1*, this time allowing the plan for *goal1* to reuse *node* and the possibly-preceding-nodes of *node* in *plan2*. Letting these comprise the list *new-nodes*, we accomplish this by adopting interest in "(plan-for *plan1 goal1 goals new-nodes* nil {*link*})". Recalling that GOAL-REGRESSION is formulated as follows:

```
(def-backwards-reason GOAL-REGRESSION
   :conclusions "(plan-for plan goal goals nodes nodes-used links)"
   :condition (and (interest-variable plan) (not (mem goal goals)) (null nodes-used))
   :backwards-premises
      "((precondition & action) => goal)"
      (:condition (not (mem precondition goals)))
      "(define new-goals (cons goal goals))"
      "(plan-for subplan precondition new-goals nodes nodes-used links)"
      "(define plan (extend-plan action goal subplan links))"
      (:condition (not (null plan)))
   :variables precondition action goal plan subplan goals new-goals nodes nodes-used links)
```

we supplement it with

```
(def-backwards-reason REUSE-NODE
   :conclusions "(plan-for plan goal goals nodes nodes-used links)"
   :condition (and (interest-variable plan) (not (null nodes)))
   :backwards-premises
      "(node-result node precondition goal)"
      (:condition (member node nodes))
      "(define new-nodes (remove node nodes))"
      "(define new-nodes-used (cons node nodes-used))"
      "(plan-for subplan precondition goals new-nodes new-nodes-used links)"
      "(define plan (extend-plan-with-node node goal subplan links))"
      (:condition (not (null plan)))
   :variables precondition plan goal goals nodes node new-nodes subplan nodes-used new-nodes-used links)
```

EXTEND-PLAN-WITH-NODE is like EXTEND-PLAN, except that it extends the plan with the given node instead of building a new node. The variable *links* designates a set of undermined links, and so both EXTEND-PLAN-WITH-NODE and EXTEND-PLAN preclude reusing those links. Once we have begun reusing nodes, the nodes used are stored in the variable *nodes-used*, Once *nodes-used* becomes non-empty, we are precluded from further use of GOAL-REGRESSION and must use REUSE-NODE instead. This corresponds to the fact that we are just using the nodes in *nodes* to construct a new start-state.

RESUSE-NODES-1 and REUSE-NODES-2 are added to the list of defeatees for UNDERMINE-

CAUSAL-LINKS. With these additions, OSCAR readily finds plan #14 in the above problem:

```
(
======================================================================

                OSCAR_3.25     4/15/1998     16:32:27
                     Non-Linear-Planner-31

Problem number 8:

Forwards-substantive-reasons:
     simplify-=>

Backwards-substantive-reasons:
     null-plan
     goal-regression
     split-conjunctive-goal
     reuse-nodes-1
     reuse-nodes-2
     reuse-node
     plan-node-result
     =>-adjunction
     =>-neg1
     =>-neg2
     undermine-causal-links
     undermine-embedded-causal-links
     plan-undermines-first-causal-link
     plan-undermines-another-causal-link
     plan-undermines-causal-link
     embellished-plan-for-goal
     embedded-null-plan
     split-embedded-conjunctive-goal
     embedded-goal-regression
     add-ordering-constraints
     add-embedded-ordering-constraints

Goal-state:
     have-food

Inputs:

Given:
     ~have-money  :  with justification = 0.99
     ~have-food  :  with justification = 0.99
     ~at-store  :  with justification = 0.99
     ( (~have-money & beg) => have-money)  :  with justification = 0.99
     ( ((at-store & have-money) & buy-food) => (have-food & ~have-money))  :  with justification = 0.99
     ( ((~at-store & have-money) & take-bus) => (at-store & ~have-money))  :  with justification = 0.99

=========================================================================
THE FOLLOWING IS THE REASONING INVOLVED IN THE SOLUTION
Nodes marked DEFEATED have that status at the end of the reasoning.

 # 1
 ~have-money
 given
 This discharges interest 24
 # 3
 ~at-store
 given
 This discharges interest 18
 # 4
 ( (~have-money & beg) => have-money)
 given
 This discharges interests (41 74 21)
 # 5
 ( ((at-store & have-money) & buy-food) => (have-food & ~have-money))
 given
 # 6
 ( ((~at-store & have-money) & take-bus) => (at-store & ~have-money))
 given
                    # 7
                    interest: ( plan-for ^@y0 have-food nil nil nil nil)
                    This is of ultimate interest
                    # 8
                    interest: ( (^@y1 & ^@y2) => have-food)
                    For interest 7 by goal-regression
```

```
                                This interest is discharged by node 8
 # 8
 ( ((at-store & have-money) & buy-food) => have-food)
 Inferred by:
            support-link #8 from { 5 } by simplify-=>
 This discharges interest 8
                                # 9
                                interest: ( plan-for ^@y3 (at-store & have-money) ( have-food) nil nil nil)
                                For interest 7 by goal-regression using node 8
                                This interest is discharged by nodes (24 62)
 # 9
 ( ((~at-store & have-money) & take-bus) => ~have-money)
 Inferred by:
            support-link #9 from { 6 } by simplify-=>
 This discharges interests (78 45)
 # 10
 ( ((~at-store & have-money) & take-bus) => at-store)
 Inferred by:
            support-link #10 from { 6 } by simplify-=>
 This discharges interest 13
                                # 10
                                interest: ( plan-for ^@y4 at-store ( (at-store & have-money) have-food) nil nil nil)
                                For interest 9 by split-conjunctive-goal
                                This interest is discharged by node 20
                                # 13
                                interest: ( (^@y7 & ^@y8) => at-store)
                                For interest 10 by goal-regression
                                This interest is discharged by node 10
                                # 14
                                interest: ( plan-for ^@y9 (~at-store & have-money) ( at-store (at-store & have-money) have-food) nil nil
                                      nil)
                               For interest 10 by goal-regression using node 10
                                This interest is discharged by node 17
                                # 15
                                interest: ( plan-for ^@y10 ~at-store ( (~at-store & have-money) at-store (at-store & have-money)
                                      have-food) nil nil nil)
                                For interest 14 by split-conjunctive-goal
                                This interest is discharged by node 11
                                # 18
                                interest: ~at-store
                                For interest 15 by null-plan
                                For interest 64 by embedded-null-plan
                                For interest 97 by null-plan
                                This interest is discharged by node 3
 # 11
 ( plan-for <plan 1> ~at-store ( (~at-store & have-money) at-store (at-store & have-money) have-food) nil nil nil)
 Inferred by:
            support-link #11 from { 3 } by null-plan
 This node is inferred by discharging a link to interest #15

Plan #1 has been constructed
      GOAL: ~at-store
          established by:
            0 --> ~at-store

                                # 20
                                interest: ( plan-for ^@y15 have-money ( (~at-store & have-money) at-store (at-store & have-money)
                                      have-food) nil nil nil)
                               For interest 14 by split-conjunctive-goal using node 11
                                This interest is discharged by node 15
                                # 21
                                interest: ( (^@y16 & ^@y17) => have-money)
                                For interest 20 by goal-regression
                                For interest 28 by goal-regression
                                This interest is discharged by node 4
                                # 22
                                interest: ( plan-for ^@y18 ~have-money ( have-money (~at-store & have-money) at-store (at-store &
                                      have-money) have-food) nil nil nil)
                                For interest 20 by goal-regression using node 4
                                This interest is discharged by node 12
                                # 24
                                interest: ~have-money
                                For interest 22 by null-plan
                                For interest 29 by null-plan
                                For interest 75 by null-plan
                                For interest 101 by null-plan
                                This interest is discharged by node 1
 # 12
 ( plan-for <plan 2> ~have-money ( have-money (~at-store & have-money) at-store (at-store & have-money) have-food) nil nil nil)
```

```
  Inferred by:
             support-link #12 from { 1 } by null-plan
  This node is inferred by discharging a link to interest #22

Plan #2 has been constructed
      GOAL: ~have-money
          established by:
            0 --> ~have-money

  # 13
  ( plan-node <pn1: beg>)
  Inferred by:
             support-link #13 from {  } by given
  This discharges interest 42
  # 15
  ( plan-for <plan 3> have-money ( (~at-store & have-money) at-store (at-store & have-money) have-food) nil nil nil)
  Inferred by:
             support-link #15 from { 4 , 12 } by goal-regression
  This node is inferred by discharging a link to interest #20

Plan #3 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      GOAL: have-money
          established by:
            1 --> have-money

  # 17
  ( plan-for <plan 4> (~at-store & have-money) ( at-store (at-store & have-money) have-food) nil nil nil)
  Inferred by:
             support-link #17 from { 11 , 15 } by split-conjunctive-goal
  This node is inferred by discharging a link to interest #14

Plan #4 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      GOAL: (~at-store & have-money)
          established by:
            0 --> ~at-store
            1 --> have-money

  # 18
  ( plan-node <pn2: take-bus>)
  Inferred by:
             support-link #18 from {  } by given
  This discharges interest 42
  # 20
  ( plan-for <plan 5> at-store ( (at-store & have-money) have-food) nil nil nil)
  Inferred by:
             support-link #20 from { 10 , 17 } by goal-regression
  This node is inferred by discharging a link to interest #10

Plan #5 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      GOAL: at-store
          established by:
            2 --> at-store

                                   # 28
                                   interest: ( plan-for ^@y21 have-money ( (at-store & have-money) have-food) nil nil nil)
                                   For interest 9 by split-conjunctive-goal using node 20
                                   This interest is discharged by node 22
                                   # 29
                                   interest: ( plan-for ^@y24 ~have-money ( have-money (at-store & have-money) have-food) nil nil nil)
                                   For interest 28 by goal-regression using node 4
                                   This interest is discharged by node 21
```

```
 # 21
 ( plan-for <plan 2> ~have-money ( have-money (at-store & have-money) have-food) nil nil nil)
 Inferred by:
              support-link #21 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #29

Plan #2 has been constructed
     GOAL: ~have-money
         established by:
           0 --> ~have-money

 # 22
 ( plan-for <plan 3> have-money ( (at-store & have-money) have-food) nil nil nil)
 Inferred by:
              support-link #22 from { 4 , 21 } by goal-regression
 This node is inferred by discharging a link to interest #28

Plan #3 has been constructed
   PLAN-STEPS:
     (1) beg
         causal-links:
           0 --~have-money--> 1
     GOAL: have-money
         established by:
           1 --> have-money

 # 23
 ( merged-plan <plan 6> <plan 5> <plan 3> ( have-food))   --  NOT USED IN PROOF
 Inferred by:
 This node is inferred by discharging links to interests (9 9)
 # 24
 ( plan-for <plan 6> (at-store & have-money) ( have-food) nil nil nil)                DEFEATED
 Inferred by:
              support-link #24 from { 20 , 22 } by split-conjunctive-goal  defeaters: { 43 }   DEFEATED
 This node is inferred by discharging links to interests (9 9)

Plan #6 has been constructed
   PLAN-STEPS:
     (1) beg
         causal-links:
           0 --~have-money--> 1
     (2) take-bus
         causal-links:
           1 --have-money--> 2
           0 --~at-store--> 2
         ordering-constraints:
           2 > 1
     GOAL: (at-store & have-money)
         established by:
           2 --> at-store
           1 --> have-money

                                   # 31
                                   interest: ((( plan-for <plan 5> at-store ( (at-store & have-money) have-food) nil nil nil) & ( plan-for <plan
                                          3> have-money ( (at-store & have-money) have-food) nil nil nil)) @ ( plan-for <plan 6> (at-store &
                                          have-money) ( have-food) nil nil nil))
                                  Of interest as a defeater for support-link 24 for node 24
                                   This interest is discharged by node 43
                                   # 32
                                   interest: ( plan-undermines-causal-links <plan 6> ( <1 --have-money--> 2> <0 --~at-store--> 2> <2
                                          --at-store--> -1> <1 --have-money--> -1> <0 --~have-money--> 1>))
                                   For interest 31 by undermine-causal-links
                                   For interest 80 by undermine-causal-links
                                   This interest is discharged by node 42
 # 27
 ( plan-for <plan 7> have-food nil nil nil nil)                DEFEATED
 Inferred by:
              support-link #27 from { 8 , 24 } by goal-regression   DEFEATED
 This node is inferred by discharging a link to interest #7

Plan #7 has been constructed
   PLAN-STEPS:
     (1) beg
         causal-links:
           0 --~have-money--> 1
     (2) take-bus
         causal-links:
           1 --have-money--> 2
           0 --~at-store--> 2
```

```
          ordering-constraints:
            2 > 1
      (3) buy-food
          causal-links:
            2 --at-store--> 3
            1 --have-money--> 3
          ordering-constraints:
            3 > 2
      GOAL: have-food
          established by:
            3 --> have-food

            =========================================
            Justified belief in ( plan-for <plan 7> have-food nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan have-food nil nil nil nil)>
            =========================================

Plan #7 has been adopted
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            1 --have-money--> 2
            0 --~at-store--> 2
          ordering-constraints:
            2 > 1
      (3) buy-food
          causal-links:
            2 --at-store--> 3
            1 --have-money--> 3
          ordering-constraints:
            3 > 2
      GOAL: have-food
          established by:
            3 --> have-food

                                # 40
                                interest: ( node-result ^@y35 ^@y36 have-money)
                                For interest 66 by embedded-goal-regression
                                This interest is discharged by node 28
                                but no inference made by discharging this interest is used in the solution.
                                # 41
                                interest: ( (^@y36 & ^@y37) => have-money)
                                For interest 40 by plan-node-result
                                This interest is discharged by node 4
                                # 42
                                interest: ( plan-node ^@y38)
                                For interest 40 by plan-node-result using node 4
                                For interest 44 by plan-node-result using node 9
                                For interest 44 by plan-node-result using node 7
                                This interest is discharged by nodes (18 13)
 # 28
 ( node-result <pn1: beg> ~have-money have-money)
 Inferred by:
            support-link #28 from { 4 , 13 } by plan-node-result
 This node is inferred by discharging a link to interest #40
 This discharges interest 100
                                # 44
                                interest: ( node-result ^@y40 ^@y41 ~have-money)
                                For interest 61 by embedded-goal-regression
                                This interest is discharged by node 29
                                # 45
                                interest: ( (^@y41 & ^@y42) => ~have-money)
                                For interest 44 by plan-node-result
                                This interest is discharged by node 9
 # 29
 ( node-result <pn2: take-bus> (~at-store & have-money) ~have-money)
 Inferred by:
            support-link #29 from { 9 , 18 } by plan-node-result
 This node is inferred by discharging a link to interest #44
                                # 49
                                interest: ( plan-undermines-causal-links <plan 6> ( <0 --~at-store--> 2> <2 --at-store--> -1> <1 --have-
                                      money--> -1> <0 --~have-money--> 1>))
                                For interest 32 by plan-undermines-another-causal-link
                                This interest is discharged by node 41
                                # 52
```

```
                                 interest: ( plan-undermines-causal-links <plan 6> ( <2 --at-store--> -1> <1 --have-money--> -1> <0
                                        --~have-money--> 1>))
                                 For interest 49 by plan-undermines-another-causal-link
                                 This interest is discharged by node 40
                                 # 55
                                 interest: ( plan-undermines-causal-links <plan 6> ( <1 --have-money--> -1> <0 --~have-money--> 1>))
                                 For interest 52 by plan-undermines-another-causal-link
                                 This interest is discharged by node 39
                                 # 58
                                 interest: ( plan-undermines-causal-link <plan 6> ^@y63 ^@y62 <1 --have-money--> -1>)
                                 For interest 55 by plan-undermines-first-causal-link
                                 This interest is discharged by node 38
                                 # 61
                                 interest: ( embellished-plan-for ^@y69 <plan 6> ~have-money <pn1: beg> *finish* ( (<pn1: beg> . <pn2:
                                        take-bus>) (<pn2: take-bus> . *finish*)) nil)
                                 For interest 58 by plan-undermines-causal-link
                                 This interest is discharged by node 37
                                 # 62
                                 interest: ( embellished-plan-for ^@y72 <plan 6> (~at-store & have-money) nil <pn2: take-bus> ( (<pn1:
                                        beg> . <pn2: take-bus>) (<pn2: take-bus> . *finish*)) nil)
                                 For interest 61 by embedded-goal-regression using node 29
                                 This interest is discharged by node 35
                                 # 64
                                 interest: ( embellished-plan-for ^@y75 <plan 6> ~at-store nil <pn2: take-bus> ( (<pn1: beg> . <pn2:
                                        take-bus>) (<pn2: take-bus> . *finish*)) nil)
                                 For interest 62 by split-embedded-conjunctive-goal
                                 This interest is discharged by node 32
 # 32
 ( embellished-plan-for <plan 8> <plan 6> ~at-store nil <pn2: take-bus> ( (<pn1: beg> . <pn2: take-bus>) (<pn2: take-bus> .
          *finish*)) nil)
 Inferred by:
          support-link #32 from { 3 } by embedded-null-plan
 This node is inferred by discharging a link to interest #64

Plan #8 has been constructed
   PLAN-STEPS:
     GOAL: ~at-store
         established by:
           0 --> ~at-store
     (1) beg
         ordering-constraints:
           1 > *finish*
     (2) take-bus
         ordering-constraints:
           2 > 1

                                 # 66
                                 interest: ( embellished-plan-for ^@y78 <plan 6> have-money nil <pn2: take-bus> ( (*finish* . <pn1:
                                        beg>) (<pn1: beg> . <pn2: take-bus>)) nil)
                                 For interest 62 by split-embedded-conjunctive-goal using node 32
                                 This interest is discharged by node 33
 # 33
 ( embellished-plan-for <plan 9> <plan 6> have-money nil <pn2: take-bus> ( (*finish* . <pn1: beg>) (<pn1: beg> . <pn2:
          take-bus>)) nil)
 Inferred by:
          support-link #33 from { 22 } by embellished-plan-for-goal
 This discharges interest 66

Plan #9 has been constructed
   PLAN-STEPS:
     (1) beg
         causal-links:
           0 --~have-money--> 1
     GOAL: have-money
         established by:
           1 --> have-money
     (2) take-bus
         ordering-constraints:
           2 > *finish*

 # 35
 ( embellished-plan-for <plan 10> <plan 6> (~at-store & have-money) nil <pn2: take-bus> ( (<pn1: beg> . <pn2: take-bus>)
          (<pn2: take-bus> . *finish*)) nil)
 Inferred by:
          support-link #35 from { 32 , 33 } by split-embedded-conjunctive-goal
 This node is inferred by discharging a link to interest #62

Plan #10 has been constructed
   PLAN-STEPS:
```

```
      (1) beg
          causal-links:
            0 --~have-money--> 1
      GOAL: (~at-store & have-money)
          established by:
            0 --> ~at-store
            1 --> have-money
      (2) take-bus
          ordering-constraints:
            2 > *finish*

 # 37
 ( embellished-plan-for <plan 11> <plan 6> ~have-money <pn1: beg> *finish* ( (<pn1: beg> . <pn2: take-bus>) (<pn2: take-bus>
           . *finish*)) nil)
 Inferred by:
             support-link #37 from { 29 , 35 } by embedded-goal-regression
 This node is inferred by discharging a link to interest #61

Plan #11 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      GOAL: ~have-money
          established by:
            2 --> ~have-money

 # 38
 ( plan-undermines-causal-link <plan 6> (~at-store & have-money) <pn2: take-bus> <1 --have-money--> -1>)
 Inferred by:
             support-link #38 from { 37 } by plan-undermines-causal-link
 This node is inferred by discharging a link to interest #58
 # 39
 ( plan-undermines-causal-links <plan 6> ( <1 --have-money--> -1> <0 --~have-money--> 1>))
 Inferred by:
             support-link #39 from { 38 } by plan-undermines-first-causal-link
 This node is inferred by discharging a link to interest #55
 # 40
 ( plan-undermines-causal-links <plan 6> ( <2 --at-store--> -1> <1 --have-money--> -1> <0 --~have-money--> 1>))
 Inferred by:
             support-link #40 from { 39 } by plan-undermines-another-causal-link
 This node is inferred by discharging a link to interest #52
 # 41
 ( plan-undermines-causal-links <plan 6> ( <0 --~at-store--> 2> <2 --at-store--> -1> <1 --have-money--> -1> <0 --~have-money--
           > 1>))
 Inferred by:
             support-link #41 from { 40 } by plan-undermines-another-causal-link
 This node is inferred by discharging a link to interest #49
 # 42
 ( plan-undermines-causal-links <plan 6> ( <1 --have-money--> 2> <0 --~at-store--> 2> <2 --at-store--> -1> <1 --have-money-->
           -1> <0 --~have-money--> 1>))
 Inferred by:
             support-link #42 from { 41 } by plan-undermines-another-causal-link
 This node is inferred by discharging a link to interest #32
 # 43
 ((( plan-for <plan 5> at-store ( (at-store & have-money) have-food) nil nil nil) & ( plan-for <plan 3> have-money ( (at-store &
           have-money) have-food) nil nil nil)) @ ( plan-for <plan 6> (at-store & have-money) ( have-food) nil nil nil))
 Inferred by:
             support-link #43 from { 42 } by undermine-causal-links
 defeatees: { link 24 for node 24 }
 This node is inferred by discharging a link to interest #31
            ========================================
            Lowering the undefeated-degree-of-support of ( plan-for <plan 7> have-food nil nil nil nil)
            retracts the previous answer to #<Query 1: (? plan)( plan-for plan have-food nil nil nil nil)>
            ========================================

Plan #7 has been retracted:
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
```

```
          1 --have-money--> 2
          0 --~at-store--> 2
        ordering-constraints:
          2 > 1
     (3) buy-food
        causal-links:
          2 --at-store--> 3
          1 --have-money--> 3
        ordering-constraints:
          3 > 2
     GOAL: have-food
        established by:
          3 --> have-food

 # 44
 ( adds-ordering-constraints nil <plan 6> <pn2: take-bus> <1 --have-money--> -1>)   --  NOT USED IN PROOF
 Inferred by:
          support-link #44 from {  } by given
                         # 71
                         interest: ( plan-for ^@y82 have-money ( (at-store & have-money) have-food) ( <pn1: beg> <pn2:
                            take-bus>) nil nil)
                         For interest 9 by reuse-nodes-2 using node 20 with clues (22 23 44 38)
                         This interest is discharged by nodes (46 60)
                         # 74
                         interest: ( (^@y86 & ^@y87) => have-money)
                         For interest 71 by goal-regression
                         This interest is discharged by node 4
                         # 75
                         interest: ( plan-for ^@y88 ~have-money ( have-money (at-store & have-money) have-food) ( <pn1:
                            beg> <pn2: take-bus>) nil nil)
                         For interest 71 by goal-regression using node 4
                         This interest is discharged by nodes (45 56)
                         # 78
                         interest: ( (^@y92 & ^@y93) => ~have-money)
                         For interest 75 by goal-regression
                         This interest is discharged by node 9
                         # 79
                         interest: ( plan-for ^@y94 (~at-store & have-money) ( ~have-money have-money (at-store & have-
                            money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
                         For interest 75 by goal-regression using node 9
                         This interest is discharged by node 54
 # 45
 ( plan-for <plan 2> ~have-money ( have-money (at-store & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
 Inferred by:
          support-link #45 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #75

Plan #2 has been constructed
     GOAL: ~have-money
        established by:
          0 --> ~have-money

 # 46
 ( plan-for <plan 3> have-money ( (at-store & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
 Inferred by:
          support-link #46 from { 4 , 45 } by goal-regression
 This node is inferred by discharging a link to interest #71

Plan #3 has been constructed
   PLAN-STEPS:
     (1) beg
        causal-links:
          0 --~have-money--> 1
     GOAL: have-money
        established by:
          1 --> have-money

 # 23
 ( merged-plan <plan 6> <plan 5> <plan 3> ( have-food))   --  NOT USED IN PROOF
 Inferred by:
          support-link #47 from { 20 , 46 } by reuse-nodes-2 with clues { 22 , 23 , 44 , 38 }
 This node is inferred by discharging links to interests (9 9)
 # 24
 ( plan-for <plan 6> (at-store & have-money) ( have-food) nil nil nil)                DEFEATED
 Inferred by:
          support-link #48 from { 20 , 46 } by reuse-nodes-2 with clues { 22 , 23 , 44 , 38 }  defeaters: { 47 }   DEFEATED
          support-link #24 from { 20 , 22 } by split-conjunctive-goal  defeaters: { 43 }   DEFEATED
 This node is inferred by discharging links to interests (9 9)
```

```
Plan #6 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            1 --have-money--> 2
            0 --~at-store--> 2
          ordering-constraints:
            2 > 1
      GOAL: (at-store & have-money)
          established by:
            2 --> at-store
            1 --> have-money

                                  # 80
                                  interest: ((( plan-for <plan 5> at-store ( (at-store & have-money) have-food) nil nil nil) & ( plan-for <plan
                                        3> have-money ( (at-store & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)) @ (
                                        plan-for <plan 6> (at-store & have-money) ( have-food) nil nil nil))
                                  Of interest as a defeater for support-link 48 for node 24
                                  This interest is discharged by node 47
  # 47
  ((( plan-for <plan 5> at-store ( (at-store & have-money) have-food) nil nil nil) & ( plan-for <plan 3> have-money ( (at-store &
            have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)) @ ( plan-for <plan 6> (at-store & have-money) (
            have-food) nil nil nil))
  Inferred by:
             support-link #49 from { 42 } by undermine-causal-links
  defeatees: { link 48 for node 24 }
  This node is inferred by discharging a link to interest #80
                                  # 97
                                  interest: ( plan-for ^@y127 ~at-store ( (~at-store & have-money) ~have-money have-money (at-store &
            have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
                                  For interest 79 by split-conjunctive-goal
                                  This interest is discharged by node 50
  # 50
  ( plan-for <plan 1> ~at-store ( (~at-store & have-money) ~have-money have-money (at-store & have-money) have-food) ( <pn1:
            beg> <pn2: take-bus>) nil nil)
  Inferred by:
             support-link #52 from { 3 } by null-plan
  This node is inferred by discharging a link to interest #97

Plan #1 has been constructed
      GOAL: ~at-store
          established by:
            0 --> ~at-store

                                  # 99
                                  interest: ( plan-for ^@y134 have-money ( (~at-store & have-money) ~have-money have-money (at-store
                                        & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
                                 For interest 79 by split-conjunctive-goal using node 50
                                  This interest is discharged by node 52
                                  # 100
                                  interest: ( node-result ^@y136 ^@y135 have-money)
                                  For interest 99 by reuse-node
                                  This interest is discharged by node 28
                                  # 101
                                  interest: ( plan-for ^@y137 ~have-money ( (~at-store & have-money) ~have-money have-money
                                        (at-store & have-money) have-food) ( <pn2: take-bus>) ( <pn1: beg>) nil)
                                  For interest 99 by reuse-node using node 28
                                  This interest is discharged by node 51
  # 51
  ( plan-for <plan 2> ~have-money ( (~at-store & have-money) ~have-money have-money (at-store & have-money) have-food) (
            <pn2: take-bus>) ( <pn1: beg>) nil)
  Inferred by:
             support-link #53 from { 1 } by null-plan
  This node is inferred by discharging a link to interest #101

Plan #2 has been constructed
      GOAL: ~have-money
          established by:
            0 --> ~have-money

  # 52
  ( plan-for <plan 3> have-money ( (~at-store & have-money) ~have-money have-money (at-store & have-money) have-food) (
            <pn1: beg> <pn2: take-bus>) nil nil)
  Inferred by:
             support-link #54 from { 28 , 51 } by reuse-node
  This node is inferred by discharging a link to interest #99
```

```
Plan #3 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      GOAL: have-money
          established by:
            1 --> have-money

  # 54
  ( plan-for <plan 4> (~at-store & have-money) ( ~have-money have-money (at-store & have-money) have-food) ( <pn1: beg>
            <pn2: take-bus>) nil nil)
  Inferred by:
               support-link #56 from { 50 , 52 } by split-conjunctive-goal
  This node is inferred by discharging a link to interest #79

Plan #4 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      GOAL: (~at-store & have-money)
          established by:
            0 --> ~at-store
            1 --> have-money

  # 56
  ( plan-for <plan 11> ~have-money ( have-money (at-store & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
  Inferred by:
               support-link #58 from { 9 , 54 } by goal-regression
  This node is inferred by discharging a link to interest #75

Plan #11 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      GOAL: ~have-money
          established by:
            2 --> ~have-money

  # 60
  ( plan-for <plan 12> have-money ( (at-store & have-money) have-food) ( <pn1: beg> <pn2: take-bus>) nil nil)
  Inferred by:
               support-link #62 from { 4 , 56 } by goal-regression
  This node is inferred by discharging a link to interest #71

Plan #12 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      (4) beg
          causal-links:
            2 --~have-money--> 4
          ordering-constraints:
            4 > 2
      GOAL: have-money
          established by:
            4 --> have-money

  # 62
  ( plan-for <plan 13> (at-store & have-money) ( have-food) nil nil nil)
  Inferred by:
               support-link #64 from { 20 , 60 } by reuse-nodes-2 with clues { 22 , 23 , 44 , 38 }
  This node is inferred by discharging a link to interest #9
```

```
Plan #13 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      (4) beg
          causal-links:
            2 --~have-money--> 4
          ordering-constraints:
            4 > 2
      GOAL: (at-store & have-money)
          established by:
            2 --> at-store
            4 --> have-money

  # 64
  ( plan-for <plan 14> have-food nil nil nil nil)
  Inferred by:
              support-link #66 from { 8 , 62 } by goal-regression
  This node is inferred by discharging a link to interest #7

Plan #14 has been constructed
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      (4) beg
          causal-links:
            2 --~have-money--> 4
          ordering-constraints:
            4 > 2
      (3) buy-food
          causal-links:
            2 --at-store--> 3
            4 --have-money--> 3
          ordering-constraints:
            3 > 4
      GOAL: have-food
          established by:
            3 --> have-food

            ===========================================
            Justified belief in ( plan-for <plan 14> have-food nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan have-food nil nil nil nil)>
            ===========================================

Plan #14 has been adopted
    PLAN-STEPS:
      (1) beg
          causal-links:
            0 --~have-money--> 1
      (2) take-bus
          causal-links:
            0 --~at-store--> 2
            1 --have-money--> 2
          ordering-constraints:
            2 > 1
      (4) beg
          causal-links:
            2 --~have-money--> 4
          ordering-constraints:
            4 > 2
      (3) buy-food
          causal-links:
            2 --at-store--> 3
```

```
        4 --have-money--> 3
      ordering-constraints:
        3 > 4
  GOAL: have-food
      established by:
        3 --> have-food

=================================================================


================= ULTIMATE EPISTEMIC INTERESTS ===================
 Interest in (? plan)( plan-for plan have-food nil nil nil nil)
 is answered by node 64:  ( plan-for <plan 14> have-food nil nil nil nil)
---------------------------------------------------
```

8. Resolving Threats by Confrontation

One way of resolving threats to plans constructed by applying SPLIT-CONJUNCTIVE-GOAL is to impose ordering-constraints. That is all that is required for planning problems that can be formulated using the STRIPS representation.[11] However, more expressive formalisms require additional machinery. UCPOP extends the STRIPS representation by allowing operators to have conditional effects. Allowing operators to have conditional effects just amounts to observing that the same action can have different consequences under different circumstances. OSCAR's very general representation of actions and their consequences (using the conditionals '⇒') automatically allows for conditional effects, and thus requires additional machinery for resolving threats.

Penberthy and Weld [1992] introduce a second way of resolving threats, called *confrontation*,, and prove that it renders UCPOP complete when planning using operators with conditional effects. Their observations can be recast in the present formalsim. A threat arises when *plan*& is constructed by SPLIT-CONJUNCTIVE-GOAL and *plan*& undermines one of its own causal-links (where those causal-links are inherited from the plans $plan_1$ and $plan_2$ for the individual conjuncts). This in turn occurs when one of the plans $plan_1$ and $plan_2$ contains a causal-link $node_1$ *--goal-->* $node_2$, and *plan&* contains an embellished-plan *plan** for *~goal*. *Plan**'s achieving *~goal* will either depend upon facts holding in the start-state and continuing to hold until some steps of *plan** are executed, or facts brought about by some steps of *plan&*. If *plan&* does not require those same facts to continue to hold to the point at which the undermining occurs, then it may be possible to add some steps to *plan&* that will have the result that one or more of those facts do not continue to hold. In other words, we can add some steps to *plan&* that will have the effect of undermining a causal-link of *plan** without at the same time undermining a causal-link of *plan&*. This strategy can be formulated as follows:

CONFRONTATION

Given an interest in finding a plan for achieving a conjunctive goal (g_1 & g_2), and plans $plan_1$ for g_1 and $plan_2$ for g_2, if *plan&* is a putative plan for (g_1 & g_2) constructed by merging plans $plan_1$ and $plan_2$ (and possibly other plans), but a plan-step n of *plan&* undermines one of its own causal-links $n_1 \rightarrow subgoal \rightarrow n_2$ by virtue of there being an embellishment $plan_0$ that achieves P where P is either $\sim subgoal_1$ or the negation of a conjunct of $subgoal_1$, then for each pair of causal-links $n_0 \rightarrow G \rightarrow n \rightarrow P$ and $n \rightarrow P \rightarrow$ **finish** of $plan_0$, adopt interest in finding a plan for achieving $\sim G$ (or if G is a conjunction, for each conjunct of G, adopt interest in finding a plan for achieving its negation). If a plan *repair-plan* is proposed for achieving $\sim G$ or the negation of one of its conjuncts, construct a new plan *plan+* by adding to *plan&* the plan-steps, ordering-constraints, and causal-links of *repair-plan*, with the following exception. Replace each causal-link of the form $n^* \rightarrow SG \rightarrow$

[11] This follows from the proof of the completeness of SNLP in McAllister and Rosenblitt [1991].

*finish** in *repair-plan* by the causal-link $n^* \rightarrow SG \rightarrow n$ and order n^* before n. If this ordering is consistent, infer defeasibly that *plan*+ will achieve (g_1 & g_2).

This means of resolving threats can be illustrated by an example due to Pednault [1988]. Suppose my briefcase contains my paycheck, and is currently at home. I want my briefcase to be at my office, but I want my paycheck to be at home. The obvious plan is to remove the paycheck from the briefcase and then take the briefcase to the office. Using the reason-schemas discussed above, OSCAR will construct the following plan:

```
Plan #4 has been constructed:
  PLAN-STEPS:
    (1) take-briefcase-to-office
        causal-links:
          0 --( at-home briefcase)--> 1
    GOAL: (( at-office briefcase) & ( at-home paycheck))
        established by:
          1 --> ( at-office briefcase)
          0 --> ( at-home paycheck)
```

However, the causal-link `0 --(at-home paycheck)--> *finish*` of the goal-node is undermined by plan-node 1, so the plan is retracted. To repair this plan, we can note that plan-node 1 only undermines the causal-link because the paycheck is in the briefcase. That is, *plan** is the following:

```
Plan #8 has been constructed
  PLAN-STEPS:
    (1) take-briefcase-to-office
        causal-links:
          0 --( at-home briefcase)--> 1
          0 --( in-briefcase paycheck)--> 1
    GOAL: ~( at-home paycheck)
        established by:
          1 --> ~( at-home paycheck)
```

We can resolve this threat by adding to plan #4 a step that removes the paycheck from the briefcase before the briefcase is taken to the office:

```
Plan #11 has been constructed
  PLAN-STEPS:
    (2) ( remove-from-briefcase paycheck)
        causal-links:
          0 --( in-briefcase paycheck)--> 2
    (1) take-briefcase-to-office
        causal-links:
          2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1
          0 --( at-home briefcase)--> 1
        ordering-constraints:
          1 > 2
    GOAL: (( at-office briefcase) & ( at-home paycheck))
        established by:
          1 --> ( at-office briefcase)
          0 --> ( at-home paycheck)
```

The plan-steps of plan #8 are still contained in those of plan #11, however plan #8 is not an embellished-plan of plan #11, because the causal-link `0 --( in-briefcase paycheck)--> 1` is undermined by node 2. Formally, the defeasible argument (using EMBELLISHED-PLAN-FOR) to the effect that plan #8 is an embellished-plan of plan #11 is defeated by UNDERMINE-EMBEDDED-CAUSAL-LINKS.

To enable OSCAR to find plan #11, we must add a reason-schema. Plan #8 is obtained by applying PLAN-UNDERMINES-CAUSAL-LINK. It is observed that plan-node 1 has the result "~(at-home paycheck)" provided "((at-home briefcase) & (in-briefcase paycheck))" holds. Then OSCAR looks for an embellished-plan for the latter. In this case, a null-plan suffices. More generally, a plan constructed out of plan-nodes of plan #5 might be found. Plan #5 can be repaired by adding steps constituting a "repair-plan" for the negation of the causal-link-goal of some causal-link for the penultimate node of the embellished-plan. To achieve the proper ordering, we add causal-links to the call-set of plan-node 1. For each causal-link $n \rightarrow \sim R \rightarrow$ **finish** for the goal-node of the repair-plan, we add a causal-link $n \rightarrow \sim R \rightarrow 1$ to the call-set of plan-node 1. This is accomplished with the following backwards-reason-schema:

```
(def-backwards-reason CONFRONTATION
   :conclusions
   "(plan-for plan& goal goals nodes nodes-used links)"
   "(merged-plan plan& repair-plan plan1 goals)"
     (:defeasible? nil)
   :condition  (interest-variable plan&)
   :forwards-premises
      "(plan-for plan goal goals nodes nodes-used links)"
      (:clue? t)
      "(merged-plan plan plan1 plan2 goals)"
      "(plan-undermines-causal-link plan R node link)"
      (:clue? t)
   :backwards-premises
      "(define -R (neg R))"
      "(plan-for repair-plan -R nil nodes nodes-used links)"
      "(define plan& (make-confrontation-plan repair-plan plan -R node links))"
      (:condition (not (null plan&)))
   :defeasible? t
   :variables  plan1 plan2 plan& plan R -R repair-plan node link goal goals nodes nodes-used links)
```

The function (MAKE-CONFRONTATION-PLAN *repair-plan pla& -R node links*) produces a new plan by merging *repair-plan* and *plan* and englarging the call-set of *node* as described above. With the addition of this reason-schema, the Pednault problem is done as follows:

```
(
======================================================================

                    OSCAR_3.25        4/17/1998        15:41:43
                          Non-Linear-Planner-31

Problem number 14:  Pednault's briefcase example.

Forwards-substantive-reasons:
        simplify-=>

Backwards-substantive-reasons:
        null-plan
        goal-regression
        split-conjunctive-goal
        reuse-nodes-1
        reuse-nodes-2
        reuse-node
        plan-node-result
        =>-adjunction
        =>-neg1
        =>-neg2
        undermine-causal-links
        undermine-embedded-causal-links
        plan-undermines-first-causal-link
        plan-undermines-another-causal-link
        plan-undermines-causal-link
        embellished-plan-for-goal
        embedded-null-plan
        split-embedded-conjunctive-goal
        embedded-goal-regression
        add-ordering-constraints
        add-embedded-ordering-constraints
        confrontation
        embedded-confrontation
```

```
Goal-state:
        ( at-office briefcase)
        ( at-home paycheck)

Inputs:

Given:
        ( at-home briefcase)  :  with justification = 0.99
        ( at-home paycheck)  :  with justification = 0.99
        ( in-briefcase paycheck)  :  with justification = 0.99
        (all x)( (( in-briefcase x) & ( remove-from-briefcase x)) => ~( in-briefcase x))  :  with justification = 0.99
        ( (( at-home briefcase) & take-briefcase-to-office) => ( at-office briefcase))  :  with justification = 0.99
        (all x)( ((( at-home briefcase) & ( in-briefcase x)) & take-briefcase-to-office) => ~( at-home x))  :  with justification = 0.99

============================================================================
THE FOLLOWING IS THE REASONING INVOLVED IN THE SOLUTION
Nodes marked DEFEATED have that status at the end of the reasoning.

 # 1
 ( at-home briefcase)
 given
 This discharges interest 15
 # 2
 ( at-home paycheck)
 given
 This discharges interest 18
 # 3
 ( in-briefcase paycheck)
 given
 This discharges interest 44
 # 4
 (all x)( (( in-briefcase x) & ( remove-from-briefcase x)) => ~( in-briefcase x))
 given
 # 5
 ( (( at-home briefcase) & take-briefcase-to-office) => ( at-office briefcase))
 given
 This discharges interest 11
 # 6
 (all x)( ((( at-home briefcase) & ( in-briefcase x)) & take-briefcase-to-office) => ~( at-home x))
 given
                                 # 7
                                 interest: ( plan-for ^@y0 (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)
                                 This is of ultimate interest
                                 # 8
                                 interest: ( plan-for ^@y1 ( at-office briefcase) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil)
                                 For interest 7 by split-conjunctive-goal
                                 This interest is discharged by node 12
 # 7
 ( (( in-briefcase x4) & ( remove-from-briefcase x4)) => ~( in-briefcase x4))
 Inferred by:
           support-link #7 from { 4 } by UI
 This discharges interest 62
 # 8
 ( ((( at-home briefcase) & ( in-briefcase x5)) & take-briefcase-to-office) => ~( at-home x5))
 Inferred by:
           support-link #8 from { 6 } by UI
 This discharges interest 36
                                 # 11
                                 interest: ( (^@y6 & ^@y7) => ( at-office briefcase))
                                 For interest 8 by goal-regression
                                 This interest is discharged by node 5
                                 # 12
                                 interest: ( plan-for ^@y8 ( at-home briefcase) ( ( at-office briefcase) (( at-office briefcase) & ( at-home
                                       paycheck))) nil nil nil)
                                 For interest 8 by goal-regression using node 5
                                 This interest is discharged by node 9
                                 # 15
                                 interest: ( at-home briefcase)
                                 For interest 12 by null-plan
                                 For interest 39 by embedded-null-plan
                                 For interest 93 by embedded-null-plan
                                 This interest is discharged by node 1
 # 9
 ( plan-for <plan 1> ( at-home briefcase) ( ( at-office briefcase) (( at-office briefcase) & ( at-home paycheck))) nil nil nil)
 Inferred by:
           support-link #9 from { 1 } by null-plan
 This node is inferred by discharging a link to interest #12

Plan #1 has been constructed
```

```
      GOAL: ( at-home briefcase)
          established by:
            0 --> ( at-home briefcase)

 # 10
 ( plan-node <pn1: take-briefcase-to-office>)
 Inferred by:
             support-link #10 from {  } by given
 This discharges interest 34
 # 12
 ( plan-for <plan 2> ( at-office briefcase) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil)
 Inferred by:
             support-link #12 from { 5 , 9 } by goal-regression
 This node is inferred by discharging a link to interest #8

Plan #2 has been constructed
    PLAN-STEPS:
      (1) take-briefcase-to-office
          causal-links:
            0 --( at-home briefcase)--> 1
      GOAL: ( at-office briefcase)
          established by:
            1 --> ( at-office briefcase)

                              # 16
                              interest: ( plan-for ^@y11 ( at-home paycheck) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil)
                              For interest 7 by split-conjunctive-goal using node 12
                              This interest is discharged by node 13
                              # 18
                              interest: ( at-home paycheck)
                              For interest 16 by null-plan
                              This interest is discharged by node 2
 # 13
 ( plan-for <plan 3> ( at-home paycheck) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil)
 Inferred by:
             support-link #13 from { 2 } by null-plan
 This node is inferred by discharging a link to interest #16

Plan #3 has been constructed
      GOAL: ( at-home paycheck)
          established by:
            0 --> ( at-home paycheck)

 # 14
 ( merged-plan <plan 4> <plan 2> <plan 3> nil)
 Inferred by:
             support-link #14 from { 12 , 13 } by split-conjunctive-goal
 This node is inferred by discharging a link to interest #7
 # 15
 ( plan-for <plan 4> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)            DEFEATED
 Inferred by:
             support-link #15 from { 12 , 13 } by split-conjunctive-goal  defeaters: { 28 }   DEFEATED
 This node is inferred by discharging a link to interest #7

Plan #4 has been constructed
    PLAN-STEPS:
      (1) take-briefcase-to-office
          causal-links:
            0 --( at-home briefcase)--> 1
      GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)

                              # 19
                              interest: ((( plan-for <plan 2> ( at-office briefcase) ( (( at-office briefcase) & ( at-home paycheck))) nil nil
                                    nil) & ( plan-for <plan 3> ( at-home paycheck) ( (( at-office briefcase) & ( at-home paycheck))) nil nil
                                    nil)) @ ( plan-for <plan 4> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil))
                             Of interest as a defeater for support-link 15 for node 15
                              This interest is discharged by node 28
                              # 20
                              interest: ( plan-undermines-causal-links <plan 4> ( <0 --( at-home briefcase)--> 1> <1 --( at-office
                                    briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
                              For interest 19 by undermine-causal-links
                              This interest is discharged by node 27
            ========================================
            Justified belief in ( plan-for <plan 4> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)>
```

```
           ========================================

Plan #4 has been adopted
   PLAN-STEPS:
     (1) take-briefcase-to-office
          causal-links:
            0 --( at-home briefcase)--> 1
     GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)

                              # 22
                              interest: ( plan-undermines-causal-links <plan 4> ( <1 --( at-office briefcase)--> -1> <0 --( at-home
                                    paycheck)--> -1>))
                              For interest 20 by plan-undermines-another-causal-link
                              This interest is discharged by node 26
                              # 24
                              interest: ( plan-undermines-causal-links <plan 4> ( <0 --( at-home paycheck)--> -1>))
                              For interest 22 by plan-undermines-another-causal-link
                              This interest is discharged by node 25
                              # 26
                              interest: ( plan-undermines-causal-link <plan 4> ^@y19 ^@y18 <0 --( at-home paycheck)--> -1>)
                              For interest 24 by plan-undermines-first-causal-link
                              This interest is discharged by node 24
                              # 27
                              interest: ( embellished-plan-for ^@y20 <plan 4> ~( at-home paycheck) nil *finish* ( (<pn1: take-
                                    briefcase-to-office> . *finish*)) nil)
                              For interest 26 by plan-undermines-causal-link
                              This interest is discharged by node 23
                              # 28
                              interest: ( node-result ^@y21 ^@y22 ~( at-home paycheck))
                              For interest 27 by embedded-goal-regression
                              For interest 91 by embedded-goal-regression
                              This interest is discharged by node 17
                              # 34
                              interest: ( plan-node ^@y30)
                              For interest 28 by plan-node-result using node 8
                              For interest 61 by plan-node-result using node 7
                              This interest is discharged by nodes (10 36)
                              # 36
                              interest: ( (^@y22 & ^@y32) => ~( at-home paycheck))
                              For interest 28 by plan-node-result
                              This interest is discharged by node 8
 # 17
 ( node-result <pn1: take-briefcase-to-office> (( at-home briefcase) & ( in-briefcase paycheck)) ~( at-home paycheck))
 Inferred by:
           support-link #17 from { 8 , 10 } by plan-node-result
 This node is inferred by discharging a link to interest #28
                              # 37
                              interest: ( embellished-plan-for ^@y34 <plan 4> (( at-home briefcase) & ( in-briefcase paycheck)) nil
                                    <pn1: take-briefcase-to-office> ( (<pn1: take-briefcase-to-office> . *finish*)) nil)
                              For interest 27 by embedded-goal-regression using node 17
                              This interest is discharged by node 21
                              # 39
                              interest: ( embellished-plan-for ^@y37 <plan 4> ( at-home briefcase) nil <pn1: take-briefcase-to-office> (
                                    (<pn1: take-briefcase-to-office> . *finish*)) nil)
                              For interest 37 by split-embedded-conjunctive-goal
                              This interest is discharged by node 18
 # 18
 ( embellished-plan-for <plan 5> <plan 4> ( at-home briefcase) nil <pn1: take-briefcase-to-office> ( (<pn1: take-briefcase-to-
        office> . *finish*)) nil)
 Inferred by:
           support-link #18 from { 1 } by embedded-null-plan
 This node is inferred by discharging a link to interest #39

Plan #5 has been constructed
   PLAN-STEPS:
     GOAL: ( at-home briefcase)
          established by:
            0 --> ( at-home briefcase)
     (1) take-briefcase-to-office
          ordering-constraints:
            1 > *finish*

                              # 42
                              interest: ( embellished-plan-for ^@y40 <plan 4> ( in-briefcase paycheck) nil <pn1: take-briefcase-to-
                                    office> ( (*finish* . <pn1: take-briefcase-to-office>)) nil)
                              For interest 37 by split-embedded-conjunctive-goal using node 18
```

```
                                        This interest is discharged by node 19
                                        # 44
                                        interest: ( in-briefcase paycheck)
                                        For interest 42 by embedded-null-plan
                                        For interest 74 by null-plan
                                        For interest 94 by embedded-null-plan
                                        For interest 106 by embedded-null-plan
                                        This interest is discharged by node 3
 # 19
 ( embellished-plan-for <plan 6> <plan 4> ( in-briefcase paycheck) nil <pn1: take-briefcase-to-office> ( (*finish* . <pn1: take-
        briefcase-to-office>)) nil)
 Inferred by:
         support-link #19 from { 3 } by embedded-null-plan
 This node is inferred by discharging a link to interest #42

Plan #6 has been constructed
   PLAN-STEPS:
     GOAL: ( in-briefcase paycheck)
         established by:
           0 --> ( in-briefcase paycheck)
     (1) take-briefcase-to-office
         ordering-constraints:
           1 > *finish*

 # 21
 ( embellished-plan-for <plan 7> <plan 4> (( at-home briefcase) & ( in-briefcase paycheck)) nil <pn1: take-briefcase-to-office> (
        (<pn1: take-briefcase-to-office> . *finish*)) nil)
 Inferred by:
         support-link #21 from { 18 , 19 } by split-embedded-conjunctive-goal
 This node is inferred by discharging a link to interest #37

Plan #7 has been constructed
   PLAN-STEPS:
     GOAL: (( at-home briefcase) & ( in-briefcase paycheck))
         established by:
           0 --> ( at-home briefcase)
           0 --> ( in-briefcase paycheck)
     (1) take-briefcase-to-office
         ordering-constraints:
           1 > *finish*

 # 23
 ( embellished-plan-for <plan 8> <plan 4> ~( at-home paycheck) nil *finish* ( (<pn1: take-briefcase-to-office> . *finish*)) nil)
 Inferred by:
         support-link #23 from { 17 , 21 } by embedded-goal-regression
 This node is inferred by discharging a link to interest #27

Plan #8 has been constructed
   PLAN-STEPS:
     (1) take-briefcase-to-office
         causal-links:
           0 --( at-home briefcase)--> 1
           0 --( in-briefcase paycheck)--> 1
     GOAL: ~( at-home paycheck)
         established by:
           1 --> ~( at-home paycheck)

 # 24
 ( plan-undermines-causal-link <plan 4> (( at-home briefcase) & ( in-briefcase paycheck)) <pn1: take-briefcase-to-office> <0 --(
        at-home paycheck)--> -1>)
 Inferred by:
         support-link #24 from { 23 } by plan-undermines-causal-link
 This node is inferred by discharging a link to interest #26
 # 25
 ( plan-undermines-causal-links <plan 4> ( <0 --( at-home paycheck)--> -1>))
 Inferred by:
         support-link #25 from { 24 } by plan-undermines-first-causal-link
 This node is inferred by discharging a link to interest #24
 # 26
 ( plan-undermines-causal-links <plan 4> ( <1 --( at-office briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
 Inferred by:
         support-link #26 from { 25 } by plan-undermines-another-causal-link
 This node is inferred by discharging a link to interest #22
 # 27
 ( plan-undermines-causal-links <plan 4> ( <0 --( at-home briefcase)--> 1> <1 --( at-office briefcase)--> -1> <0 --( at-home
        paycheck)--> -1>))
 Inferred by:
         support-link #27 from { 26 } by plan-undermines-another-causal-link
 This node is inferred by discharging a link to interest #20
```

```
 # 28
 ((( plan-for <plan 2> ( at-office briefcase) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil) & ( plan-for <plan 3> (
            at-home paycheck) ( (( at-office briefcase) & ( at-home paycheck))) nil nil nil)) @ ( plan-for <plan 4> (( at-office
            briefcase) & ( at-home paycheck)) nil nil nil nil))
 Inferred by:
             support-link #28 from { 27 } by undermine-causal-links
 defeatees: { link 15 for node 15 }
            ==========================================
            Lowering the undefeated-degree-of-support of ( plan-for <plan 4> (( at-office briefcase) & ( at-home paycheck)) nil nil nil
            nil)
            retracts the previous answer to #<Query 1: (? plan)( plan-for plan (( at-office briefcase) & ( at-home paycheck)) nil nil nil
            nil)>
            ==========================================

Plan #4 has been retracted:
    PLAN-STEPS:
      (1) take-briefcase-to-office
          causal-links:
            0 --( at-home briefcase)--> 1
      GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)

                              # 48
                              interest: ( plan-for ^@y43 ~(( at-home briefcase) & ( in-briefcase paycheck)) nil nil nil nil)
                              For interest 7 by confrontation using node 14 with clues (24 15)
                              For interest 7 by confrontation using node 40 with clues (48 41)
                              This interest is discharged by node 39
                              # 61
                              interest: ( node-result ^@y64 ^@y65 ~( in-briefcase paycheck))
                              For interest 105 by embedded-goal-regression
                              This interest is discharged by node 37
                              # 62
                              interest: ( (^@y65 & ^@y66) => ~( in-briefcase paycheck))
                              For interest 61 by plan-node-result
                              For interest 63 by =>-neg2
                              This interest is discharged by node 7
                              # 63
                              interest: ( (^@y68 & ^@y69) => ~(( at-home briefcase) & ( in-briefcase paycheck)))
                              For interest 48 by goal-regression
                              This interest is discharged by node 33
 # 33
 ( (( in-briefcase paycheck) & ( remove-from-briefcase paycheck)) => ~(( at-home briefcase) & ( in-briefcase paycheck)))
 Inferred by:
             support-link #31 from { 7 } by =>-neg2
 This node is inferred by discharging a link to interest #63
                              # 74
                              interest: ( plan-for ^@y71 ( in-briefcase paycheck) ( ~(( at-home briefcase) & ( in-briefcase paycheck)))
                                    nil nil nil)
                              For interest 48 by goal-regression using node 33
                              This interest is discharged by node 35
 # 35
 ( plan-for <plan 9> ( in-briefcase paycheck) ( ~(( at-home briefcase) & ( in-briefcase paycheck))) nil nil nil)
 Inferred by:
             support-link #33 from { 3 } by null-plan
 This node is inferred by discharging a link to interest #74

Plan #9 has been constructed
      GOAL: ( in-briefcase paycheck)
          established by:
            0 --> ( in-briefcase paycheck)

 # 36
 ( plan-node <pn2: (remove-from-briefcase paycheck)>)
 Inferred by:
             support-link #34 from {  } by given
 This discharges interest 34
 # 37
 ( node-result <pn2: (remove-from-briefcase paycheck)> ( in-briefcase paycheck) ~( in-briefcase paycheck))
 Inferred by:
             support-link #35 from { 7 , 36 } by plan-node-result
 This node is inferred by discharging a link to interest #61
 # 39
 ( plan-for <plan 10> ~(( at-home briefcase) & ( in-briefcase paycheck)) nil nil nil nil)
 Inferred by:
             support-link #37 from { 33 , 35 } by goal-regression
 This node is inferred by discharging a link to interest #48
```

```
Plan #10 has been constructed
    PLAN-STEPS:
      (2) ( remove-from-briefcase paycheck)
          causal-links:
            0 --( in-briefcase paycheck)--> 2
      GOAL: ~(( at-home briefcase) & ( in-briefcase paycheck))
          established by:
            2 --> ~(( at-home briefcase) & ( in-briefcase paycheck))

  # 41
  ( plan-for <plan 11> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)
  Inferred by:
              support-link #39 from { 14 , 39 } by confrontation with clues { 24 , 15 }  defeaters: { 54 }
  This node is inferred by discharging a link to interest #7

Plan #11 has been constructed
    PLAN-STEPS:
      (2) ( remove-from-briefcase paycheck)
          causal-links:
            0 --( in-briefcase paycheck)--> 2
      (1) take-briefcase-to-office
          causal-links:
            2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1
            0 --( at-home briefcase)--> 1
          ordering-constraints:
            1 > 2
      GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)

                                # 77
                                interest: ((( merged-plan <plan 4> <plan 2> <plan 3> nil) & ( plan-for <plan 10> ~(( at-home briefcase) &
                                      ( in-briefcase paycheck)) nil nil nil nil)) @ ( plan-for <plan 11> (( at-office briefcase) & ( at-home
                                      paycheck)) nil nil nil nil))
                               Of interest as a defeater for support-link 39 for node 41
                                This interest is discharged by node 54
                                # 78
                                interest: ( plan-undermines-causal-links <plan 11> ( <0 --( in-briefcase paycheck)--> 2> <2 --~(( at-home
                                      briefcase) & ( in-briefcase paycheck))--> 1> <0 --( at-home briefcase)--> 1> <1 --( at-office
                                      briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
                                For interest 77 by undermine-causal-links
                                This interest is discharged by node 53
            ========================================
            Justified belief in ( plan-for <plan 11> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)
            with undefeated-degree-of-support 0.99
            answers #<Query 1: (? plan)( plan-for plan (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)>
            ========================================

Plan #11 has been adopted
    PLAN-STEPS:
      (2) ( remove-from-briefcase paycheck)
          causal-links:
            0 --( in-briefcase paycheck)--> 2
      (1) take-briefcase-to-office
          causal-links:
            2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1
            0 --( at-home briefcase)--> 1
          ordering-constraints:
            1 > 2
      GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)

                                # 80
                                interest: ( plan-undermines-causal-links <plan 11> ( <2 --~(( at-home briefcase) & ( in-briefcase
                                      paycheck))--> 1> <0 --( at-home briefcase)--> 1> <1 --( at-office briefcase)--> -1> <0 --( at-home
                                      paycheck)--> -1>))
                               For interest 78 by plan-undermines-another-causal-link
                                This interest is discharged by node 52
                                # 83
                                interest: ( plan-undermines-causal-links <plan 11> ( <0 --( at-home briefcase)--> 1> <1 --( at-office
                                      briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
                                For interest 80 by plan-undermines-another-causal-link
                                This interest is discharged by node 51
                                # 86
                                interest: ( plan-undermines-causal-links <plan 11> ( <1 --( at-office briefcase)--> -1> <0 --( at-home
                                      paycheck)--> -1>))
```

```
                    For interest 83 by plan-undermines-another-causal-link
                    This interest is discharged by node 50
                    # 88
                    interest: ( plan-undermines-causal-links <plan 11> ( <0 --( at-home paycheck)--> -1>))
                    For interest 86 by plan-undermines-another-causal-link
                    This interest is discharged by node 49
                    # 90
                    interest: ( plan-undermines-causal-link <plan 11> ^@y90 ^@y89 <0 --( at-home paycheck)--> -1>)
                    For interest 88 by plan-undermines-first-causal-link
                    This interest is discharged by node 48
                    # 91
                    interest: ( embellished-plan-for ^@y91 <plan 11> ~( at-home paycheck) nil *finish* ( (<pn2: (remove-
                        from-briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> .
                        *finish*)) nil)
                    For interest 90 by plan-undermines-causal-link
                    This interest is discharged by node 47
                    # 92
                    interest: ( embellished-plan-for ^@y94 <plan 11> (( at-home briefcase) & ( in-briefcase paycheck)) nil
                        <pn1: take-briefcase-to-office> ( (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-
                        to-office>) (<pn1: take-briefcase-to-office> . *finish*)) nil)
                    For interest 91 by embedded-goal-regression using node 17
                    This interest is discharged by node 45
                    # 93
                    interest: ( embellished-plan-for ^@y97 <plan 11> ( at-home briefcase) nil <pn1: take-briefcase-to-office>
                        ( (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-
                        briefcase-to-office> . *finish*)) nil)
                    For interest 92 by split-embedded-conjunctive-goal
                    This interest is discharged by node 42
 # 42
 ( embellished-plan-for <plan 12> <plan 11> ( at-home briefcase) nil <pn1: take-briefcase-to-office> ( (<pn2: (remove-from-
       briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*)) nil)
 Inferred by:
        support-link #40 from { 1 } by embedded-null-plan
 This node is inferred by discharging a link to interest #93

Plan #12 has been constructed
   PLAN-STEPS:
     GOAL: ( at-home briefcase)
        established by:
          0 --> ( at-home briefcase)
     (2) ( remove-from-briefcase paycheck)
        ordering-constraints:
          2 > *finish*
     (1) take-briefcase-to-office
        ordering-constraints:
          1 > 2

                    # 94
                    interest: ( embellished-plan-for ^@y100 <plan 11> ( in-briefcase paycheck) nil <pn1: take-briefcase-to-
                        office> ( (*finish* . <pn2: (remove-from-briefcase paycheck)>) (<pn2: (remove-from-briefcase
                        paycheck)> . <pn1: take-briefcase-to-office>)) nil)
                    For interest 92 by split-embedded-conjunctive-goal using node 42
                    This interest is discharged by node 43
 # 43
 ( embellished-plan-for <plan 13> <plan 11> ( in-briefcase paycheck) nil <pn1: take-briefcase-to-office> ( (*finish* . <pn2:
       (remove-from-briefcase paycheck)>) (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>)) nil)
 Inferred by:
        support-link #41 from { 3 } by embedded-null-plan
 This node is inferred by discharging a link to interest #94

Plan #13 has been constructed
   PLAN-STEPS:
     GOAL: ( in-briefcase paycheck)
        established by:
          0 --> ( in-briefcase paycheck)
     (2) ( remove-from-briefcase paycheck)
        ordering-constraints:
          2 > *finish*
     (1) take-briefcase-to-office
        ordering-constraints:
          1 > 2

 # 45
 ( embellished-plan-for <plan 14> <plan 11> (( at-home briefcase) & ( in-briefcase paycheck)) nil <pn1: take-briefcase-to-office>
       ( (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*))
       nil)
 Inferred by:
        support-link #43 from { 42 , 43 } by split-embedded-conjunctive-goal
 This node is inferred by discharging a link to interest #92
```

```
Plan #14 has been constructed
    PLAN-STEPS:
      GOAL: (( at-home briefcase) & ( in-briefcase paycheck))
          established by:
            0 --> ( at-home briefcase)
            0 --> ( in-briefcase paycheck)
      (2) ( remove-from-briefcase paycheck)
          ordering-constraints:
            2 > *finish*
      (1) take-briefcase-to-office
          ordering-constraints:
            1 > 2

  # 47
  ( embellished-plan-for <plan 15> <plan 11> ~( at-home paycheck) nil *finish* ( (<pn2: (remove-from-briefcase paycheck)> .
            <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*)) nil)                  DEFEATED
  Inferred by:
             support-link #45 from { 17 , 45 } by embedded-goal-regression  defeaters: { 61 }   DEFEATED
  This node is inferred by discharging a link to interest #91

Plan #15 has been constructed
    PLAN-STEPS:
      (2) ( remove-from-briefcase paycheck)
      (1) take-briefcase-to-office
          causal-links:
            0 --( at-home briefcase)--> 1
            0 --( in-briefcase paycheck)--> 1
          ordering-constraints:
            1 > 2
      GOAL: ~( at-home paycheck)
          established by:
            1 --> ~( at-home paycheck)

                                        # 95
                                        interest: ((( node-result <pn1: take-briefcase-to-office> (( at-home briefcase) & ( in-briefcase paycheck))
                                              ~( at-home paycheck)) & ( embellished-plan-for <plan 14> <plan 11> (( at-home briefcase) & (
                                              in-briefcase paycheck)) nil <pn1: take-briefcase-to-office> ( (<pn2: (remove-from-briefcase
                                              paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*)) nil)) @ (
                                              embellished-plan-for <plan 15> <plan 11> ~( at-home paycheck) nil *finish* ( (<pn2: (remove-from-
                                              briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*))
                                              nil))
                                       Of interest as a defeater for support-link 45 for node 47
                                        This interest is discharged by node 61
                                        # 96
                                        interest: ( plan-undermines-causal-links <plan 15> ( <0 --( in-briefcase paycheck)--> 1> <0 --( at-home
                                              briefcase)--> 1> <1 --~( at-home paycheck)--> -1>))
                                        For interest 95 by undermine-embedded-causal-links
                                        This interest is discharged by node 60
  # 48
  ( plan-undermines-causal-link <plan 11> (( at-home briefcase) & ( in-briefcase paycheck)) <pn1: take-briefcase-to-office> <0 --(
            at-home paycheck)--> -1>)                  DEFEATED
  Inferred by:
             support-link #46 from { 47 } by plan-undermines-causal-link   DEFEATED
  This node is inferred by discharging a link to interest #90
  # 49
  ( plan-undermines-causal-links <plan 11> ( <0 --( at-home paycheck)--> -1>))                  DEFEATED
  Inferred by:
             support-link #47 from { 48 } by plan-undermines-first-causal-link   DEFEATED
  This node is inferred by discharging a link to interest #88
  # 50
  ( plan-undermines-causal-links <plan 11> ( <1 --( at-office briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
                     DEFEATED
  Inferred by:
             support-link #48 from { 49 } by plan-undermines-another-causal-link   DEFEATED
  This node is inferred by discharging a link to interest #86
  # 51
  ( plan-undermines-causal-links <plan 11> ( <0 --( at-home briefcase)--> 1> <1 --( at-office briefcase)--> -1> <0 --( at-home
            paycheck)--> -1>))                  DEFEATED
  Inferred by:
             support-link #49 from { 50 } by plan-undermines-another-causal-link   DEFEATED
  This node is inferred by discharging a link to interest #83
  # 52
  ( plan-undermines-causal-links <plan 11> ( <2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1> <0 --( at-home
            briefcase)--> 1> <1 --( at-office briefcase)--> -1> <0 --( at-home paycheck)--> -1>))                  DEFEATED
  Inferred by:
             support-link #50 from { 51 } by plan-undermines-another-causal-link   DEFEATED
  This node is inferred by discharging a link to interest #80
  # 53
```

```
( plan-undermines-causal-links <plan 11> ( <0 --( in-briefcase paycheck)--> 2> <2 --~(( at-home briefcase) & ( in-briefcase
          paycheck))--> 1> <0 --( at-home briefcase)--> 1> <1 --( at-office briefcase)--> -1> <0 --( at-home paycheck)--> -1>))
                     DEFEATED
Inferred by:
           support-link #51 from { 52 } by plan-undermines-another-causal-link   DEFEATED
This node is inferred by discharging a link to interest #78
# 54
((( merged-plan <plan 4> <plan 2> <plan 3> nil) & ( plan-for <plan 10> ~(( at-home briefcase) & ( in-briefcase paycheck)) nil nil
          nil nil)) @ ( plan-for <plan 11> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil))                DEFEATED
Inferred by:
           support-link #52 from { 53 } by undermine-causal-links   DEFEATED
defeatees: { link 39 for node 41 }
          ==========================================
          Lowering the undefeated-degree-of-support of ( plan-for <plan 11> (( at-office briefcase) & ( at-home paycheck)) nil nil
          nil nil)
          retracts the previous answer to #<Query 1: (? plan)( plan-for plan (( at-office briefcase) & ( at-home paycheck)) nil nil nil
          nil)>
          ==========================================

Plan #11 has been retracted:
   PLAN-STEPS:
     (2) ( remove-from-briefcase paycheck)
         causal-links:
           0 --( in-briefcase paycheck)--> 2
     (1) take-briefcase-to-office
         causal-links:
           2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1
           0 --( at-home briefcase)--> 1
         ordering-constraints:
           1 > 2
     GOAL: (( at-office briefcase) & ( at-home paycheck))
         established by:
           1 --> ( at-office briefcase)
           0 --> ( at-home paycheck)

                              # 99
                              interest: ( plan-undermines-causal-link <plan 15> ^@y105 ^@y104 <0 --( in-briefcase paycheck)--> 1>)
                              For interest 96 by plan-undermines-first-causal-link
                              This interest is discharged by node 59
                              # 105
                              interest: ( embellished-plan-for ^@y116 <plan 15> ~( in-briefcase paycheck) nil <pn1: take-briefcase-to-
                                    office> ( (<pn1: take-briefcase-to-office> . *finish*) (<pn2: (remove-from-briefcase paycheck)> .
                                    <pn1: take-briefcase-to-office>)) nil)
                              For interest 99 by plan-undermines-causal-link
                              This interest is discharged by node 58
                              # 106
                              interest: ( embellished-plan-for ^@y119 <plan 15> ( in-briefcase paycheck) nil <pn2: (remove-from-
                                    briefcase paycheck)> ( (<pn1: take-briefcase-to-office> . *finish*) (<pn2: (remove-from-briefcase
                                    paycheck)> . <pn1: take-briefcase-to-office>)) nil)
                              For interest 105 by embedded-goal-regression using node 37
                              This interest is discharged by node 56
# 56
( embellished-plan-for <plan 13> <plan 15> ( in-briefcase paycheck) nil <pn2: (remove-from-briefcase paycheck)> ( (<pn1:
          take-briefcase-to-office> . *finish*) (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>)) nil)
Inferred by:
           support-link #54 from { 3 } by embedded-null-plan
This node is inferred by discharging a link to interest #106

Plan #13 has been constructed
   PLAN-STEPS:
     GOAL: ( in-briefcase paycheck)
         established by:
           0 --> ( in-briefcase paycheck)
     (2) ( remove-from-briefcase paycheck)
         ordering-constraints:
           2 > *finish*
     (1) take-briefcase-to-office
         ordering-constraints:
           1 > 2

# 58
( embellished-plan-for <plan 16> <plan 15> ~( in-briefcase paycheck) nil <pn1: take-briefcase-to-office> ( (<pn1: take-briefcase-
          to-office> . *finish*) (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>)) nil)
Inferred by:
           support-link #56 from { 37 , 56 } by embedded-goal-regression
This node is inferred by discharging a link to interest #105

Plan #16 has been constructed
   PLAN-STEPS:
```

```
      (2) ( remove-from-briefcase paycheck)
          causal-links:
            0 --( in-briefcase paycheck)--> 2
      GOAL: ~( in-briefcase paycheck)
          established by:
            2 --> ~( in-briefcase paycheck)
      (1) take-briefcase-to-office
          ordering-constraints:
            1 > *finish*

 # 59
 ( plan-undermines-causal-link <plan 15> ( in-briefcase paycheck) <pn2: (remove-from-briefcase paycheck)> <0 --( in-briefcase
          paycheck)--> 1>)
 Inferred by:
           support-link #57 from { 58 } by plan-undermines-causal-link
 This node is inferred by discharging a link to interest #99
 # 60
 ( plan-undermines-causal-links <plan 15> ( <0 --( in-briefcase paycheck)--> 1> <0 --( at-home briefcase)--> 1> <1 --~( at-home
          paycheck)--> -1>))
 Inferred by:
           support-link #58 from { 59 } by plan-undermines-first-causal-link
 This node is inferred by discharging a link to interest #96
 # 61
 ((( node-result <pn1: take-briefcase-to-office> (( at-home briefcase) & ( in-briefcase paycheck)) ~( at-home paycheck)) & (
          embellished-plan-for <plan 14> <plan 11> (( at-home briefcase) & ( in-briefcase paycheck)) nil <pn1: take-briefcase-to-
          office> ( (<pn2: (remove-from-briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> .
          *finish*)) nil)) @ ( embellished-plan-for <plan 15> <plan 11> ~( at-home paycheck) nil *finish* ( (<pn2: (remove-from-
          briefcase paycheck)> . <pn1: take-briefcase-to-office>) (<pn1: take-briefcase-to-office> . *finish*)) nil))
 Inferred by:
           support-link #59 from { 60 } by undermine-embedded-causal-links
 defeatees: { link 45 for node 47 }
 This node is inferred by discharging a link to interest #95
          ========================================
          Justified belief in ( plan-for <plan 11> (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)
          with undefeated-degree-of-support 0.99
          answers #<Query 1: (? plan)( plan-for plan (( at-office briefcase) & ( at-home paycheck)) nil nil nil nil)>
          ========================================

Plan #11 has been adopted
   PLAN-STEPS:
      (2) ( remove-from-briefcase paycheck)
          causal-links:
            0 --( in-briefcase paycheck)--> 2
      (1) take-briefcase-to-office
          causal-links:
            2 --~(( at-home briefcase) & ( in-briefcase paycheck))--> 1
            0 --( at-home briefcase)--> 1
          ordering-constraints:
            1 > 2
      GOAL: (( at-office briefcase) & ( at-home paycheck))
          established by:
            1 --> ( at-office briefcase)
            0 --> ( at-home paycheck)
```

CONFRONTATION constructs *plan&* by merging *plan1* and *plan2*, so it is defeasible in the same way SPLIT-CONJUNCTIVE-GOAL is defeasible:

```
(def-backwards-undercutter UNDERMINE-CAUSAL-LINKS
   :defeatee  split-conjunctive-goal add-ordering-constraints add-embedded-ordering-constraints
                  confrontation reuse-nodes-1 reuse-nodes-2
   :backwards-premises
   "(define links (causal-links plan&))"
   "(plan-undermines-causal-links plan& links)"
   :variables plan& links)
```

Embellishments of plans can also be defeated by undermining, and can sometimes be repaired by confrontation. Thus we also need the following analogue of the preceding reason-schemas:

```
(def-backwards-reason EMBEDDED-CONFRONTATION
   :conclusions "(embellished-plan-for plan plan+ goal node1 node2 before not-between)"
   :condition  (interest-variable plan)
```

```
:forwards-premises
   "(plan-undermines-causal-link plan+ R node link)"
   (:clue? t)
   "(extended-plan plan+ subplan node)"
   (:clue? t)
   "(node-result node precondition goal)"
   "(embellished-plan-for subplan plan+ precondition nil new-node new-before new-not-between)"
:backwards-premises
   "(define -R (neg R))"
   "(define node1* (causal-link-root link))"
   "(define node2* (causal-link-target link))"
   "(embellished-plan-for repair-plan plan+ -R node1* node2* new-before new-not-between)"
   "(define plan (make-confrontation-plan repair-plan subplan -R node (list link)))"
   (:condition (not (null plan)))
:defeasible? t
:variables  plan plan+ goal node1 node2 before not-between R node link subplan precondition
            new-node new-before new-not-between -R node1* node2* repair-plan)
```

EMBEDDED-CONFRONTATION is added to the list of defeatees for UNDERMINE-EMBEDDED-CAUSAL-LINKS.

9. Control Issues

OSCAR searches plan space, in much the same sense that an algorithmic planner does. The main difference is that OSCAR interleaves the plan search with epistemic reasoning. That is what dictates the defeasible structure of OSCAR's plan reasoning. The structure of the plan search is determined by the control structures that determine the course of OSCAR's reasoning. That in turn is handled by prioritizing the inference-queue. Prioritizing the inference-queue in different ways will lead to different plan searches. For example, the inference-queue could be prioritized in such a way that OSCAR performs a breadth-first or depth-first search. OSCAR's default prioritization scheme does not lead to a plan search that is so neatly describable. However, two features of that scheme are particularly important for getting OSCAR to solve planning problems, and bear mention here.

First, many of OSCAR's reason-schemas proceed by refining plans in the face of defeaters. In general, the search for defeaters is given a lower priority in OSCAR than most other reasoning. But this can create difficulties for plan search. Consider the Sussman anomoly.[12] This is the blocks-world problem diagrammed in figure three.

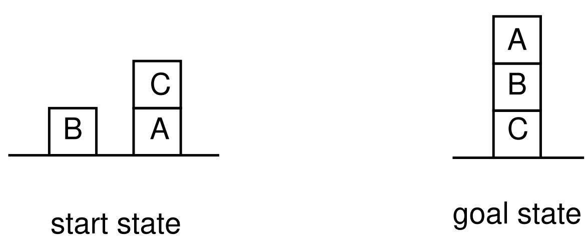

Figure 3. The Sussman Anomoly

The solution to the Sussman anomoly is the following simple plan:

[12] Sussman [1975].

```
Plan #18 has been adopted
   PLAN-STEPS:
     (1) ( move-to-table C)
        causal-links:
         0 --(( on C A) & ( clear C))--> 1
     (3) ( move B C)
        causal-links:
         0 --(( clear B) & ( clear C))--> 3
        ordering-constraints:
         3 > 1
     (2) ( move A B)
        causal-links:
         0 --( clear B)--> 2
         1 --( clear A)--> 2
        ordering-constraints:
         2 > 3
     GOAL: (( on A B) & ( on B C))
        established by:
         2 --> ( on A B)
         3 --> ( on B C)
```

This plan is found by first constructing:

```
Plan #10 has been constructed
   PLAN-STEPS:
     (1) ( move-to-table C)
        causal-links:
         0 --(( on C A) & ( clear C))--> 1
     (2) ( move A B)
        causal-links:
         0 --( clear B)--> 2
         1 --( clear A)--> 2
        ordering-constraints:
         2 > 1
     (3) ( move B C)
        causal-links:
         0 --(( clear B) & ( clear C))--> 3
     GOAL: (( on A B) & ( on B C))
        established by:
         2 --> ( on A B)
         3 --> ( on B C)
```

then adding an ordering constraint in response to undermining:

```
Plan #18 has been adopted
   PLAN-STEPS:
     (1) ( move-to-table C)
        causal-links:
         0 --(( on C A) & ( clear C))--> 1
     (3) ( move B C)
        causal-links:
         0 --(( clear B) & ( clear C))--> 3
        ordering-constraints:
         3 > 1
     (2) ( move A B)
        causal-links:
         0 --( clear B)--> 2
         1 --( clear A)--> 2
        ordering-constraints:
```

```
   2 > 3
GOAL: (( on A B) & ( on B C))
   established by:
    2 --> ( on A B)
    3 --> ( on B C)
```

and then adding a second ordering constraint in response to more undermining.

The difficulty is that in the blocks world there are always infinitely many plans for achieving a goal. E.g., first move B onto C, then move B back to the table, and then execute plan 18. If the search for defeaters always receives lower priority than the search for additional plans, OSCAR will begin by constructing plan 10, then construct systematically more complicated variants of plan 10, without ever finding the defeaters that lead to the construction of plans 12 and 18. This problem is avoided in OSCAR by lowering the priority of ultimate-epistemic-interests once an undefeated answer is found. The priority is made less than the priority of the search for defeaters, so that OSCAR will find the defeaters and then find plans 12 and 18 before finding the more complex variants of plan 10. This seems to be the same way human beings avoid this problem.

The second feature of OSCAR's prioritization scheme that is important for planning concerns collective undercutting defeat. The general OSCAR architecture assumes that we cannot stop reasoning from defeated conclusions, because there is always the possibility of collective defeat that can only be discovered by continuing such reasoning. Collective rebutting defeat is common, but collective undercutting defeat is very rare. The only cases of collective undercutting defeat that I have found involve testimony—e.g., Jones says that Smith is unreliable and Smith says that Jones is unreliable. If we are working in a domain like planning in which collective undercutting defeat does not occur, we can gain efficiency by lowering the priority of reasoning from nodes defeated by undercutting defeat.

This is particularly important in planning, because otherwise we end up pursuing many redundant paths in the construction of plans. This happens if a plan undermines more than one of its own causal-links. In that case, it will repair each one separately, then discover that each of the resulting plans still undermines some of its own causal-links, and so repair each remaining undermining separately in each of the plans, and so on. If there are N undermined-links, this will produce N! partially repairs plans. Furthermore, if one of these plans (or a plan generated from one of them by goal-reduction) is used in a subsequent application of SPLIT-CONJUNCTIVE-GOALS, they will all be used. Then all the undermining repairs must be repeated all over again for the resulting merged plan. This problem would not occur if we did not continue to reason from nodes defeated by undercutting. When the first undermining is found, no attempt will be made to fix the other underminings in the first plan. Instead, the second undermining will only be repaired in the first partially-repaired plan. And so on. The result is that only N plans are produced, and only the last one will be used in subsequent applications of SPLIT-CONJUNCTIVE-GOALS.

We do not want to block such reasoning altogether, because collective undercutting defeat is logically possible, even if it is rare. The strategy adopted is to lower the priority of reasoning from an inference-node defeated by undercutting defeat, and also lower the priority of the search for additional undercutting defeaters for that node. If the node is subsequently reinstated, the priorities are returned to their original levels. This has the effect, for example, that when one link of a plan is undermined, the priority of the search for additional undermined links is made very low, thus avoiding the above problem.

10. Conclusions

Realistic planning agents cannot assume that they have all relevant knowledge when they begin to address a planning problem. Instead, the search for a plan will typically initiate further epistemic investigation, and the course of planning and epistemic reasoning will be interleaved.

The obvious way to accomplish this is to use the same inference engine for the plan search as for the epistemic investigation. This paper has illustrated how that can be done by employing OSCAR's defeasible reasoner as the inference engine. Epistemic reasoning is obviously defeasible, and one of the central claims of this paper has been that planning for conjunctive goals must also be done defeasibly. This is because, without the assumption that the planner has all relevant knowledge before it begins the plan search, it is impossible to *compute* whether merging two plans introduces threats for their causal-links. Merging the plans must instead initiate an epistemic search for relevant information, and such a search need not terminate.

References

Fikes, R. E., and Nilsson, N. J.

1971 STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2** 189-208.

Gelfond, Michael, and Lifschitz, Vladimir

1993 "Representing action and change by logic programs", *Journal of Logic Programming* **17**, 301-322.

Green, C

1969 "Application of theorem-proving to problem solving". *Proceedings IJCAI-69*, 219-239.

Lin, Fangzhen, and Reiter, Raymond

1994 "How to progress a database (and why) I. Logical foundations." In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation (KR'94).* 425-436.

1995 "How to progress a database II: The STRIPS connection." *IJCAI-95*. 2001-2007.

McAllester, David, and Rosenblitt, David

1991 "Systematic nonlinear planning", *Proceedings of AAAI-91*, 634-639.

McCarthy, John

1977 "Epistemological problems in artificial intelligence". *Proceedings IJCA-77.*

Pednault, E. P. D.

1988 "Synthesizing plans that contain actions with context-dependent effects", *Computational Intelligence* **4** (4): 356-372.

Penberthy, J. Scott, and Weld, Daniel

1992 "UCPOP: a sound, complete, partial order planner for ADL". *Proceedings 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 103-114.

Pollock, John

1995 *Cognitive Carpentry*, MIT Press.

1996 "Reason in a changing world", in *Practical Reasoning*, ed. Dov M. Gabbay and Hans Jürgen Ohlbach, Springer, 495-509. This can be downloaded from http://www.u.arizona.edu/~pollock/.

1996a "Perceiving and reasoning about a changing world". Forthcoming in *Computational Intelligence.* This can be downloaded from http://www.u.arizona.edu/~pollock/.

1998 "Planning agents", to appear in *Foundations of Rational Agency*, ed. Rao and Woodbridge, Kluwer.

1998a "The logical foundations of goal-regression planning". Technical report of the OSCAR Project. This can be downloaded from http://www.u.arizona.edu/~pollock/.

Shanahan, Murray

1996 "Robotics and the common sense informatic situation", *Proceedings of the 12th European Conference on Artificial Intelligence*, John Wiley & Sons.

Shoham, Yoav

1987 *Reasoning about Change*, MIT Press.

Sussman, G. J.
1975 *A Computer Model of Skill Acquisition*. Elsevier/North Holland, Amsterdam, London, New York.
Waldinger, R.
1974 "Achieving several goals simultaneously", *Machine Intelligence* **8**.
Weld, Daniel
1994 "An introduction to least commitment planning", *AI Magazine*, **15** 27-62.