Plan Related Negation in Story Understanding

Solomon Eyal Shimony
Brown University Department of Computer Science
Providence, RI 02912, U.S.A.

Technical Report No. CS-89-25, March 1989

Plan Related Negation in Story Understanding

Solomon E. Shimony
Box 1910, Brown University
Providence, RI 02906

March 20, 1989

Abstract

This paper discusses simple uses of negation in story understanding. We show that the meaning of a negated sentence is not simply its logical form, but also indicates an "expectation" that its non-negated form would hold. We extend the representation of plans to handle the counter factual statements needed for negation, by treating plans as objects. We present a theory incorporating the extra meaning into a story-understanding system based on first order plans. The plan representation introduced here is shown to be useful for understanding stories where agents change their plans, even in stories without negative sentences.

1 Introduction

Work on negation in natural language research has concentrated mainly on the problem of scoping ambiguity [1]. In that context, dealing with negation as if there is just some negated formula to add to the database is reasonable. In story understanding, however, a negated sentence means more than just the negation of the formula produced by the non-negated version of the sentence. This extra meaning is caused by the fact that the sentence was *mentioned* in the story, and depends on the context, as shown in the following simple examples.

1. Jack wanted to go to he supermarket. His car didn't start.
2. Jack wanted to go to he supermarket. He didn't start his car, but walked instead.
3. Jack didn't spend his money on horse races last month. He bought a car yesterday.
4. Even though he was bald, he didn't look old.
5. Jack painted the ladder. When he wanted to paint the ceiling, the ladder wasn't dry.

In example 1 we see a failed plan (starting the car). We would also expect that Jack *intended* to start his car, but failed, therefore that he intended to *drive* to the supermarket. Example 2 is an example of an alternate plan. It signifies that we would expect Jack to drive, but that he did not (neither did he intend to do so). Example 3 shows a buying action that does not fail because its precondition did not fail (Jack probably had money). However, the reader might also expect Jack to have spent his money gambling (maybe he is a compulsive gambler). Example 4 conveys the presence of an expectation that if someone is bald, he will look old (something like a "default rule"). Example 5 shows the failing of a plan (paint the ladder and then paint the ceiling), which fails because of violated preconditions (need a dry ladder to paint the ceiling).

We see story understanding as the process of extracting a set of formulas (in a logic) from the given text. Intuitively, this is exactly the set of formulas "implied" by the story teller. In story understanding, we assume that every such formula is relevant to the other formulas (this notion of relevance is pursued in [6]). This relevance property is used in story understanding systems, such as WIMP [4].

Henceforth, all negated sentences are assumed to convey some degree of expectation of the non-negated version of the sentence. A special case is expectation in plans of agents, where the expected actions or sub-plans can be intended by the agent, and are thus usually failed plans, or actions that we would usually expect but that the agent did not intend. Non action sentences may also be negated, such as the one in example sentence 5, and they would fit into this framework as preconditions or results of actions.

More complicated sentences with negation, such as those that show interaction with quantifiers, disjunctions and conjunctions are not pursued here. We will also ignore sentences with nested negation, since they are either uninteresting (standard "double negatives"), or are too complicated to handle withing the scope of this paper. The latter are cases that occur when negation interacts with modalities such as belief, intention, etc. (e.g. "Jack didn't believe that his plan to go to the supermarket would not succeed").

Section 2 presents a representation for hierarchical actions and plans that treats plans as objects. The semantics of plans and actions are defined in section 3. Plan-failure related predicates are defined in section 4. Unfortunately, these predicates are uncomputable. Simplifying assumptions are proposed in section 5, which essentially eliminate all temporal constraints in actions and plans, but make the plan-failure related predicates computable. Section 6 discusses implementation issues, specifically, converting WIMP to handle plans and their refinements, in order to handle negation.

2 A Representation for Actions and Plans

Common representations for actions and plans in story understanding systems are scripts [9], or formulas using action objects and slots, as in [3]. We follow the gist of these representations, except that in our representation we treat plans as objects. The necessity for this treatment of plans is shown by the following argument.

When an agent intended to perform an action as part of a plan, but the action failed, we *could* say that the agent *intended* to fill that slot of the plan with the given action, but failed. In example 1, for instance, we could say that Jack intended to fill the going-to-supermarket step with driving, but that action failed. This notion of intending to fill a slot seems too complicated, and must use a second order logic (unless equality statments are objects of the language). As a way of circumventing this problem, we say that Jack had a plan for going to the supermarket by driving (i.e. the going step filled by driving), but the plan failed. However, the partial plan where the going step is *not* specified did not necessarily fail (e.g. he may still walk). Stating whether a plan failed requires that we be able to refer to it within the language, hence plans must be objects of the logic.

Actions and plans are represented using a temporal logic similar to that of [7], but with interval-relations as in [2]. The objects of our logic are the usual ones, to which we add time points, intervals, actions, plans, and states (where a state is syntactically a predicate calculus formula *not* referring to time, actions or plans).

An interval i is a pair of time points, (i_1, i_2). Actions are objects, which belong to natural types, and have slots [1]. There are special types of slots which appear on every action instance. These are: precondition, postcondition and plan. A precondition is a state. A postcondition is a set of "pcause" statements, where (pcause **p** **q**) means that the action causes **q** unless **p** turns false before the action ends. This notation for pcause is a simplified version of the one in [7].

A plan is a pair ($\mathcal{A}$, $\mathcal{C}$), where $\mathcal{A}$ is a (possibly empty) set of actions, and $\mathcal{C}$ is a (possibly empty) set of temporal constraints. A constraint is of the form c(i, j) where i, j are intervals or actions, and c is a temporal constraint as in [2], which are as shown in the following table:

[1] slots are essentially functions from actions to objects

BEFORE(i, j)	i is before j
MEETS(i, j)	i is just before j
OVERLAP(i, j)	i starts before j and ends before j ends
DURING(i, j)	i starts before j and ends after j ends
EQUALS(i, j)	i and j start and end at the same time
STARTS(i, j)	i and j start at the same time, i ends before j ends
FINISHES(i, j)	i starts after j starts, they end at the same time
G-OVERLAP(i, j)	There exists some interval contained by both i and j

We refer to the set of actions of a plan P by actions(P), and to its set of constraints by constr(P). The special action natural type ALWAYS is taken to be the most general action. It is used for completely unspecified plans. The most unspecified plan is ({ ALWAYS }, ϕ), and is called P_{any}. The fact that actions in the plan are not named, i.e. are not slots, allows us to have plans so unspecified that even the number of actions in them is not specified. This feature is useful, but we will see that it complicates matters so much that we need to re-install some order here.

The set of natural types, $\mathcal{T}$, is partially ordered under the "isa" partial function. We say that $isa(A) = A'$ iff A and A' are natural types and A' is a generalization of A. If $isa(A) = A'$, then A inherits all the slots of A' and the restrictions on them.

Other actions are "tied" to this hierarchy by statements of the form $\text{inst}(a) = A$, where A is a natural type and a is an action, meaning that a is an instance of A. If $\text{inst}(a) = A$, then a inherits all the slots of A and the restrictions on them.

For instance, the following story can be represented as shown in the following paragraphs.

> Jack wanted to go to the supermarket. He drove there.

The natural types of actions needed are: supermarket-shopping, going, and driving. We also have a number of action instances: sms1, sms2, drive1, and the following relations and slot values:

(is-a driving going)	
(is-a going move)	actions(Plan(going)) = { ALWAYS }
(inst sms1 supermarket-shopping)	actions(Plan(sms1)) = { drive1 }
(inst sms2 supermarket-shopping)	actions(Plan(sms2)) = { going }
(inst drive1 driving)	

The most detailed plan we know about is that for sms1, as it assumes going to the supermarket by driving. The plan for sms2 is less specific, because its going action, going, is less specific than drive1 (which is an instance of an action which is-a going).

Action instances are equal iff they are instances of the same natural type and all their slot values are equal. Thus, actions are only equal if they are *exactly* the same action. [2]

Formally, action instance a_1 is equal to action instance a_2 iff both following conditions hold:

1. $\exists A.\ A \in \mathcal{T} \wedge \text{inst}(a_1) = A \wedge \text{inst}(a_2) = A$

[2] Note that the definitions for plan and action equality are mutually recursive because actions have plans, but they are also parts of higher level plans. The same mutual recursion appears in the definition of action and plan refinement.

2. $\forall s.$ [3] $(s(a_1) = v \Leftrightarrow s(a_2) = v)$

Plans are equal if their actions are equal and their constraints are logically equivalent (intuitively, if they are exactly the same plan).

1. actions($P1$) = actions($P2$)
2. constr($P1$) $\leftrightarrow$ constr($P2$)

A plan P1 is a Plan-part of plan P2 either if it is a plan for achieving an action in P2, or if it has a subset of the actions and constraints of P2. For example, {drive1, ϕ} is a Plan-part of Plan(sms1) above. Formally, plan P1 is a Plan-part of plan P2 iff at least one of the following conditions hold:

1. $\exists a.\ a \in$ actions($P2$) $\wedge$ $P1 = plan(a)$
2. All the following conditions hold:
   (a) actions($P1$) $\subseteq$ actions($P2$)
   (b) $\forall a1, a2, c.\ (a1 \in$ actions($P1$) $\wedge\ a2 \in$ actions($P1$) $\wedge c(a1, a2) \in$ constr($P1$)) $\Rightarrow c(a1, a2) \in$ constr($P2$)
   (c) $\forall a, i, c.\ (a \in$ actions($P1$) $\wedge\ c(a, i) \in$ constr($P1$)) $\Rightarrow c(a, i) \in$ constr($P2$)

A plan P1 is a proper Plan-part of P2 if it is a Plan-part of P2 and $P1 \neq P2$.

The notions of action and plan refinement are inspired by those of [8]. An action a1 is a refinement of action a2 (written $a1 \uparrow a2$) if a2 is a natural type and a1 is an instance of it, or if all slots of a1 are at least as specific as those of a2.

In our example, sms1 is a refinement of supermarket-shopping because inst(sms1) = supermarket-shopping, and sms1 is a refinement of sms2 because the plan for sms1 is more specific.

Formally, a1 is a refinement of a2 (written $a1 \uparrow a2$) iff at least of the following conditions hold:

1. $a1 = a2$
2. inst($a1$) = $a2$
3. isa*($a1$, $a2$) (isa* is the transitive closure of isa)
4. If $a1$ and $a2$ are instances, then all the following conditions hold:
   (a) $\forall s, x1, x2.\ s(a1) = x1\ \wedge\ s(a2) = x2\ \Rightarrow$ x1 is at least as specific as x2 (defined ahead).
   (b) Plan($a1$) $\uparrow$ Plan($a2$) (defined in the next paragraph)
5. $\exists a.\ a1 \uparrow a\ \wedge\ a \uparrow\ a2$

[3] "s" here (and hence) ranges over slot names. Variables that start with "a" range over actions, variables that start with "c" range over temporal constraints, and variables named "i", "j", or "k" range over intervals

We assume that all objects are arranged in a hierarchy of natural types of objects $\mathcal{OT}$, similar to the action hierarchy. We also have a set of instances of objects, $\mathcal{OI}$ A slot value s1 is a generalization of slot value s2 (or "s2 is at least as specific as s1") iff at least one of the following conditions hold:

1. $s1 \in \mathcal{OT} \ \wedge \ s2 \in \mathcal{OT} \ \wedge \ \text{isa}^*(s2, s1)$
2. $s1 \in \mathcal{OI} \ \wedge \ s2 \in \mathcal{OI} \ \wedge \ \exists s.\ s \in \mathcal{OT} \ \wedge \ inst(s2) = s \ \wedge \ \text{isa}^*(s, s1)$
3. $s1 \in \mathcal{OI} \ \wedge \ s2 \in \mathcal{OI}$ and all slots of s1 are generalizations of the slots of s2.
4. $s1 = s2$

A plan P1 is a refinement of plan P2 (written $P1 \uparrow P2$) if the actions of P1 are at least as specific than those of P2; the same should hold for the constraints. In our example, Plan(sms1) is a refinement of Plan(sms2). Formally, plan P1 is a refinement of plan P2 (written $P1 \uparrow P2$) iff at least one of the following conditions hold:

1. $\text{P2} = \text{P}_{any}$
2. There exists a bijective function M : actions($P1$) $\Rightarrow$ actions($P2$) such that all the following conditions hold:
   (a) $\forall a.\ a \in \text{actions}(P1) \Rightarrow (a \uparrow M(a) \ \wedge \ M(a) \in \text{actions}(P2))$
   (b) $\text{constr}(P1) \vdash R(\text{constr}(P2))$
3. $\exists P.\ P1 \uparrow P \ \wedge \ P \uparrow P2$

where R is a function on sets of constraints, defined as follows:

$$R(C) = \{c(M^{-1}(a_1), M^{-1}(a_2)) \mid c(a_1, a_2) \in C\}$$

This mapping function between the action sets is awkward. We can get rid of it (as we will, in fact, later on) by assigning sub-slots to actions in the plan, where each sub-slot designates an action that has to do something specific (i.e. achieve some part of the postconditions). This will simplify the definition of refinement, but weaken the representation in that we cannot have additional actions in a refinement of a given plan (at least not at the top level of the plan). Hence, we will treat the set of actions in a plan as an n-tuple of named slots, where M maps the "contents" of a slot in the first plan to the context of the respective slot in the second plan.

A plan P1 is a *strict* refinement of plan P2 (written $P1 \Uparrow P2$) if $P1 \uparrow P2$ and $P1 \neq P2$.

We also use a number of fluents in our theory:

plan-of(**agent**, **plan**)	**agent** has **plan** in his plan set.
intends(**agent**, **action**)	**agent** intends **action**
occur(**action**)	**action** occurs.

All these fluents hold (or do not hold) over intervals. Thus, hold(fluent, interval) is true iff the fluent "holds" over this interval. We discuss this further in section 3, where we define the semantics.

3 Semantics of Actions and Plans

We use a temporal structure where time points are primitive. An interval is a pair of time points, as in [10].

3.1 Events and Negative Events

An event is a set of intervals, which is neither upward nor downward hereditary. Not downward hereditary means that if i is an interval in the set then any subinterval is not in the set. Not upward hereditary means that no interval containing i is in the set. A Nevent (negative event) is a a set of intervals that *are* interval upward and downward hereditary, and behaves essentially like a *fact*. Negative events are needed to represent events we know not to occur, and are usually indicated by negation in the sentence. We say negevent(**event1**, **event2**) events **event1** and **event2** are "negative" relative to ach other.

3.2 Temporal Structure

As in McDermotts paper [7], The temporal structure is forward-branching only, that is: $\forall i, j, k$ which are intervals, the following condition holds:

$$MEETS(i,k) \wedge MEETS(j,k) \Rightarrow ENDS(i,j) \vee ENDS(j,i) \vee EQUAL(i,j)$$

The reverse in time may not hold, if we reverse the positions of k in the MEETS relation, as then intervals i and j may not be comparable. Intuitively, one of the temporal constraints holds among intervals only if they are in the "same chronicle", using McDermott terminology.

3.3 Semantics in Terms of Models

The semantics is in terms of the holds predicate. holds(occurs(**a**), i) iff i is the longest interval such that all of i is in **a**.

Note that temporal constraints between intervals and actions *inside* the plans are also in terms of MEETS, BEFORE, etc. As these relation only hold between intervals in the same chronicle, we cannot talk of what happens in different chronicles in plans. This does not allow an agent to have a plan of the form "go to the supermarket if it is possible that I will have no food left".

We *can* talk about different chronicles in the meta-language, where we define what we mean by plan acceptability and applicability. Talking about agents modifying plans and preferring plans to other plans also requires reference across chronicles.

A model $\mathcal{M}$ for a set of actions $\mathcal{A}$ is a temporal structure, where the all the following conditions hold:

1. All actions in $\mathcal{A}$ occur over some intervals.
2. All temporal constraints in all plans of actions in $\mathcal{A}$ hold.

3. The model satisfies the constraints, as defined hence.

A model for $\mathcal{A}$ satisfies the constraints if all the postconditions for actions that occur hold some interval immediately after their respective actions, the preconditions for all actions that occur hold in some interval immediately before their respective actions, and there is no overlap between actions and their respective Nactions.

Formally, a model $\mathcal{M}$ satisfies the constraints iff all the following conditions hold:

1. $holds(occur(a), i) \Rightarrow \exists j.\ MEETS(i, j) \wedge holds(\bigwedge postconditions(a), j)$
2. $holds(occur(a), i) \Rightarrow \exists j.\ MEETS(j, i) \wedge holds(\bigwedge preconditions(a), j)$
3. $\neg \exists i, j, k.\ holds(occur(a), i) \wedge holds(occur(Na), j)$
   $\wedge\ DURING(k, j) \wedge DURING(k, i) \wedge negaction(a, Na)$

A model $\mathcal{M}_A$ is preferred over $\mathcal{M}_B$ if it contains a (setwise) smaller set of actions not in $\mathcal{A}$. A most preferred model $\mathcal{M}$ is minimal with respect to the above partial order. Note that a most preferred model is not necessarily unique. All definitions henceforth will be made relative to a preferred model.

3.4 Plan Acceptability and Applicability

A plan P is acceptable in an interval if its execution is possible in the interval (i.e. the preconditions of all its actions hold, and none of these actions are known *not* to occur). Formally, a plan P is acceptable in interval i if it is the empty plan, or the following conditions hold, relative to a model $\mathcal{M}$.

1. $\forall a.\ a \in \text{actions}(P) \Rightarrow \text{applicable}(\text{Plan}(a), a, i)$
2. $\exists P1.\ P1 \uparrow P \wedge \forall a.\ a \in \text{actions}(P1)\ \exists i.\ MEETS(i, a) \wedge holds(precondition(a), i)$

Note that plans for primitive actions hold vacuously, because their plans have empty action sets. This constitutes the base case for the mutual recursion.

A plan P is applicable to an action **a** in interval if it is acceptable and its actions cause all changes required by **a**. Formally, a plan P is applicable to an action a in interval i if the following conditions hold (if a is a primitive action, then the empty plan is always applicable to it):

1. acceptable(P, i)
2. $\forall st.\ st \in postconditions(a) \Rightarrow (\exists A.\ A \in \text{actions}(P) \wedge\ postconditions(A) \Rightarrow st)$
3. $\forall a1.\ \in \text{actions}(P) \Rightarrow DURING(a1, i)$

Unfortunately, both acceptability and applicability of a plan appear to be badly undecidable, in general, as computing whether the preconditions hold is equivalent to the general temporal projection problem [10].

4 Negation in Terms of Plan Refinement

We say that plan-failed(agent, P, Na), (where Na is the Naction and **a** is the respective action [4]), if there is a plan which includes action **a** that is acceptable prior to the occurrence of Na, but becomes unacceptable after Na occurs instead. Formally, we say plan-failed(agent, P, Na) iff the following condition holds:

$$\exists i, k.\ holds(\text{plan-of}(agent, P), k) \wedge$$
$$\text{acceptable}(P, i) \wedge BEFORE(i, Na) \wedge G-OVERLAP(i, k) \wedge$$
$$\neg\exists j.\ G-OVERLAP(Na, j) \wedge \text{acceptable}(P, j)$$

Typically, it is possible to recognize this in a sentence, because a negated action without an agent is an Naction which was not intended. It is possible that the inverse action *was* intended, and was a part of a failed plan. Whether the inverse action was intended depends on the kind of verb in the action (A transitive verb would tend to show intention). This evidence, however, is not conclusive and some other method should be used to help determine intention. Further discussion of such clues is outside the scope of this paper.

We say that a plan P1 was modified to plan P2 as a way of executing plan P because action **a** failed (written plan-modified(agent, P1, P2, P, Na)), if **a** is a part of P1, P1 fails because of Na, P2 does *not* fail because of Na, and P1, P2 are refinements of P). Formally, we say that plan-modified(agent, P1, P2, P, Na) iff the following condition holds:

$$\exists i, j.\ BEFORE(j, Na) \wedge BEFORE(Na, i) \wedge$$
$$holds(\text{plan-of}(agent, P), j) \wedge holds(\text{plan-of}(agent, P1), j) \wedge$$
$$\text{plan-failed}(agent, P1, Na) \wedge$$
$$\text{acceptable}(P2, i) \wedge holds(\text{plan-of}(agent, P2), i)$$

If some action **a** is in a plan P1, but its inverse (Na) occurs, and some other action is in some plan P2 of agent, where both P1 and P2 are refinements of the same plan P of agent, we say that the agent prefers P2 over P1 as a way to execute P (written plan-preferred(agent, P2, P1, P)). Na has to occur intentionally. Formally, we say plan-preferred(agent, P2, P1, P) iff following condition holds:

$$\exists i, j.\ BEFORE(j, Na) \wedge BEFORE(Na, i) \wedge$$
$$holds(\text{plan-of}(agent, P), i) \wedge holds(\text{plan-of}(agent, P), j) \wedge$$
$$holds(\text{plan-of}(agent, P2), i) \wedge \neg holds(\text{plan-of}(agent, P1), j) \wedge$$
$$\text{plan-failed}(agent, P1, Na) \wedge$$

[4] We actually mean that action **a** belongs to a *natural type* that is the inverse of that which Na belongs to. Instead, we use meaningful names to convey this for brevity and simplicity

$applicable(P2, i) \wedge$

$holds(intends(agent, Na), j)$

It is possible to recognize intended negated actions because a negated action with an agent and a transitive verb describing the action is usually an intended Naction. The inverse action, **a**, is *not* intended. Again, this evidence is not conclusive.

Failure of preconditions is treated in the following manner: A negated precondition ($\neg$Pr) means that there existed a plan, that plan is acceptable but with the negated precondition it is no longer acceptable. With a negation of a statement that negates preconditions, it means that the precondition (Pr) holds, and the plan *is* acceptable. In the first case, we say that plan-succeeds-if(P, Pr) if P is acceptable in exactly the same chronicles as Pr. In the second case, we substitute $\neg$Pr for Pr in plan-succeeds-if.

Calculating anything defined in this section requires evaluating the truth values of preconditions, acceptability and applicability, which are undecidable.

5 Simplifying Assumptions and the Resulting Theory

To make the predicates defined in the previous section computable, and enable us to implement plan-related negation in story understanding, we make some simplifying assumptions. These simplifying assumptions are made only so as to enable implementation, and are not necessary for the theory as a whole. The assumptions given here weaken our representation, but we can see that examples 1 through 4 can still be handled in a reasonable way.

Assumption 1: All plans are refinements of a (finite) given set of plans, namely plans of actions that are natural types. This assumption is that all plans are essentially "known" to the system (not necessarily explicitly), thus limiting the search space considerably.

Assumption 2: Everything is a-temporal, and all plans are represented without temporal relationships. That means that occurrence of all actions of an acceptable plan implies that the plan succeeds. The *explicit* non-occurrence of an action which is part of a plan implies failure of the plan.

Assumption 3: Actions that are parts of plans that are executed are assumed to occur. Preconditions and postconditions are ignored [5]. In our example, if sms2 occurs, then going is assumed to occur, but not necessarily drive1 (even though it occurs as well, if we ignore the negation).

Assumption 4: The action type hierarchy is finite, and the number of slots in every action is finite.

[5] Sometimes, preconditions can be translated into an action of (achieve preconditions). As we are ignoring time, actions and general propositions can be made to behave the same way. E.g. in our example we can either have driving be an action with the precondition (running car), or have an action (achieve (running car)) or even an action (start car).

Assumption 5: There are no cycles relative to isa, inst, and substeps of actions.

With these assumptions, we formulate the problem of explanation (including negation) as follows:

Input: A set of actions, tagged with "intends" for intended actions, and "occurred" for actions that occured.

World knowledge: A set of action types that have plans with no temporal constraints.

Output: A set of plans and actions, with refinement, plan-part and negation-type relations, which are the *explanations* of actions and things in the story.

Assertion 1: The size of the output is finite.

Assertion 2: The size of the output is potentially exponential in the size of the action natural type hierarchy.

A modified plan-recognition type story understanding system could solve this problem. However, there are two complications:

1. For every *mentioned* action and plan every state of binding constitutes another distinct plan or action in our theory. This is in contrast to standard slot notation, where only the current binding (or specification) of the plan or action is accessible.

   For example, a supermarket-going plan with an unspecified going-step is an object. So is the same plan where the going-step is-a driving action. These plans are *distinct* objects which can be referred to, as Plan(sms2) and Plan(sms1), respectively. Once we say that the going-step is-a driving ((go-step sms1) = drive1 and (inst drive1 driving)), we cannot refer to the unspecified version of it. This complication has nothing to do with negation, as it is caused by our notions of equality (and refinement) between actions and plans, and "generates" a multitude of actions and plans, most of them uninteresting from the story-understanding point of view.

2. For every action **a**, we have to propose its negated action Na, marking it with a tag signifying that it did not actually occur. Then we check whether *it* would appear in each possible plan, so that we can check whether plan-failed(agent, P, **a**) holds, i.e. to explain the failure of P by the occurrence of **a**. This further increases the number of actions, but seemingly only by a constant factor.

To handle our example 1, we use the representation as in section 2, except that now action drive1 is known *not* to occur (because its negative action Ndrive1, where (inst Ndrive1 not-driving), occurs). Therefore, Plan(sms1) fails and sms1 does not occur. However, if drive1 does occur, then Plan(sms1) succeeds, sms1 can occur and Plan(sms1) is acceptable. Thus, plan-failed(Jack1, Plan(sms1), Ndrive1) holds. Plan(going) does *not* fail, so going is still acceptable, as is sms2.

If, in example 1, we are also told "Jack then walked instead", we have the following (additional) objects: sms4, and walk1. These objects have the following relations between them (and the existing objects):

(inst sms4 supermarket-shopping)	actions(Plan(sms4)) = { walk1}
(inst walk1 walking)	(is-a walking going)
sms4 $\Uparrow$ sms2	sms1 $\Uparrow$ sms2

As the failed sms1 and the successful sms4 are both refinements of sms2, we can see that sms4 is explained by the failure of sms1. Thus, plan-modified(Jack1, Plan(sms1), Plan(sms4), Plan(sms2), Ndrive1) holds.

In example 2, we have exactly the same objects as in the extended example 1, except that drive1 is not intended, so that we can say that plan-preferred(Jack1, Plan(sms4), Plan(sms1), Plan(sms2)), i.e. Jack preferred to walk instead of driving to execute sms2.

To handle example 3, we have to introduce the following axiomatization of the domain. As we do not support preconditions directly, we need the following "actions": have-money, not-have-money (the Naction for have-money), spend-money, not-spend-money, and buy-car. We then need to have have-money $\in$ actions(Plan(buy-car)), and (is-a spend-money not-have-money). With all that given, we can show that the plan for buying the car succeeds because not-spend-money occurs.

Example 4 is *not* a plan domain problem. However, given that we are already ignoring time and treating preconditions as actions, we can also treat "being-bald" and "looking-old" as actions. We could then have being-bald $\in$ actions(Plan(looking-old)). The notion of Plan(looking-old) may look somewhat unnatural, but it only means that one way of looking old is being bald, unless we are told otherwise. When "being-bald" occurs, we expect "looking-old" to occur (as the "plan" for it "succeeds"), but it does not. The system could then have both the expectation of "looking-old", and the "not-looking-old" action occurring.

Example 5 would fail to work with assumption 3, since we cannot encode the (essential) temporal constraint that we have to have a dry ladder to paint the ceiling, and thus need to paint it *after* painting the ceiling. We *could* encode the fact that we need a dry ladder to paint the ceiling as an "action", ladder-dry, but if we are then told that the ladder is wet *after* painting the ceiling (say the ceiling was painted first), we would *still* incorrectly expect the plan to fail.

6 implementation

We need a slot notation story understanding system, with a capability for reasoning under assumptions, to implement the theory. WIMP (Charniak and Goldman, [4]) is already such a system, with its FRAIL rule interpreter and its underlying Assumption based Truth Maintenance System (see [5]). As seen in the last section, the potential amount of output is exponential. To limit the "uninteresting" output, we present the following assumption:

Assumption 6: We are not interested in lengthy, "unlikely" chains of substep

relationships where shorter, more likely chains exist.

WIMP uses a quasi- probabilistic cutoff, and that helps reduce the length of chains of actions generated, in accordance with assumption 6.

Plans are not objects in WIMP, and making them so is hard, as that requires making a plan instance for all possible sets of actions that can be fit into some reasonable plan hierarchy. Instead, we can refer to the action of which plan slot a given plan "fills", and get back to "intending" to fill a substep slot in an action, without actually saying something second-order like (intend agent '(== '(go-step sms1) drive1)). Instead, we can just say (intend-fill '(go-step sms1) drive1). The latter method may not allow us to deduce things that we could have otherwise, but it seems that we can still do the same interesting things.

Every action gets a subset of the following tags:

- intended(agent, action)
- occurred(action)
- expected(action)
- not-occurred(action)

Actions also "inherit" plan tags, a subset of the following:

- expect-fill(agent, substep, action, subaction)
- intend-fill(agent, substep, action, subaction)
- occur-fill(agent, substep, action, subaction)
- plan-failed(agent, action1, action2)
- plan-modified(agent, action1, action2, action3, action4)
- plan-preferred(agent, action1, action2, action3, action4)

We assume that the set of actions, their agents, and whether they occur, are intended, etc. are given. These would be generated by WIMP's parser, as they are now, except for the "intend", "occur", and similar tags.

For each new object instance, new-inst, and for each natural object (including action) type new-type that has a slot which new-inst can fill, WIMP asserts the following statements:

1. An equality statement (== new-inst old-inst) if the types agree.
2. Create a new instance of new-type, and assert that new-inst fills the relevant slot of new-type.

The assertion of equality with some old instance is the one preferred by WIMP (it gets a higher quasi-probability). Slot filling is done in wimp using equality. We, however, replace this equality by an expect-fill assertion. The expect-fill assertion is

then "strengthened" to an occur-fill if the new-inst is not a negated action (i.e. is not tagged by a not-occur). If new-inst is an intended action, we also assert an intend-fill relation between the instances.

An action instance *a* thus having two different expect-fill relations with two different slot fillings (action instances) *a1* and *a2*, in fact has plan refinement relations with them, because both *a1* and *a2* can be treated as (plan-parts) refinements for the plan of executing (part of) *a*. Additionally, if *a1* is intended but does not-occur, we can say that plan-modified(agent(*a*, *a1*, *a2*, reason), where "reason" is the not-occurring action which causes *a1* not to occur. Similarly for finding out possible plan-preferred, plan-failed, etc.

7 Conclusion

In this paper, we introduced a representation for plans that is a variant of the slot notation, in that plans are objects (they correspond to an entire sub-action tree). This notation allows us to define relations of sub-parts of plans and of plans refining plans. The latter allows us, in turn, to refer to plans which are possibilities that are known not to occur (due to negation), to explain the real plan (i.e. whether it is a modified plan because of failure, or because of preference by the agent). This theory can also be used to explain stories without negative sentences, but in which plan modifications occurred.

The notation uses temporal constraints and preconditions, both of which cause the problem of determining which actions belong to which possible plans undecidable. By ignoring time and making other simplifying assumptions, the problem becomes manageable, and allows us to understand simple instances of plan refinement (and thus negation). However, some planning cases *need* these temporally ordered preconditions and actions, and we cannot deal with them using our simplifying assumptions. The fact that we no longer have time in the system allows us to deal with negation in non plan oriented situations.

Future research could be aimed at limiting the temporal constraints to make the problem tractable, while keeping sufficient structure to deal with example 5 and similar ones correctly.

8 Acknowledgments

Special thanks are due to my advisor, Eugene Charniak, for long discussions about the theory, and for suffering through unpolished versions of this paper. For reading and commenting on cryptic versions of this paper, I wish to thank Leora Morgenstern, Randy Callistri and Lynn Stein.

References

[1] James Allen. *Natural Language Understanding*, chapter 10, pages 288–289. Benjamin/Cummings, 1987.

[2] James F. Allen. Towards a general theory of action and time. *AI Journal*, 23, 1984.

[3] Eugene Charniak. A common representation for problem solving and language comprehension information. *Artificial Intelligence*, 1981.

[4] Eugene Charniak and Robert Goldman. A logic for semantic interpretation. *AAAI*, 1988.

[5] Johan de Kleer. An assumption based tms. *Artificial Intelligence*, 1986.

[6] H. P. Grice. Logic and conversation. In P. Cole and Morgan, editors, *Syntax and Semantics 3: Speech Acts*, pages 41–58. Academic Press, New York, 1975.

[7] D. McDermott. A temporal logic for reasoning about processes and plans. Technical Report 196, Computer Science Department, Yale University, 1981.

[8] Leora Morgenstern. Replanning. In *DARPA Knowledge-based Planning Workshop*, Austin, 1987.

[9] Roger C. Schank and Rober P. Abelson. *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum, Hillsdale, N.J., 1977.

[10] Yoav Shoham. *Reasoning about Change: Time and Causation from the Standpoint of AI.* MIT, 1988.