Automatic Generation of Warning Instructions from Plan Spaces

Yun-Gyung Cheong
Department of Computer Science
North Carolina State University
Raleigh, NC, 27606
ycheong@ncsu.edu

Automatic Generation of Warning Instructions from Plan Spaces

Yun-Gyung Cheong

Liquid Narrative Group
North Carolina State University

Abstract

Warnings are widely and commonly used in our lives. People almost unconsciously utter warnings, but the process of generating them has not been explained clearly. In this paper, we present a method of automatic warning generation from plan spaces. The type of warning dealt in this paper is confined as eliminatory warnings, which has a role of excluding other plans in the user's plan space. The main procedure of the method consists of three phases – building the user's plan space, identifying bad plans, and finding the plan features classifying bad plans from good plans in the user's plan space using C4.5 technique. Warning instruction sentences are generated from the bad plan classifying features. The method introduced in this paper can be used for a personal assistant agent which generates instructions dynamically.

1 Introduction

Warnings are seen easily and used commonly in our lives. Ayres et al (1994) have characterized a warning as "… information about a possible negative consequence – a message that something undesirable may occur to someone or something as a result of taking (or failing to take) some action." By communicating warnings, people can avoid a possible injury or damage may be caused by their actions.

Although people almost unconsciously utter warnings, the process of generating them has not been explained clearly. Over the decades, psychological researchers have examined various aspects of warnings to study the pragmatic use of warnings such as the effects of warnings on humans and necessary conditions for effective warnings. However, few papers have been published on the technical aspects of warnings, particularly generating them automatically by machine. In this paper a method of generating warning instructions, which is a kind of warnings serving as a tool of invalidating other unsuccessful actions in a user's plan space, is presented.

Warnings are different from instructions in that they have a role of preventing target users from being in undesirable or dangerous situations whereas instructions guide them to achieve a state what they pursue. When people lean how to use a new system, they show a great tendency to depend on the instructions given from the system or experienced human users. As they become more knowledgeable about the system, the importance of instructions lessens. If a user has knowledge to a certain extent, excessive detail instructions would be rather troublesome to them. In such a case, giving warnings where a user might perform bad plans would be appropriate.

From the observations of the generation of warning instructions by people we acknowledged that different types of warnings exist. We defined three types of warnings based on their functions. *Precautionary warnings* enable the user aware injury, dangerous, and harmful situations. For instance, the warning "Do not touch the toaster while it is hot" would prevent the user from getting burned. *Eliminatory warnings* let the user exclude several possible plans from her or his plan space created to achieve a specific goal. The excluded plans should not be dangerous or harmful plans. For example, the instruction "Do not take a bus when you go to school" eliminates a means of transportation (e.g., taking a bus) from a user's plan space. *Emphasizing warnings* provide the user preconditions of another action or a necessary action which is not included in the user's plan. An instance of a warning which gives a user a precondition is the instruction "Prepare a ticket or money before taking a bus." In this case, the warning provides the precondition of taking a bus that might be overlooked by the user. Reminding the definition of warnings by Ayres et al, "something undesirable may occur to someone or something as a result of failing to take some action" can be prevented by this kind of warnings.

In this paper, we describe a method to generate warning instructions automatically by finding bad plans in a user's plan space and applying a decision tree algorithm to the bad and good plans. Warnings handled in our system are confined to *eliminatory warnings* as defined beforehand. In addition to generating these warnings, the system also creates instructions following Grice's Maxim of Quantity, which means that a sender or a speaker of information should not convey more information than a receiver or a hearer needs to hear or receive.

The previous research related with this topic is reviewed in chapter 2. In chapter 3, the details of the automatic warning generation system are presented. Chapter 4 introduces an example to help illustrate the system's behavior. In the last chapter, we discuss the benefits and weakness of our system, and propose the future work.

2 Related Work

Our work is related to a number of research areas such as discourse planning (e.g., generating instructions from plans, generating warning instructions), natural language pragmatics (e.g., Grice's maxim of quantity) and decision tree techniques. In this section, we review significant projects in each of these research areas. As we will show, little research has been done on automatic warning generation whereas many researchers have investigated instruction generation.

Discourse can be defined as a sequence of sentences which implies the intention of a speaker. Among many researchers who have studied discourse planning Mann and Thompson's Rhetorical Structure Theory (RST) has played a crucial role in analyzing and generating multi-sentential utterances (Mann and Thompson, 1987). They claim that there is a preferred relationship between continuous discourse elements, and it can be defined formally. RST has been implemented successfully into practical systems such as intellectual tutor, explanation models and question-answering systems by several researchers (Moore and Paris, 1994; Hovy, 1993). However, there was some criticism on RST as well. As Moore and Pollack (Moore and Pollack, 1992) point out, RST is not enough to express speakers' intentional structure because it ignores that discourse elements are connected simultaneously on multiple levels. They argue that a new discourse model explaining informational and intentional relations should be devised. Lambert and Carberry (1991) devise a tripartite discourse model to recognize goal-oriented nature of discourse. The tripartite model is composed of three different levels, which are a domain level, a problem-solving level, and a discourse level. The domain level expresses a user's ultimate goal, for example to earn an undergraduate degree. The problem-solving level expresses a meta-goal that is pursued to construct a domain plan. And the discourse level has a communicative goal such as obtaining information or expressing surprise.

Mellish and Evans (1988) developed a complete system to generate text from plans. On receiving a plan structure, their system outputs natural language text explaining how to execute the plan. The system has four stages to manipulate their messages. The first stage is message planning, which decides the content to be included in the text. The generated message is simplified using localized rewrite rules in the second stage so that unnecessary and redundant information can be excluded. In the structure building stage, a functional description of the message is built to help the actual realization of the message. This stage considers the problem of syntactic structures, pronominalization, and lexical choices as well. The last stage produces a linear sequence of words for the input plans using the linguistic information sent from the previous stage.

The machine learning theories are broadly used in various Artificial Intelligence research areas. Di Eugenio et al (1997) apply the C4.5 machine learning algorithm to the problem of predicting cue phrases. They define three main problems: cue occurrence, cue placement, and cue selection. They show that the decision tree algorithm is effective to know whether the cue would be included in the text, and where it should be placed. Nakato and Kato (1998) apply the C4.5 algorithm to selecting cue phrase problem. Aone and Bennett (1995) use the C4.5 algorithm to anaphora resolution. Vander Linden and Di Eugenio (1996) have reported that C4.5 successfully generates the rules to determine imperative expressions for generating warning text based on three factors: intentionality, awareness, and safety.

Young (1999) explores Grice's Maxim of Quantity problem from the view of a cooperative approach to plan descriptions. He developed CPI(cooperative plan identification) architecture, which was a computational model that generated textual descriptions of plans following Grice's Maxim of Quantity. In CPI architecture, speakers build candidate partial plans by leaving unnecessary information out of source plans. The communicated partial plan is identified as a cooperative one when a hearer can reconstruct a complete plan from the given partial plan using her reasoning processes. As indicated in his experiments, it is proved that human subjects who are given with concise instructions considering her user model performs the given tasks better than those given with full instruction descriptions.

Several recent researches have reported issues on instructions generation and corpus analysis of warning. Nevertheless, research on automatic warning generation has not been much reported. A closely related to our work is that of Ansari (1995). He developed a system which generated procedural and warning instructions from environment models. His system is mainly composed of three phases. On receiving a user's goals, the system generates a basic plan to achieve her goals. Then, a plan which leads to an injury situation is generated. Several core actions in the injury sub-plan are then inserted into the user's basic plan whenever the preconditions of the plan are met. With situation calculus and several examples, he shows that the system successfully generates warnings in a specific domain. However, his system has several limitations as followings. First, how to select proper injury goals for a user's plan among multiple possible injury goals is not mentioned. Second, the question of how to pick parts of injury plan to be included in the basic plan is still remained. Third, in a situation where multiple injury sub-plans exist for an injury goal which one would be selected to be included in the user's plan is another problem. Fourth, they assume that most actions of the injury sub-plan overlap the user plan, but there are many cases where their assumption does not fit. Lastly, how to differentiate warning instructions from positive instructions is not explained. If an action can occur in a basic plan or an injury sub-plan, his system cannot decide if it should be realized as a negative imperative or a positive instruction.

There is research on the corpus analysis of warnings and effects of warnings, however, we do not mention the corpus related work due to the limitations of the space.

3 System

3.1 Overview

The focus of our research is on generating concise warnings from the plans that a user is considering to achieve her goal before one of the plans are executed. We restrict the type of warnings to *eliminatory warning*, which excludes bad plans being considered by a user. We also assume that our warning generation system correctly models the user's plan space. I believe that modeling the user's plan space can be possible if the system can observe the behaviors of a user for a certain amount of time. A mail classification agent, for instance, can construct a user model after it has kept track of what the user has read and deleted without reading. The formal model of the planner used in our system is illustrated in section 3.2. The system described as in Figure 1, is composed of five components. They are planner, bad plans identifier, encoder, bad plan classifying features constructor, and linguistic realization component.

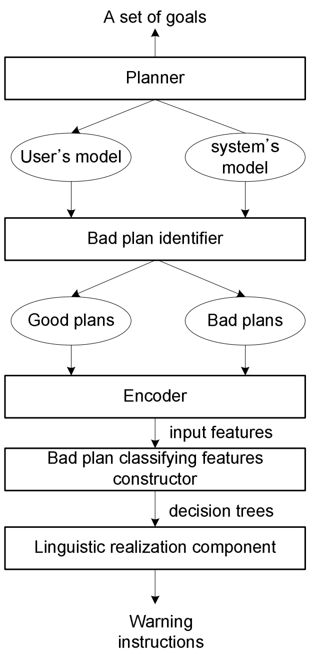


Figure 1: System overview

The planner first models a user's plans, which are represented as a plan space. The planner then generates the system's plan space, which is assumed to be sound in any situation. In the bad plan identification phase, bad plans are detected by inspecting and comparing the plan space of the system and that of the user. Bad plans are termed in this paper to refer to successful plans with the user's perspective but failed plans with the system's while good plans are successful in the plan spaces of both. The user can have bad plans due to her lack of information or her incorrect information. This is represented in our system as not having plan operator or having incorrect plan operators. On receiving the classified plans, encoder provides the bad plan classifying features constructor with training data. A training data is produced by encoding various properties of the user's plan along with its class, good plan or bad plan. Next the constructor identifies *bad plan classifying features* utilizing the C4.5 algorithm and builds a decision tree, which determines the content of the warning. Finally, linguistic realization component converts the generated decision tree to warnings as text. The system gives her the warnings so that she can rebuild her plan space excluding bad plans. The architecture of our system is similar to that of Mellish and Evans (1988) in that we generate texts from input plans. Our planner, bad plan identifier, and the constructions of bad plan classifying features correspond to their message planning stage. Message simplification unit exists in both systems. Linguistic realization component functions as their compositional structure building and linearization components. The difference is that we have more stages, identifying bad plans and building bad plan classifying features, to make the content of warnings that a user should not perform.

3.2 Planning

The planning stage starts when the user posts her goal as input to the planning process. We use Longbow Planner (Young *et al.*, 1994), a hierarchical partial-order causal link planner, as our planning component. A set of goals from the user and an initial state from the system is given in the beginning of the planning process. The planner generates all the plans which can achieve the goal state from the initial state if the plans are executed correctly. The plan structures are similar to those used in partial-order, causal link (Penberty and Weld, 1992) and HTN-style planning systems (McAllester and Rosenblitt 1991).

Let LB be a planner, L_U a user's library which is a set of operators, and PP_U a user's planning problem world: initial state and goals. On the other hand, let us define L_S a system's library and PP_S as a system's planning problem world. We assume that the system uses the same planner LB of a user. Even though we assume the same planner for both perspectives, resulting plans from each view can be different due to the difference of their planning libraries and states of the world. Let us define PB_U as a set of bad plans of users. Since we assume that a system's plans are sound, a user's plan may be wrong when there are differences between the system's plan and the user's one. Thus, what the system shall do is to generate δ, a set of warning instructions constructed from PFB_U which is a set of plan features leading the user's plans failed. We expect that a user could build a sound plan space excluding the plans having some of PFB_U by receiving δ from the system. Simply put, the system can help a user to transfer her or his planning space $< LB, L_U, PP_U >$ to ($< LB, L_U, PP_U, > - P_B$) by giving δ to users. Bad plans are recognized as successful in the planning space $< LB, L_U, PP_U >$, but not successful in the planning space ($< LB, L_U, PP_U > - P_B$) or $< LB, L_S, PP_S >$. On receiving warning instructions from the system, whenever the user refines the current partial plan node, a partial plan inconsistent with δ would be pruned from the user's plan space. If one of the steps of the plan is a component in the set δ, then the user discards the plan. The components in the set δ shall not appear in any single plan of the user's complete plans. Because the nodes inconsistent with δ are pruned during the checking phase, the planner would give more chances to the user to exploit unexpanded nodes which were not examined before due to the limited searching time. Therefore, we expect that the user may generate more successful plans than without given warnings. Since the processing capacity of the user is limited and by the Grice's maxim of quantity, we restrict that δ should be the minimal set among possible sets.

L_U	{op ∪ d_op | op is a primitive action of a user and d_op is an abstract action of a user}
L_S	{op ∪ d_op | op is a primitive action of a user and d_op is an abstract action of the system}
op	< Name, Preconds, Effects >
d_op	<Name, Ops, Orderings, CausalLinks>, where Ops has a set of op { op | op $\in$ L_U}
Name	A unique string
Preconds	Conditions of the world which need to be met before op is done
Effects	Conditions of the world after op is done
Orderings	{(S1 S2) | plan step S1 proceeds plan step S2}
Causal Links	{(S1 -> S2; e) | S1 makes condition e true, which is one of S2's preconditions}
PP_U	A planning problem of a user, $< I_U, G_U >$
PP_S	A planning problem of the system, $< I_S, G_S >$
I_U	An initial state of a user's world
G_U	A goal state of a user's world
I_S	An initial state of the system's world
G_S	A goal state of the system's world
P_U	Complete plans of the user's plan space PSG_U
P_S	Complete plans of the system's plan space PSG_S
PB_U	A set of bad plans of a user's plan space {bp | bp$\in P_U$ } and { bp | bp$\notin P_S$ }
PF_U	Plan features of p, where {p | p$\in P_U$ }
PFB_U	A set of plan features which leads a user's plans to failed plans, {pf | pf$\in PF_U$ }
δ	A set of warnings conveying PFB_U
PSG_U	A user's plan space given <LB, L_U, PP_U>, {p | p$\in P_U$ }
PSG_S	The system's plan space given <LB, L_S, PP_S>, {p | p$\in P_S$ }

Figure 2: Definition of symbols

3.3 Identification of Bad Plans

We apply the approach of filtering the successful plans in the user's plan space to finding bad plans of a user. When the user's plan space is modeled with a set of initial states and given goals, the system checks each successful plan in her plan space. If the plan is successful with the system's view: the goal state of the planning problem is achieved by the sequence of actions specified by the plan, it is identified as a good plan. Otherwise, it is classified as a bad plan.

Suppose that a user who is living in New York has a plan to go to Paris by flight with her driver's license. However, we immediately sense that her plan is faulty since international flight requires her passport. Similarly, the system can detect the user's faulty plans by determining whether the user's goal is achieved or not or by comparing plan operators in the system's library and those in the user's library. We define six different cases to determine the class of plans in Figure 3.

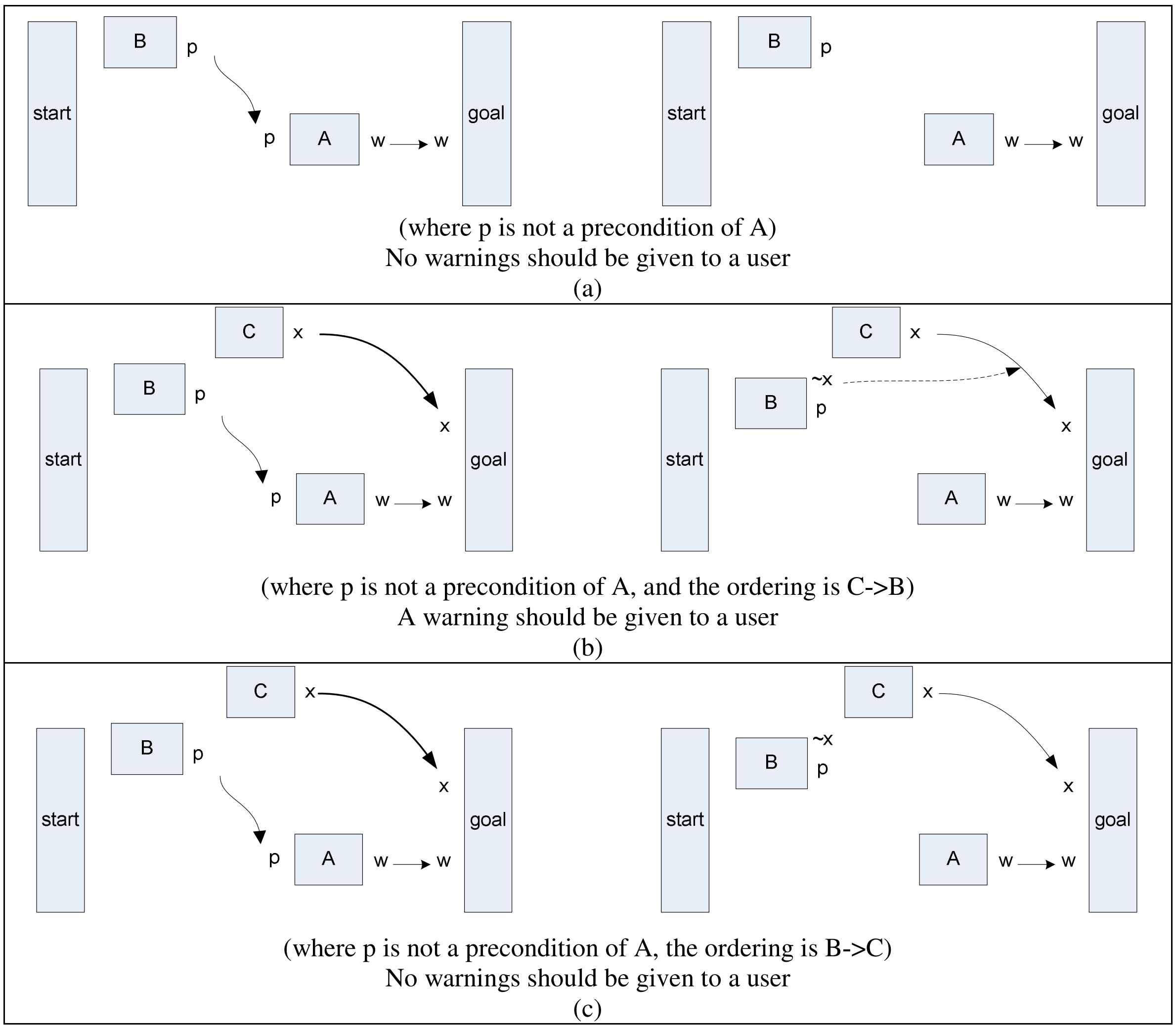
In this figure α denotes a plan operator, p represents a precondition of α, and w represents an effect of α. In case 1, for instance, the system generates a warning if a user includes a plan operator which is not in L_S (the system's plan library) since there is no way to check if the plan is successful due to the incomplete information. Other cases can be explained in the same manner as we did with case 1. Case 2 is not dealt with in our current system because the user would not have a plan including an operator which does not exist in her or his plan library. However, this issue can be considered in our future work.

1. $\alpha \in L_U$ and $\alpha \notin L_S$
// an operator is in the user's library, but not in the system's library
2. $\alpha \notin L_U$ and $\alpha \in L_S$
// an operator is not in user's library, but in the system's library
3. $\alpha \in L_U$ and $\alpha \in L_S$, $p \in$ a set of preconditions of α in L_U and $p \notin$ a set of preconditions of α in L_S
// an operator is in the user's and the system's libraries, a precondition of the operator is in the set of preconditions of α in the user's library, but not in the set of preconditions of α in the system's library
4. $\alpha \in L_U$ and $\alpha \in L_S$, $p \notin$ a set of preconditions of α in L_U and $p \in$ a set of preconditions of α in L_S
// an operator is in the user's and the system's libraries, a precondition of the operator is not in the set of preconditions of α in the user's library, but in the set of preconditions of α in the system's library
5. $\alpha \in L_U$ and $\alpha \in L_S$, $w \in$ a set of effects of α in L_U and $w \notin$ a set of effects of α in L_S
// an operator is in the user's and the system's libraries, an effect of the operator is in the set of effects of α in the user's library, but not in the set of effects of α in the system's library
6. $\alpha \in L_U$ and $\alpha \in L_S$, $w \notin$ a set of effects of α in L_U and $w \in$ a set of effects of α in L_S
// an operator is in the user's and the system's libraries, an effect of the operator is not in the set of effects of α in the user's library, but in the set of effects of α in the system's library

Figure 3: Cases where bad plans can be detected

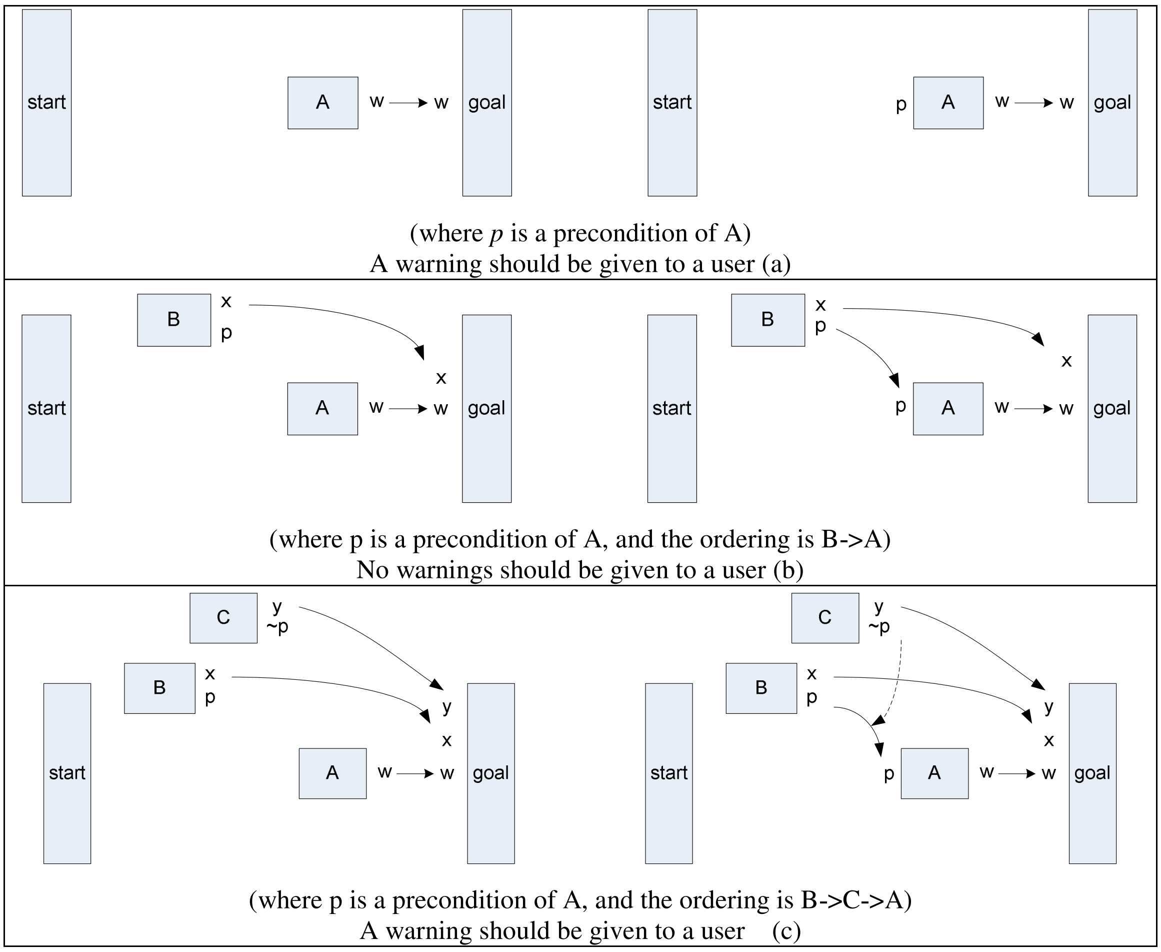
Furthermore, we suggest whether to generate warning instructions or not in a specific situation. Figure 4 illustrates three different situations of case 3 and a response for each. Each situation is presented with two diagrams. The left one in a figure describes the plan of a user, and the right one corresponds to that of the system. A slashed line in the figure denotes a threat. The correct operator corresponding to the wrong one is shown within parentheses below each diagram.

Figure 4 shows a situation where a user thinks *p* as a precondition of A, but *p* is not a precondition of A in the system's operator library. It is observed that the system does not identify the plan in Figure 4(a) as a bad plan even though the user has an incorrect operator because there the user's goal is successfully achieved in the system's perspective. In Figure 4(b), conversely, the system generates warnings since the addition of plan operator B after operator C results in threatening the other goal of the user, which is *x*. In the meantime, Figure 4(c) shows a situation where the system does not give the user warnings since the inclusion of the operator B before the operator C does not invalidate the goal of the user, *x*.


start
B
p
p
A
w
w
goal
start
B
p
A
w
w
goal
(where p is not a precondition of A)
No warnings should be given to a user
(a)
C
x
x
start
B
p
p
A
w
w
goal
C
x
~x
B
p
x
start
A
w
w
goal
(where p is not a precondition of A, and the ordering is C->B)
A warning should be given to a user
(b)
C
x
x
start
B
p
p
A
w
w
goal
C
x
~x
B
p
x
start
A
w
w
goal
(where p is not a precondition of A, the ordering is B->C)
No warnings should be given to a user
(c)
Figure 4: A scenario of Case 3 ($\alpha\in L_U$ and $\alpha\in L_{S,}$ $p\in L_U$ and $p\notin L_S$)

Concerning case 3, we raise an issue on how to generate effective and efficient warnings. We observe that a user may have inefficient or plans due to unnecessary preconditions of the operators that she has. As an example, when a user tries to travel from San Francisco to New York City and does not have her passport, she may think that she needs her passport to take the flight and plans a sequence of actions to get her passport. These actions would be redundant since she has her driver's license, and it enables her to take the flight. In this case, the system can generate a warning focusing on the elimination of the redundant actions such as "Do not apply for the passport." Or, the warning can focus on the superfluous precondition such as "Your passport is not needed for a domestic flight." The second warning would be more efficient and easier for the user to understand where the number of redundant actions is large. However, the problem of generating effective warnings for the Case 3 will be studied in our future work.

Figure 5 shows situations where a user assumes *p* is not a precondition of A, but it is a precondition of A in the system's operator library. In the cases of Figure 5(a) and Figure 5(c), the system identifies them as bad plans since the plans are not complete – *p* is still remain as an open condition of the plans. However, in the case of Figure 5(b), the system does not regard it as a bad plan because the addition of the operator B makes the precondition *p* of the operator A true.


start
A
w
w
goal
start
p
A
w
w
goal
(where p is a precondition of A)
A warning should be given to a user (a)
B
x
p
x
start
A
w
w
goal
start
B
x
p
x
p
A
w
w
goal
(where p is a precondition of A, and the ordering is B->A)
No warnings should be given to a user (b)
C
y
~p
B
x
p
y
x
start
A
w
w
goal
start
C
y
~p
B
x
p
y
x
p
A
w
w
goal
(where p is a precondition of A, and the ordering is B->C->A)
A warning should be given to a user (c)
Figure 5: A scenario of Case 4 ($\alpha \in L_U$ and $\alpha \in L_S$, $p \notin L_U$ and $p \in L_S$)

Situations where a user assumes *w* as an effect of the operator A, but it is not in the system's perspective appear in Figure 6. In the cases of Figure 6(a) and Figure 6(c), the plans are considered as bad plans since the goal of the user, *w*, is not achieved due to her wrong knowledge about A's effect. In the case of Figure 6(b), however, the plan is not a bad plan since the addition of plan operator B makes the goal condition, *w*, true.

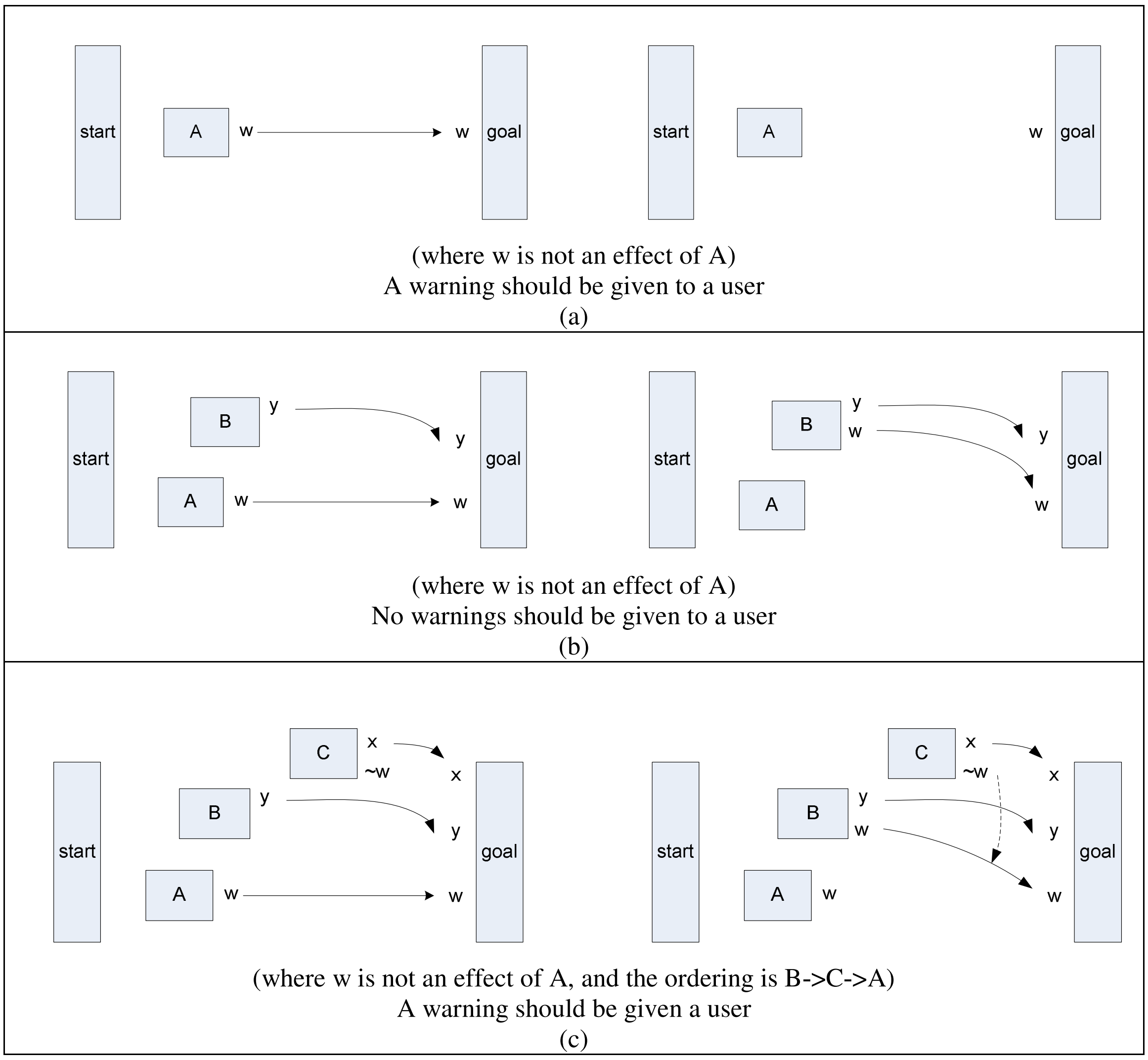
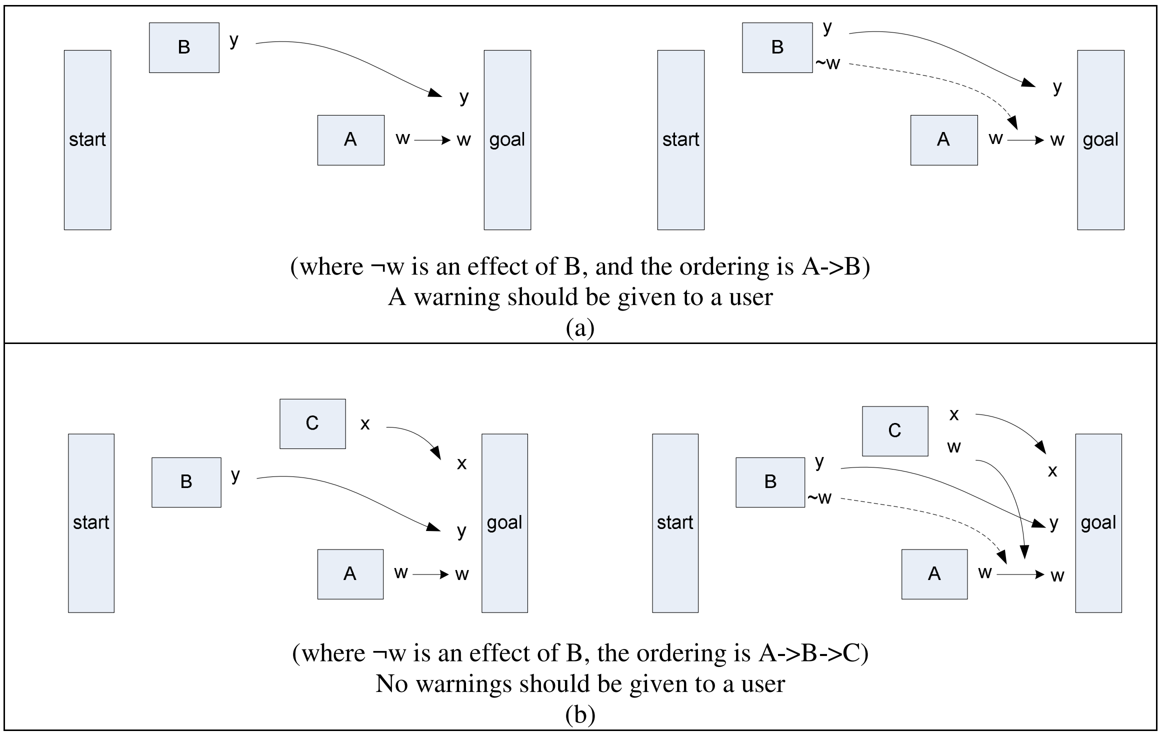

start
A
w
w
goal
start
A
w
goal
(where w is not an effect of A)
A warning should be given to a user
(a)
start
B
y
y
A
w
w
goal
start
B
y
w
y
w
A
goal
(where w is not an effect of A)
No warnings should be given to a user
(b)
start
C
x
~w
x
B
y
y
A
w
w
goal
start
C
x
~w
x
B
y
w
y
A
w
w
goal
(where w is not an effect of A, and the ordering is B->C->A)
A warning should be given a user
(c)

Figure 6: A scenario of Case 5 ($\alpha \in L_U$ and $\alpha \in L_S$, $w \in L_U$ and $w \notin L_S$)

Figure 7 presents situations where a user thinks ¬*w* or *y* is not an effect of the operator, but it is its effect in the system's perspective. In the case of Figure 7(a) it is a bad plan because the effect of operator B, ¬*w*, invalidates the goal condition. Nevertheless, in the cases of Figure 7(b), the plan is considered as a good plan since the goal of the user is achieved by the inclusion of the operator C which makes the goal condition, *w*, true.



(where ¬w is an effect of B, and the ordering is A->B)
A warning should be given to a user
(a)

(where ¬w is an effect of B, the ordering is A->B->C)
No warnings should be given to a user
(b)

Figure 7: A scenario of Case 6 ($\alpha \in L_U$ and $\alpha \in L_S$, $w \notin L_U$ and $w \in L_S$)

3.4 Encoder

When plans and their classes, good plan or bad plan, are sent from the bad plan identifier, the encoder produces training data for the next step. The training data is composed of plans, and each of the plans is represented by a series of attribute-value pairs. These pairs encode a plan's various features such as orderings, causal links, variable bindings, and hierarchical relationships. The naming an attribute is restricted by a set of rules so that the system can use the encoded information in the later phase. In addition, the encoder adds the information whether the plan is a bad plan or a good plan. The class value for a bad plan is "yes," and yet the class value for a good plan is "no." The details of the process of encoding features are exemplified in chapter 4. Currently, the task of encoding plan features is not much elaborated, and hence the work is done by human operators.

3.5 Construction of Bad Plan Classifying Features

In this section we present a bad plan constructor that builds a decision tree which determines the content of warnings. Given training data from the bad plan identifier, the constructor, based on the C4.5 algorithm (Quinlan, 1993), produces a set of plan features identifying bad plans. We choose the C4.5 algorithm over ID3 as our constructor because C4.5 algorithm builds efficient decision trees without redundant nodes which can be found in a decision tree when using the ID3 algorithm. Although C4.5 is based on ID3, the use of *information gain ratio* to measure the importance of features allows C4.5 to create more accurate decision trees during its training phase than those built by ID3. Information gain ratio is based on information gain, which is a difference between the previous information value and that after a decision or a feature is picked. In ID3 a feature which maximizes information gain is picked as the next node to be added to the current node. Information gain ratio is devised to compensate the ID3's preference for a tree which has nodes with only one number of member data. ID3 has another weakness of generating over-filtering decision trees. C4.5 overcomes this problem by pruning decision trees, which is a method of removing some portions of the tree if the part does not contribute the classification accuracy. The detail algorithm of C4.5 is described in (Quinlan, 1993).

In our system a decision tree is composed of plan features which differentiate bad plans from a set of complete plans of the user's plan space. The definitions for the C4.5 algorithm are shown in Figure 8.

Goal predicate	(IS Bad-plan?)
Decision tree	Consists of nodes (N) and edges (E) where N = {N_1, N_2, N_3, ..., N_i, .. N_n l N_i is a node } and E = { E_1, E_2, E_3, ..., E_i, .. E_n l E_i is an edge }
Node (N_i)	a decision point which is in accordance with a feature in examples
Edge (E_{in})	a possible value for a node N_i, " yes" or "no" n is the number of possible values ($0 < n < 3$)
Training set (T)	A list of examples T = {X_1, X_2, X_3, … X_i, .. X_n l X_i is an example }
Example (X_i)	a complete plan { (F_1, F_2, F_3, … F_i, … F_n ; D), where F_i is a feature-value pair and D is the value of goal predicate, which can be either "yes" or "no" }
Feature	the name of plan steps which compose a complete plan
Feature Value	If the feature is the name of a primitive plan step, then yes or no Else the name of a primitive plan step which achieve a abstract plan step's effects

Figure 8: The definition in decision tree learning algorithm

3.6 Message Simplification

This section specifically deals with a message simplification problem when the generated decision tree has a lot of number of nodes. If a branch of a decision tree is deep, giving a user a warning which includes all the plans leading to bad plans may cause him confused. As Young (1999) demonstrated that people showed a higher accuracy of performing a given task with concise instructions than given with detail instructions. He explained that a user may eliminate successful plans when he was given a rather lengthy instruction.

An example of a user's plan space is shown in Figure 9. In this figure, Pn means a plan, Sn represents a plan step, and ASn denotes an abstract plan step. Abstract plan steps cannot be performed directly. Therefore, it is decomposed into primitive plan steps. Si->ASj illustrates that Si is a primitive plan step for ASj. Good plans are represented as white ellipses, and bad plans are black ellipses. The planner starts with en empty plan, P0, and expands the plan whenever a plan step is added. If a primitive plan step is added to decompose an abstract plan, a hierarchical information about it is stored in the plan structure to express the relationship between the abstract operator and the primitive operator. The internal nodes in the diagram are incomplete plans, and the leaf nodes represent complete plans. Pn is a sequence of plan steps added from the root node to the node Pn. In Figure 9 bad plans are P20, P21, and P23. The plan P20 and P21 can be identified by a series of plan features (S4 as a primitive for AS1, S6 as a primitive for AS2, S8 as a primitive for AS3), and the plan P23 is identified by a series of plan features (S4 as a primitive operator for AS1, S5 as a primitive operator for AS2). Let the first series be Warning 1, and let the second series of plan features be Warning 2. A generated warning text corresponding to Warning 1 would be "If you choose S4 as a primitive action of AS1 and S6 as an action of AS2, do not do S8 to achieve AS3." And a text corresponding to Warning 2 would be "If you choose S4 as a primitive action of AS1, do not do S5 for achieving AS2." If the system wants to generate a warning which identifies all the bad plans, the generated warning would be the disjunction of Warning 1 and Warning 2. Then, the user would be confused due to the lengthy warning.

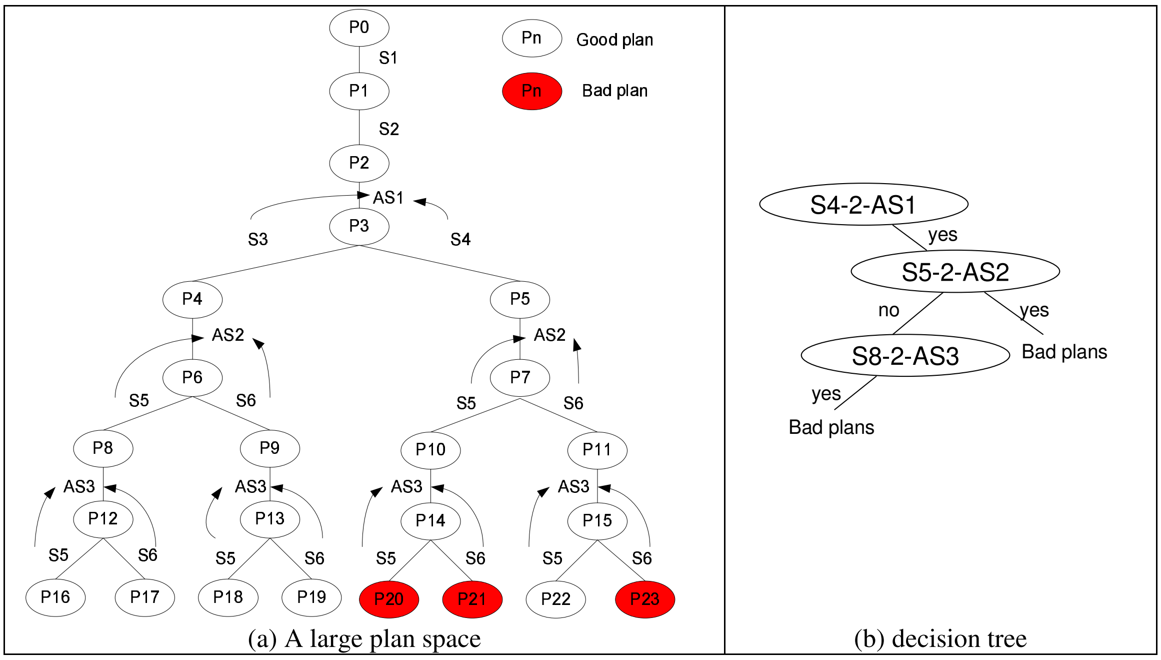


Figure 9: Deep Branch and its generated decision tree

In order to simplify the message so that the user can easily follow the recommendation from the system, the following three approaches are developed. First, simplifying message by *ignoring a number of good plans* can be considered. In the above example, with the cost of P7 the node S4 is picked as the node which includes all the bad plans. Instead saying above, the system can say "Do not choose S4 to achieve AS1." This approach entails the following limitations. When the bad plans are dispersed in the plan space, there is possibility that the highest classifying node can be a root. Then the bad plan classifying function can be threatened. How many plans shall be ignored and what plans should be selected as ignored nodes are also considerations. Second, *probabilistic approach* can be used when the system has probability value for each path. The probability can be obtained by monitoring the user's executions for some amount time. For instance, assume that the probability of choosing Warning 1 is 0.7, and that of Warning 2 is 0.4 with the previous example. In this case, the system may only give warning instructions generated from Warning 1. The weakness of this approach is apparent that the warning would be useless when the user chooses the other path. And, how to collect the probabilistic data is also another concern. Finally, we propose *pruning some part of the generated decision trees* as the most effective way. With the previous example, the (S4-S6-S8) plan steps and (S4-S5) plan steps can be reduced (S4-S6) and (S4) respectively. The disjunction of them will be definitely shorter than that of Warning 1 and Warning 2. The process of pruning the decision tree is also very simple – cutting from its leaves until the system has appropriate number of nodes.

3.7 Linguistic Realization

FUF and SURGE (Elhadad and Robin, 1996) are used as the surface realization component of our system. FUF is an extension of the functional unification formalism, and SURGE, written in FUF, is a grammar used for text generation in English. A translator is used to create an FD (functional descriptor) from the bad plan classifying features, i.e., the generated decision tree. Then the FD is instantiated using the content of the generated decision tree. Lastly, the instantiated FD is realized by FUF and SURGE.

4 Example

In this chapter, we provide an example illustrating how our system generates warnings in order to exclude unsuccessful plans in the user's plan space.

4.1 Planning

An example planning problem -- composed of an initial state and a goal state specification -- is defined as in Figure 10. This example be seen in a situation where a personal assistant agent informing the user how to get the place where she wants to go from the place where she currently is. It illustrates a situation where Anne is a person, Bus24 is a bus, Yellow-subway is a subway, and Anne is not at school or her lab at the initial state. The goal state – what the user wants to do – is to get Anne to the lab.

Initial state: ((IS-PERSON ANNE) (IS-BUS BUS24) (IS-SUBWAY YELLOW-SUBWAY) (NOT (AT-LAB ANNE)) (NOT (AT-SCHOOL ANNE)))
Goal state: ((AT-LAB ANNE))

Figure 10: Planning Problem

In the user's plan library, two abstract plan operators and a set of primitive plan operators are defined as in Figure 11 and Figure 12 respectively. COMMUTE-TO-SCHOOL and COMMUTE-TO-LAB are defined as abstract actions, which mean that they need primitive actions, which can be directly performed, to achieve their effects.

```
(define (action COMMUTE-TO-SCHOOL)
        :parameters (?P ?B ?SUB)
        :precondition ((NOT (AT-SCHOOL ?P) )
        :effect ( (AT-SCHOOL ?P) ))
(define (action COMMUTE-TO-LAB)
        :parameters (?P ?B ?SUB)
        :precondition ((AT-SCHOOL ?P) )
        :effect ( (AT-LAB ?P) ))
```

Figure 11: Abstract plan operators of a user

```
(define (action PUT-ON-SHOES)
        :parameters (?P)
        :precondition NIL
        :effect ((WEARING-SHOES ?P))
        :constraints ((IS-PERSON ?P)))
(define (action LEAVE-FROM-HOME)
        :parameters (?P)
        :precondition ((WEARING-SHOES ?P))
        :effect ((LEFT-FROM-HOME ?P) )
        :constraints ((IS-PERSON ?P)))
(define (action TAKE-A-SUBWAY)
        :parameters (?P ?S)
        :precondition ((LEFT-FROM-HOME ?P))
        :effect ((AT-LOC ?P SCHOOL) (DISTANCE 3MILE))
        :constraints ((IS-SUBWAY ?S) (IS-PERSON ?P)))
(define (action TAKE-A-BUS)
        :parameters (?P ?B )
        :precondition ((LEFT-FROM-HOME ?P))
        :effect ((AT-LOC ?P SCHOOL) (DISTANCE 1MILE))
        :constraints ((IS-BUS ?B) (IS-PERSON ?P)))
(define (action TAKE-AN-ELEVATOR-TO-LAB)
        :parameters (?P)
        :precondition NIL
        :effect ((AT-LOC ?P LAB) )
        :constraints ((IS-PERSON ?P)))
(define (action WALK-TO-LAB)
        :parameters (?P)
        :precondition NIL
```

```
            :effect ((AT-LOC ?P LAB) )
            :constraints ((IS-PERSON ?P)))
```

Figure 12: Primitive plan operators of a user

To achieve the goal of Anne's being at the laboratory, the COMMUTE-TO-LAB abstract action is first chosen. As a primitive action for this, either TAKE-AN-ELEVATOR-TO-LAB or WALK-TO-LAB can be utilized. Since the precondition of the COMMUTE-TO-LAB action is (AT-SCHOOL ?P), an abstract action COMMUTE-TO-SCHOOL must be performed before COMMUTE-TO-LAB. Again, COMMUTE-TO-SCHOOL can have TAKE-A-SUBWAY or TAKE-A-BUS as its primitive action since its effect of (AT-SCHOOL ?P) is in accordance with the effect of TAKE-A-SUBWAY or TAKE-A-BUS. A plan space generated from the defined library and planning problem is shown in Figure 13. In this figure, abstract plan steps are surrounded by dotted-line boxes, and a label (e.g., $S_{n)}$ is assigned to each. The abstract plan step (COMMUTE-TO-SCHOOL ANNE BUS24 YELLOW-SUBWAY) is expanded to include (TAKE-A-BUS ANNE BUS24) or (TAKE-A-SUBWAY ANNE YELLOW-SUBWAY). The abstract plan step (COMMUTE-TO-LAB ANNE BUS24 YELLOW-SUBWAY) can be achieved by the inclusion of either (WALK-TO-LAB ANNE) or (TAKE-AN-ELEVATOR-TO-LAB ANNE).

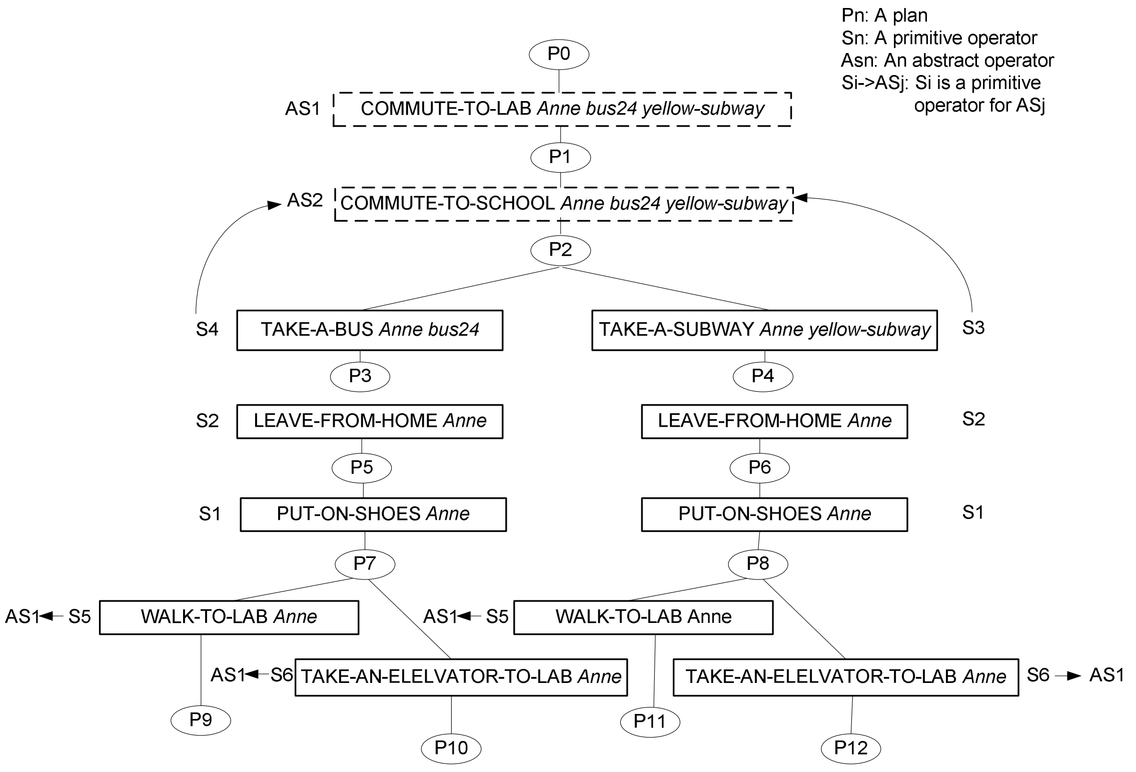


Figure 13: The user's Plan space

4.2 Identifying Bad Plans and Encoding Plan Features

In the previous section we have shown that four plans (i.e., P9, P10, P11, and P12) are generated by modeling of the user's planning process. Now, suppose that there is no plan P3 in the system's plan space because (WALK-

TO-LAB) operator in the system library requires an additional precondition: the distance shall be less than 2 miles. Thus, P3 is identified as a bad plan according to the case 4 in Figure 3.

In this phase, a set of input features corresponding to each plan is encoded according to a specific naming convention. The encoded features of plan P11 of Figure 13 appears in Figure 14. The features include hierarchical, ordering, variable binding information of the input plan. Bold face in the figure denotes an ordering or hierarchical relationship between two actions or variable binding information. The class of plan P11 is encoded as "yes" since it is bad plan. Other good plans (P9, P10, P12) are encoded in the same way as P11 is encoded, but the difference is that the class value of good plans is "no."

(PUT-ON-SHOES yes)
(PUT-ON-SHOES-**BIND**-ANNE yes)
(PUT-ON-SHOES-**BEFORE**-LEAVE-FROM-HOME yes)
(LEAVE-FROM-HOME yes)
(LEAVE-FROM-HOME-**BIND**-ANNE yes)
(TAKE-A-SUBWAY-**2** -COMMUTE-TO-SCHOOL yes)
(TAKE-A_SUBWAY-**BIND**-ANNE yes)
(TAKE-A_SUBWAY-**BIND**-YELLOW-SUBWAY yes)
(WALK-TO-LAB-**2**-COMMUTE-TO-LAB yes)
(IS-BADPLAN? yes)

Figure 14: A set of input features of plan P1

4.3 Generation of Warnings from Plan Features

With given training data produced by the encoder the C4.5 algorithm generates a bad plan classifying decision tree as in Figure 15. By looking at the tree, we acknowledge that the combination of the plan feature "TAKE-A-SUBWAY-2-COMMUTE-TO-SCHOOL" action and the plan feature "WALK-TO-LAB-2-COMMUTE-TO-LAB" leads the user's plans to fail.

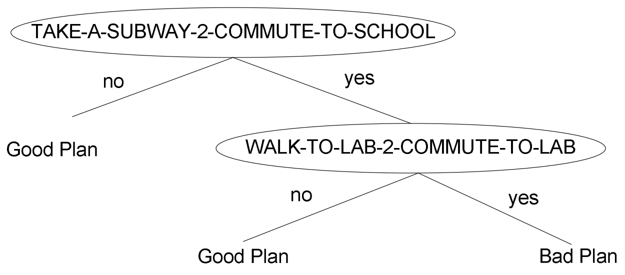

TAKE-A-SUBWAY-2-COMMUTE-TO-SCHOOL
no
yes
Good Plan
WALK-TO-LAB-2-COMMUTE-TO-LAB
no
yes
Good Plan
Bad Plan

Figure 15: A generated decision tree from the plan space

On receiving the decision tree from the constructor, the translator chooses an FD based on the structure and content of the generated decision tree. As shown in Figure 10, the generated tree has two nodes having hierarchical plan features. For example, by interpreting the root node "TAKE-A-SUBWAY-2-COMMUTE-TO-SCHOOL" yields that TAKE-A-SUBWAY is a primitive action for achieving the effect of the abstract operator COMMUTE-TO-SCHOOL. The translator then searches for a proper FD template corresponding to a tree having two nodes which encode hierarchical plan features. The found FD template is filled with the content information of the decision tree. Finally, the above example can be realized into either "Do not walk to lab if you want to take a subway to school" or "Do not take a subway to school if you want to walk to the lab." Since we elaborate on how to build the content of warnings, the process of surface realization is briefly introduced here.

5 Discussion

This research has attempted to provide an algorithm for determining the content of warnings; the algorithm that we have defined functions by eliminating bad plans from consideration by the user by applying the C4.5 algorithm to the user's plan spaces. In this article, we demonstrated an example that an automated warning generation is feasible by our system. The issues of Grice's maxim of quantity and message simplification are also mentioned in this paper. Furthermore, we briefly describe the use of our system to account for warnings in ways that are sensitive to task context. For instance, "taking a subway" can be a good plan in New York City, but it would not work where there is no public transportation.

While the results of our work are encouraging, we acknowledge that the use of plan features as input to decision tree learning algorithms has several limitations. First, the number of input features increases exponentially as the number of plan steps grows. As a result, it may take exponential time to construct bad plan classifying features in such a situation. Instead of using all the features of plans, encoding only a small number of important features, e.g., just plan steps, could reduce the number of input features and help overcome the complexity problem. Second, when the decision tree is large, selecting a proper text structure for the decision tree is another issue.

We believe that our system can be used for personal assistant agents generating appropriate instructions for users; the problems raised in this report will be addressed in our future work.

References

(Ayres et al, 1994) Thomas J. Ayres, Madeleine M. Gross, Christine T. Wood, Donald P. Horst, Roman R. Beyer, and J. Neil Robinson. 1994. What is Warning and When Will it Work? *Human Factors Perspectives on Warnings*, Edited by Kenneth R. Laughery, Sr., Michael S. Wogalter, and Stephen L. Young, the Human Factors and Ergonomics Society. Pages 1-5.

(Ansari, 1995) Daniel Ansari. 1995. Deriving Procedural and Warning Instructions from Device and Environment Models. *Technical report CSRI-329, Department of Computer Science*, University of Toronto.

(Aone and Bennett, 1995) Chinatsu Aone and William Bennett. 1995. *Evaluating* Automated and Manual Acquisition of Anaphora Resolution Strategies. In *Proceedings of the 33rd Annual Meeting of the ACL,* pages 122-129.

(Di Eugenio et al, 1997) Barbara Di Eugenio, Johanna D. Moore, and Massimo Paolucci. 1997 Learning Features that Predict Cue Usage. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 80-87.

(Elhadad and Robin, 1996) Michael Elhadad and Jacques Robin. 1996. An overview of SURGE: A reusable comprehensive syntactic realization component. *Technical Report Technical Report 96-03, Dept of Mathematics and Computer Science*, Ben Gurion University, Beer Sheva, Israel.

(Hovy, 1993) Eduard H. Hovy. 1993. Automated Discourse Generation Using Discourse Structure Relations. *Artificial Intelligence*, vol. 63(1-2), pages = 341-385.

(Lambert and Carberry, 1991) Lynn Lambert and Sandra Carberry. 1991. A Tripartitie Plan-Based Model of Dialogues. In *Proceedings of the ACL*, pages 47-54.

(Mann and Thompson, 1988) William Mann and Sandra Thompson. 1988. Rhetorical Structure Theory: Towards a Functional theory of text organization. *TEXT*, 8(3).

(McAllester and Rosenblitt, 1991) David McAllester and David Rosenblitt. 1991. Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, vol. 2, Anaheim, California, USA, pages 634--639.

(Mellish and Evans, 1989) Chris Mellish and Roger Evans. 1989. Natural Language Generation From Plans. *Computational Linguistics,* Vol. 15(4), pages 233-249.

(Moore and Paris, 1994) Johanna D. Moore and Cecile L.Paris. 1994. Planning Text for Advisory Dialogues: Capturing Intentional and Rhetorical Information. *Computational Linguistics*, vol. 19(4), pages 651-694.

(Moore and Pollack, 1992) Johanna D. Moore and Martha E. Pollack. 1992. A Problem for RST: The Need for Multi-Level Discourse Analysis. *Computational Linguistics*, Vol. 18 (4), pages. 537-544, 1992.

(Nakano and Kato, 1998) Yukiko I. Nakano; Tsuneaki Kato. Cue Phrase Selection in Instruction Dialogue Using Machine Learning. Workshop on Content Visualization and Intermedia Representations (CVIR'98), pages 100-106, 1998.

(Penberthy and Weld, 1992) J. Scott Penberthy and Daniel S. Weld. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 103-114, Cambridge, MA.

(Quinlan, 1993) J.Ross Quinlan. 1993. C4.5: *Programs for Machine Learning*. Morgan Kaufman.

(Vander Linden and Di Eugenio, 1996) Vander Linden, K. and Di Eugenio, B. 1996. Learning micro-planning rules for preventative expressions. In Proceddings of the Eighth International Natural Language Generation Workshop, pages 11-20, Sussex, UK.

(Young et al, 1994) R. Michael Young, Martha E. Pollack, and Johanna D. Moore. 1994. Decomposition and causality in partial-order planning. In *Proceedings of the Second International Conference on AI and Planning Systems*, Chicago, IL.

(Young, 1999) R. Michael Young. 1999. Using Grice's maxim of Quantity to select the content of plan descriptions. *Artificial Intelligence*, vol. 115(2), pages 215-256.