1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

THÈSES CANADIENNES

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction même partielle de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Canadian Theses
on Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale
du Canada

Services des thèses canadiennes
sur microfiches

0-315-22239-5

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Ecrire en lettres moulees ou dactylographier

Full Name of Author — Nom complet de l'auteur

KARL GERHARD ROTH

Date of Birth — Date de naissance

JUNE 8, 52

Country of Birth — Lieu de naissance

WEST GERMANY

Permanent Address — Residence fixe

1195 AGINCOURT RD.
OTTAWA, ONT. K2C 2H8

Title of Thesis — Titre de la these

TOWARDS AN INTELLIGENT ASSISTANT
IN A TASK ENVIRONMENT

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette these fut presentee

MASTER OF COMPUTER SCIENCE

Year this degree conferred — Année d'obtention de ce grade

1985

Name of Supervisor — Nom du directeur de these

WILF LALONDE

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est par la presente accordée à la BIBLIOTHEQUE NATIONALE DU CANADA de microfilmer cette these et de prêter ou de vendre des exemplaires du film.

L'auteur se reserve les autres droits de publication ni la these ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

JAN 7, 85

Signature

Barbara Rich

**Towards an Intelligent Assistant
in a Task Environment**

By

Gerhard Roth

**A Thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for
the degree of
Master of Computer Science**

School of Computer Science

Carleton University

Ottawa, Ontario

September 30, 1984

© Copyright

1984, Gerhard Roth

Thesis Acceptance Form - Master's

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis .

TOWARDS AN INTELLIGENT ASSISTANT

IN A TASK ENVIRONMENT

submitted by Gerhard Roth, B. Math.

in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chairman, Department of Computer Science

Carleton University

8 January, 1985

Abstract

This thesis describes an attempt to implement an intelligent assistant in a task environment. The function of the assistant is to help a person by giving information that is necessary for the successful performance of a task. In order to do this the assistant infers the goals, simulates the planning and models the beliefs of the person. The communication between the person and the assistant is assumed to be in parsed english, an english that has been parsed into a semantic form. This thesis concentrates on explaining the reasoning that is used to process dialogues given the semantic form. The result of the thesis is a computer program that can engage in some limited dialogues, but is not yet intelligent in the sense that we have defined this term.

Acknowledgements

First of all I want to thank my advisor, *Wulf Lulonde* for the support, patience, and advice that have helped me so much, and for the use of his TeX macros. I would also like to thank *Mark Waldvogel* and my wife *Cathy* for patiently reading and commenting on earlier drafts of this thesis.

This work has been partially supported by Defense Research Establishment Atlantic under grant No. 284E and by the Natural Sciences and Engineering Research Council of Canada under grant No. 284B. I would also like to thank the School of Computer Science for providing such excellent computing facilities.

Dedication

This thesis is dedicated to my wife *Cathy*, my son *Andrew* and to my friends at our Thursday prayer group.

Colossians 3:17

"και παν ο τι εαν ποιητε εν λογω η εν εργω παντα εν ονοματι κυριου Ιησου, ευχαριστουντες τω θεω πατρι δι αυτου"

The Greek New Testament United Bible Societies

"And whatever you do, whether in word or in deed, do it all in the name of the Lord Jesus, giving thanks to God the Father through him."

New Testament New International Version

Table of Contents

Abstract	i
Acknowledgements	ii
Dedication	iii
Chapter 1 Introduction	1
1.1 Objectives	1
1.2 Justification	2
1.3 Statement of Thesis	3
1.4 Contributions	4
1.5 Related Research	5
1.6 Assumptions	5
1.7 System Architecture	6
1.7.1 Knowledge Representation	6
1.7.2 Reasoning Process	7
1.8 An Example	7
1.9 Thesis Overview	8
Chapter 2 Related Research	9
2.1 Planning	9
2.2 Cognitive Modelling	11
2.2.1 Definition	11
2.2.2 Compared to Expert Systems	11
2.3 Common Sense Reasoning	13
2.4 Intensional Concepts	14
2.5 Speech Acts	16
2.5.1 Definition	16
2.5.2 Implemented Systems	19
2.6 Natural Language Processing	20
2.6.1 SHRLDU	21
2.6.2 TDUS	22
2.6.3 PAM	23
2.6.4 UC	24
2.7 PROLOG	26
Chapter 3 Fundamental Concepts	28
3.1 Possible Worlds Prolog	28
3.2 Knowledge Representation	35
3.2.1 Beliefs	35
3.2.2 Mutual Beliefs	37
3.2.3 Knowledge	39
3.2.4 World	39
3.3 Actions	41
3.3.1 Action Definition	41
3.3.2 Action Description	42
3.4 Plans	46
3.4.1 Plan Definition	46
3.4.2 Planning Method	47

Chapter 4 System Description	51
4.1 Overview	51
4.2 Parse	53
4.3 Speech Act Processing	54
4.4 Plan	58
4.5 Warn	61
4.6 Infer	63
4.7 Execute	67
4.8 Justify	71
4.9 Debug	72
Chapter 5 Examples	76
5.1 Monkey And Bananas	76
5.1.1 Situation	76
5.1.1.1 Dialogue	77
5.1.1.2 Explanation	78
5.2 Simple Generator Installation	79
5.2.1 Situation	79
5.2.1.1 Novice Dialogue	80
5.2.1.2 Novice Explanation	80
5.2.1.3 Expert Dialogue	81
5.2.1.4 Expert Explanation	82
5.3 Inferring Generator Installation Goal	83
5.3.1 Situation One	83
5.3.1.1 Dialogue	83
5.3.1.2 Explanation	84
5.3.2 Situation Two	84
5.3.2.1 Dialogue	84
5.3.2.2 Explanation	85
5.3.2.3 Dialogue	85
5.3.2.4 Explanation	85
5.3.2.5 Dialogue	86
5.3.2.6 Explanation	86
5.3.3 Situation Three	87
5.3.3.1 Dialogue	87
5.3.3.2 Explanation	87
5.4 Complex Generator Installation	88
5.4.1 Situation	88
5.4.1.1 Dialogue	89
5.4.1.2 Explanation	91
Chapter 6 Conclusions	93
6.1 Evaluation	93
6.2 Future Work	95
6.2.1 Short Term	95
6.2.2 Long Term	95
6.3 Summary	97
Chapter 7 References	98

Appendix A Monkey And Bananas	1
A.1 Beliefs	1
A.2 Examples	3
Appendix B Simple Generator Installation	1
B.1 Beliefs	1
B.2 Examples	4
Appendix C Inferring Generator Installation Goals	1
C.1 Beliefs	1
C.2 Examples	5
Appendix D Complex Generator Installation Goals	1
D.1 Beliefs	1
D.2 Examples	6
Appendix E Program Listing	1
E.1 Builtin	1
E.2 Facts	3
E.3 Knowrep	4
E.4 Warn	6
E.5 Plan	8
E.6 Spact	12
E.7 Infer	14
E.8 Execute	17
E.9 Debug	21
E.10 Justify	22
E.11 Main	23
E.12 Hierarchy	25
E.13 Transform	26

Introduction

1.1 Objectives

The goal of this thesis is to create a system that can act as an intelligent assistant in a task environment. A task environment is defined to be one where a person is trying to accomplish a physical task such as setting up a Video Tape Recorder or putting together a tricycle. The person is not completely knowledgeable and needs some assistance. The system should perform the function of an assistant who is an expert in that task. The system and the person communicate in a version of English that has been parsed into a semantic form. An assistant should provide the following three main functions:

1. Helping — telling the person something that he needs to know to perform the task
2. Warning — telling the person something that he needs to know to avoid dangerous situations
3. Debugging — perform error recovery when the system and the person do not agree. This usually happens because the person has made a mistake.

We believe an intelligent assistant must provide these functions. However, providing them is only a necessary, and not a sufficient, condition for the system to be intelligent.

The only way the system can be shown to be intelligent is if it passes our version of the Turing test [Turing, 63]. Alan Turing was a computer scientist who proposed a test to answer the question "Can computers think?" The test was to have a person (the tester) communicate with either another person or a computer via a terminal but with the proviso that the tester not know which was which. The tester is able to ask any question of the two and has the goal of distinguishing them by judging the responses. If the tester cannot tell the machine and the person apart, the machine can then be said to think. In our version of

this test the system is intelligent if the person cannot distinguish it from a human expert in the task environment. In other words, if the person were not told whether a human or a computer were giving advice there should be no way for the person to tell the kind of expert from the advice that has been given.

1.2 Justification

In order to justify this work we must discuss the *deficiencies* of current systems in this area. This requires determining the characteristics of human dialogues that current natural language systems cannot duplicate and that have to do specifically with enabling an assistant to provide the functions of helping, warning, and debugging. There are four such characteristics that we have identified.

First, people engage in goal directed dialogues. Consider the following task environment.

- A person is standing below a banana tree with the bananas just out of reach.
- The person says, "I am hungry" and the assistant responds "There is a stick by the rubber tree"

To explain this conversation, that is to understand it, the hypothesis is that the assistant has

- knowledge of the situation and state of mind of the participants, and
- some reasoning mechanism that uses this knowledge to produce the proper response.

It is clear that in human discourse what was said (the literal meaning) is not the same as what was intended (the real meaning). This is an important characteristic of discourse and is one of the reasons computer systems have problems in engaging in *natural* dialogues.

Second, people can explain their actions and the reasons behind them. If the person responds with "why did you tell me that?", the likely reply from the assistant would be "because you said you were hungry". In fact the explanation is more involved, but if pressed, the assistant could come up with the reasoning behind the reply. This form of reasoning is called common sense reasoning because this ability is common to a large part of the population.

Third, people can keep track of other people's beliefs and can use them to guide their reasoning. If the assistant knows the hungry person hated bananas the reply would not be the same. Notice that this would be true even if the assistant disagreed, and personally liked bananas.

Fourth, people can remember the past, imagine the future, and discuss both in the present. If the assistant knew that the bananas were poisonous he would say so because he can imagine the person being dead in a possible future where the person ate the bananas.

We claim that these four things are not present in current natural language systems, and are necessary to provide the functions of helping, warning, and debugging (and thus to pass the Turing Test). Accordingly our thesis revolves around creating a system that has these capabilities to some degree or another. Similar arguments are made in [Perrault & Cohen & Allen, 82] and in [Grosz, 80] who says:

"A merger of research in common sense reasoning and language processing is an important goal both for developing a computational theory of the communicative use of language and for constructing computer based natural language systems."

1.3 Statement of Thesis

Our thesis is that an intelligent assistant in a task environment can be created by combining the use of speech acts, performing plan recognition and simulation, and keeping track of the person's beliefs. Furthermore there must be a new system architecture to coordinate these processes.

As evidence for this thesis we have implemented these processes in a program and demonstrated this program's "intelligence" by a number of dialogues. Though the system is not intelligent enough to pass the Turing Test (and thus to prove the thesis) the dialogues that it can engage in do provide excellent evidence that the thesis is on "solid ground". In fact, in our exploration of this thesis we have shown that it is possible to use speech acts, plan recognition and simulation as well as recording the persons beliefs with the specific system architecture used, and come up with a system that has some intelligence, even if it does not yet equal human intelligence.

1.4 Contributions

The processes that are used in the system (speech acts, plan recognition, plan simulation and belief modelling) have been studied, but always in isolation from each other. They have not been put together to create a system that can handle even the simple examples of §1.2. Our system can handle this dialogue (and more complex ones) and thus demonstrates the state of the art.

The architecture that an intelligent natural language system should have has not been explored, though a number of proposals have been made. Our system architecture shows how these processes work in unison towards creating an intelligent assistant in a task environment

In order to create an intelligent assistant the planning simulation process must keep track of the past worlds and the possible future worlds. This is done using a simple and efficient technique which is integrated with a PROLOG interpreter to create a possible worlds version of PROLOG.

1.5 Related Research

The most important works that we base our system on are [Allen, 79] and [Wilensky 83]

Allen describes a way of handling goal directed dialogues using the idea of a speech act. A speech act models human discourse from the point of view of the *intention* of the utterance (statement) that has been made during a dialogue. This is coupled with a plan deduction process which the hearer uses to reconstruct the plan of the speaker in order to find the true meaning of the utterance (that is the actual intention of the speaker). Our system architecture is very similar to that described in [Perrault & Allen & Cohen 82] who claim that natural language systems must adopt such an architecture in order to move beyond just answering questions about a database.

Wilensky proposes a model for common sense reasoning that has influenced our architecture. In this model one uses situations in an imagined future to affect what one does in the present. We use this idea to get the system to perform cooperative behavior which is defined to mean the system telling the person something he should know. This is done by having the system imagine future situations and from these situations determine where it should perform such behavior.

1.6 Assumptions

The following assumptions have been made

- There is a single person who knows the consequences of his beliefs (this really means he actually plans in the way the system simulates his planning)
- All the objects that exist in the environment are known by both the person and the system (though some things about them may not be known)
- The person is always honest and never lies

- The person is aware that the system knows everything about how to accomplish the task and that it wants to be helpful
- The system is aware of what the person knows (his level of expertise) in the initial state
- The dialogue occurs in the following pattern
  - 1 The goal is stated (or inferred) from the person's utterance
  - 2 The system performs cooperative behaviour which may result in the person being given some information
- The system always accepts the person's goals as its own and tries to help the person achieve his goal by behaving cooperatively
- The only types of questions that can be answered are those that are defined in the knowledge representation

1.7 System Architecture

In keeping with expert system methodology we separate the knowledge from the reasoning processes that uses that knowledge

1.7.1 Knowledge Representation

Our implementation uses a version of PROLOG so our representation is in terms of clauses. A world consists of all the clauses that describe a situation. This description is in terms of what the system believes and what the system believes the person believes. Worlds are created from other worlds by actions so the knowledge representation is actually a tree of worlds connected by actions. One world is designated to be the present world, and there are past worlds and future worlds. Together all the worlds are called a plan tree which is really a primitive temporal memory for the system. The reasoning processes create these worlds and use the information in them to guide the system's interactions with the person.

1.7.2 Reasoning Process

The reasoning process consists of the four parts which are explained below

Parsing takes the english input and converts it into some semantic representation. It may use the plan tree to aid in the translation. This is an area that has been studied in great detail so we will assume that such a capability exists.

Speech Act Processing performs some reasoning to recognize the meaning of the utterance which we define as the speech act that the utterance represents. The reasoning uses general speech act rules along with specific facts about the current situation.

Planning takes any goal produced by the speech act processing and produces a plan tree for that goal. Since the goal is always a goal that the person has, the system must imagine what the person must do to achieve this goal and how it can behave cooperatively. The resulting plan tree is grafted onto the present world to provide future expectations.

Execution takes the plan tree and executes some path by performing the system actions (telling the person something) and adjusting the present world. This may require debugging if the system and the person disagree about something. The most common reason for disagreement is that the person has made a mistake by either forgetting to do something or by doing it in the wrong way.

1.8 An Example

Consider an expansion of the situation that we described earlier

- A person is underneath a banana tree with the bananas out of reach
- An apple is on the ground in front of the person
- The assistant knows the apple is bad to eat
- The assistant knows there is a stick beside the rubber tree

Imagine the following conversation taking place

Person I am hungry

Assistant The apple is not good to eat so you should eat the bananas

There is a stick beside the rubber tree

The beliefs that the system must have for this conversation are in Appendix A. This example shows the ability of the system to infer user goals and to behave cooperatively. It is discussed in more detail in §5.1.

1.9 Thesis Overview

For a minimal understanding of this thesis the reader should consult the Examples chapter and the Conclusions chapter. The reader who has a limited background in this area should also consult the Related Research chapter (also see [Hendrix & Sacerdoti, 81] and [Grosz, 80]).

For an understanding of how things actually work the Fundamental Concepts chapter is necessary. After this the Examples chapter should be reread in conjunction with the System Description Chapter to enlarge this understanding.

Related Research

This chapter describes the research that is related to our work. Each area that is surveyed is compared and contrasted with our work.

2.1 Planning

One of the earliest planners was STRIPS [Fikes & Nilson, 73], which made use of actions with add and delete lists to overcome the frame problem. The frame problem is the difficulty of recording in a logical form the fact that when an action is performed most of the things in the world do not change. For this reason early attempts to do planning with a theorem prover were not very successful.

Kowalski [79] showed that the frame problem could be solved using logic without any special action operators, but there has been little reason to use this approach in practice. Most of the planners that we will survey use actions in the STRIPS format with add and delete lists.

The STRIPS system was used to model the planning for a blocks world which has blocks of various sizes and a robot that can manipulate these blocks. A typical goal in a blocks world is to stack the blocks on top of each other in a certain order. The STRIPS planner does not handle the problem of interacting subgoals in an efficient manner. An example of this is the situation where you have the goals of painting the ladder and of painting the ceiling, for which you need the ladder. If you achieved the first goal before trying to achieve the second goal you would not be able to achieve the second goal until the paint on the ladder is dry. In order to solve the problem of interacting subgoals a STRIPS planner would have to try every permutation of the goal order until it achieved an order without interactions.

The problem of interacting subgoals was solved more efficiently in NOAH [Sacerdoti, 1977]. This was accomplished by using the strategy of least commitment. This means delaying the commitment of the ordering of subgoals until the need for some ordering exists; e.g., one action undoing the preconditions of another action.

In order to detect interactions between subgoals NOAH used critics to examine the current plan and suggest ways of modifying the plan when necessary. These techniques may produce plans that are nonlinear. This means that when given a conjunction of goals the planner might produce a plan that does not completely solve one subgoal before going on to the next subgoal, but may interleave the plans for each subgoal.

The techniques of NOAH have been further refined by DEVISER [Vere, 1981] which added time constraints and event durations to plans and in SIPE [Wilkins, 1983] which added constraints and resources to the planning process.

Another important idea that has been developed for planning is that of a hierarchical planner. This is a planner that produces a skeleton of a plan at one level and fills in the details at a lower level. An example of a hierarchical planner was ABSTRIPS [Sacerdoti, 1973], a modification of STRIPS. It classes the preconditions of each action according to their difficulty and creates a plan considering only the most difficult preconditions, then the next most difficult preconditions, and so on. If the preconditions are graded accurately the planning process becomes much faster. Note that this is *not* the same as a planner which has subtasks as part of an action (such as our planner).

Our planner is a simple STRIPS derivative which does not handle interacting subgoals and is not hierarchic. This is sufficient for a prototype but a practical system would require these features.

2.2 Cognitive Modelling

2.2.1 Definition

The field of cognitive modelling as a discipline of artificial intelligence was pioneered by Schank [81]. Schank is a computer scientist with a background in linguistics who did not believe that purely syntax-oriented approaches would be sufficient to produce programs that truly understood natural language. This is because syntax recognition is only a part of the cognitive process involved in natural language understanding. It is still important, but the rest of the cognitive process is less well understood and more difficult. The desire to understand these cognitive processes lead to a series of programs (SAM, PAM, BORIS, ...) [Schank, 81] that are computational models of natural language understanding. The resulting programs were said to have understood what they read because they could answer the same questions as a person who had read the story.

Cognitive modelling tries to model human behavior by a computational process (algorithm). A better term might be computational modelling, since the cognitive model need not have any psychological significance, that is it need not work in the same way as a person does, just get the same result. The degree of intelligence of the system is the degree to which it passes the Turing Test. A cognitive scientist might make a stronger claim with the regard to the psychological significance because these models *seem* to have this characteristic. However they are not tested in any formal fashion to see whether this is really the case. We have a cognitive model of the expert assistant as a computer program that we call "the system".

2.2.2 Compared to Expert Systems

Expert systems [Shortcliffe, 76] are programs that model highly skilled humans in specialized domains. We make the claim that expert systems and cognitive modelling are

not different fields, but are simply offshoots of the same root. A similar argument is found in [Dyer, 83] who says:

"The popular distinction, Expert Systems versus Cognitive Modeling, is a convenient one to make, but is somewhat simplistic, since the distinction is essentially one of focus or emphasis. Some "expert systems" are actually years away from robustness, while other "expert systems" follow human protocols very closely. Clearly both approaches are needed. Cognitive modelling gives us insight into fundamental human cognitive skills, while high performance Expert Systems test AI theory in the real world, yielding important feedback and enhancing credibility for the field as a whole"

Another potential definition of expert systems is from IBM¹ which defines an expert system as:

"A program that uses explicitly represented knowledge and computational inference procedures to solve problems requiring significant expertise"

By this definition almost any large AI program is an expert system. This definition does make the important point that there are two parts to an expert system and they are as follows:

1. Knowledge Representation - some method of *declaring* the knowledge used in the system
2. Reasoning Process - some way of *using* the knowledge to obtain the desired results

The current generation of expert systems has a knowledge representation of "if, then" rules and a reasoning method of backwards and forwards chaining. We believe that to produce expert systems that work in varied domains (such as the task environment) we must use other knowledge representation schemes and reasoning methods. Our system for the task environment is designed using the expert systems methodology of separating the knowledge from the reasoning. We claim that our system is a cognitive model that is on its way to becoming an expert system. In other words we have some, but not all of the basic mechanisms necessary for expertise in our application.

¹ Toronto Globe and Mail, Friday, June 29, 84

2.3 Common Sense Reasoning

The goal of common sense reasoning is to provide cognitive models of the reasoning that takes place in every day situations that most people can handle, as opposed to situations that require specialized skills. The common sense planning loop that has influenced our system is that of [Wilensky, 84] which consists of the following steps.

1. goal detection which examines the current world to detect goals the intelligence should have;
2. plan proposal which creates plans to achieve these goals by lookup or by traditional planning methods;
3. projection which simulates execution of these plans by building hypothetical worlds and performing goal detection, thus invoking the planning loop recursively, and
4. execution which executes the resulting plan.

This loop handles certain common sense situations by *meta planning*. *Meta planning* means having plans and goals that are *about the planning process*. A meta goal comes up during the planning process as a result of some difficulty noticed by the goal detector. An example of a meta goal is the goal "resolve goal conflict". A goal conflict occurs when the intelligence has two goals that are in conflict, that is they cannot both be achieved together. For example consider the story found in [Wilensky, 84].

"John wanted to get to Las Vegas fast by speeding, however there were many policeman on the way and he did not want to get a ticket. John decided to drive slowly."

The planner has conflicting goals of getting to Las Vegas quickly and of not getting a ticket. One solution is the meta plan of abandoning one of the goals, which the planner choose.

The key thing is that meta goals and meta plans are handled *in the same way* as ordinary goals and plans by the planner. This makes possible a uniform view of common

sense reasoning which is largely concerned with how people modify stereotyped plans to fit different situations. The difficulty is that many of the meta plans are more like heuristics and seem to require learning and experimentation to get results. Since learning is a very difficult problem with little success to date this might lead us to despair of ever applying these ideas. However, we can use the idea of constraints to summarize simple meta goals. If some projected future violates a constraint, we have a meta goal of making sure that this future does not come about. In our system this could be done by the system telling the person the constraint and leaving it up to the person to realize that this constraint will be violated. This knowledge would force the person to abandon this particular path and to seek another method of achieving his goal that did not violate a constraint. The important things we take from common sense reasoning are the projection of the affects of a plan and the use of constraints to analyze future situations in order to modify behavior in the present.

2.4 Intensional Concepts

Intensional concepts take into account the meaning, rather than the truth of a statement. Belief is an intensional concept because whether some intelligence believes a proposition P depends on the meaning of P, not the truth of P. For example, it may be true that some person believes that the moon is made of green cheese while another person believes that this is wrong and the moon is made of rock. Note that the second person may be aware of what the first person believes, along with what he personally believes.

We want to be able to reason about the goals and beliefs of another intelligence and these are intensional concepts. We need to do this because our system is simulating the planning process of another intelligence. In other words we want to

1. represent the beliefs, goals and actions of another intelligence, and
2. reason about the beliefs, goals and actions of another intelligence

The first is simple and has been done by Cohen [79] using the concept of a *belief space*. A *belief space* consists of all the things believed by an intelligence. It is important to realize that inferences are not valid if they are done across a belief space. If both of us know A but only I know that A implies B then I can not conclude that you have inferred B^{*} only that I have inferred B.

The second part is much more difficult and we have not accomplished this in a general way. We restricted ourselves to reasoning about someone having the knowledge necessary to perform a given task.

More elegant approaches use *modal logic*. A *modal logic* depends on the idea of *possible worlds* which has been defined by Kripke [71]. A *possible world* is a set of propositions that might be the case, that is they are consistent with what an intelligence believes. A proposition is *possible* if it is true in some possible world and *necessary* if it is true in all possible worlds. A world is possible only in relation to other possible worlds. Possible worlds are related to other possible worlds by an accessibility relation.

Hutikka [62] developed a modal logic with similar semantics to Kripke but with intensional concepts represented by modal operators. For example one could consider believing as being the same as the modal operator necessity in the sense that if you believe something then the belief must hold in every possible world. Using these ideas Moore [74] created an approach for reasoning about knowledge and action. The key steps in this method are to

- define the modal operators in terms of accessibility relations,
- axiomatize the properties of these accessibility relations, and
- reason whether these relations hold between possible worlds.

There are meta language predicates that talk about possible worlds and object language predicates that talk about a particular world, and axioms that relate the two.

Moore extended these meta language predicates to include meta language relations between worlds like $R(A, W1, W2)$. This relation is true if and only if $W2$ is a world resulting from the performance of action A in world $W1$. This allows reasoning about actions that produce knowledge by restricting the possible worlds which are compatible with the intelligences' knowledge after the execution of the action.

Appelt [80] applied the ideas of Moore to create a planner called KAMP (Knowledge and Modalities Planner) for the generation of natural language utterances to satisfy multiple goals. The kinds of examples that can be handled by KAMP are similar to those that can be handled by our system. The differences between the two are:

- KAMP's reasoning about intensional concepts is done using Moore's logic while ours is done in an ad-hoc manner. The axioms required for the modal logic approach are complex but the approach can be applied in many situations with intensional concepts. Response, however, is far removed from real time. Additionally, in the task environment our approach seems sufficient and is simpler to implement and to understand;
- KAMP optimizes the speech acts that it produces to handle multiple goals while our system does not do any such optimizations, and
- KAMP considers only language generation while our system also considers language understanding.

2.5 Speech Acts

2.5.1 Definition

Speech acts are considered to have been founded by Austin [62] and expanded by Searle [75]. The basic idea here is that human communication is considered from the point of view of its underlying goals. These goals are achieved by the performance of a speech act whose execution produces an utterance and changes the beliefs of the hearer and the speaker. A speech act can be decomposed into its propositional part and illocutionary

force The illocutionary force is the type of the speech act and the propositional part is the information that the speech act contains. Searle classified the illocutionary force into the following categories (Speaker is S, Hearer is H, Proposition P)².

<i>Representative</i>	commits S to the truth of P. "The Emperor has no clothes on"
<i>Directive</i>	S attempts to get H to do something. "Begin the parade!"
<i>Commissive</i>	commits S to a future course of action "I will march through the city!"
<i>Expressive</i>	expresses psychological state of S regarding P. "Thank you for your clever work"
<i>Declaration</i>	creates the condition expressed in P. "You are sentenced to 1 year in jail"

In order for a speech act to have a particular illocutionary force when performed it must meet certain conditions, sometimes called felicity conditions. For example if I inform you of some proposition P then I must believe P and I must believe that you do not know P or else the statement does not have the illocutionary force of a representative speech act. There is no simple, direct relationship between illocutionary force and the syntax of sentences. One reason is that some statements have an illocutionary force which is not the obvious choice, these are called indirect speech acts.

An example is the statement "you are standing on my foot" which has the direct illocutionary force of a representative but has the indirect illocutionary force of a directive which we would phrase as "get off my foot". The key point is that *reasoning* is used along with *knowledge of the situation* to determine any indirect illocutionary force.

Speech acts have been used in modelling natural language processing using computational models (programs) because they can be implemented with standard computer science concepts. The relationship is defined in the following table³.

² See Mann P. 6

³ See Mann P. 5

```

(Action-Description
  Knowledge System (Inform Person ?Belief) (Preconditions)
  (Body) (Restrictions (Believe System ?Belief))
  (Additions (Believe Person ?New-Belief)) (Deletions))

```

A speech action is used to model human communication. The execution of the Inform action results in an utterance and in the modification of the beliefs of the speaker and the hearer. In the above speech action the system informs the person of some ?Belief and records this by changing what it believes the person believes. In this way the system can reason about what the person needs to know since it keeps track of the person's beliefs.

A BELIEVE clause of type Requirement states the requirements on an action and has the following form:

```
(BELIEVE ?Intelligence (Requirement ?Action-Name ?Requirement) ?Justification)
```

The meaning of each variable is:

?Action-Name	The name of the action.
?Requirement	A list of requirements where each requirement has the form (?Characteristics ?Acceptable-Objects). This means that for this action, only the named objects are acceptable as objects that fulfill the given characteristic requirements.

For example the clause

```

((BELIEVE
  Person
  (Requirement Tighten-Generator-Bolts ((Wrench) (Socket-Wrench)))))

```

means that when tightening generator bolts the person should use a socket wrench instead of just any wrench. The fact that the socket wrench should be used is not mentioned explicitly in the action because the person might know the general action but not the requirement for a specific situation. Having a requirement clause allows the system to model this condition.

A BELIEVE clause of type reversable tells whether an action can be easily reversed and has the following form:

- First, the semantic form is (ASK System (Wrench Location)) because of certain syntactic rules. A typical rule would be that a statement that starts with "Where" has a semantic form of type ASK.
- Second, the reasoning would be "This is a sensible request because he does not know the location of the wrench, the wrench is necessary to achieve the current goal, he knows that I know where the wrench is and that I want to be helpful" so the speech act is (REQUEST System (Inform Person (Wrench Location)))

The system would then respond by informing the person of the wrench location.

The use of speech acts transforms the question "Does the system really understand an utterance" to "Are the system's beliefs modified in the fashion that the speaker expected". There is no way for the speaker to prove this, it can only be tested experimentally by the speaker interrogating the hearer with regard to the hearer's beliefs before and after the utterance.

2.5.2 Implemented Systems

Allen [80] uses a plan based approach for speech act recognition. The basic idea is as follows⁴:

"Identifying the speech act becomes part of the plan deduction problem. Thus, speech acts are identified from the communicated intentions of the speaker: the plan that he (the speaker) wants the hearer to recognize as the motivation behind his utterance."

This method is used in our system to process indirect speech acts. Allen has implemented a simple example for the environment of the train station. Consider the case where a person asks the ticket attendant "Can you tell me when the train goes to Windsor" and the attendant answers "At three o'clock on track five". The attendant gives extra information about which track the train leaves from because the attendant has recognized the

⁴ see Allen, P 10

person's goal of going to London and wants to be helpful. The attendant has also recognized this utterance as a question to which he should answer more than a simple "Yes." If the attendant said "No" the person would react quite differently than if a random person in the station said "No" to the same question. The reason is the person believes the attendant has this information and is not being cooperative if the information is not given. This shows that what an utterance means depends on the beliefs of the participants in the dialogue.

4² Very few other systems have been created that use speech acts. Sidner and Israel [81] have a system that determines the speaker's intended meaning in the context of a natural language interface for a graphics display facility. Along the same lines are Carberry [83], who uses a model very similar to Allen's to track user goals in an information seeking environment. An example of such an environment is the situation where a person is interacting with a data base system to try and achieve some goal but has not explicitly stated the goal. For some convincing arguments as to why natural language systems of the future should have such capabilities see [Grosz, 82] and [Perrault & Cohen & Allen, 82]. The degree of discrimination that a system must have depends on the versatility of the system. If the system can only tell the user something and not have the user tell it anything then there is little reason for reasoning about speech acts. The more versatile the system, the more it must be able to discriminate between the different types of speech acts of the input utterances.

2.6 Natural Language Processing

The natural language field had a great deal of early interest, a dry period and a recent revival along with the entire field of artificial intelligence. For a good overview of the field see [Waltz, 82] and [Hendrix & Sacerdoti, 81]. The latter classify natural language systems into the following three types

- Type A* these are systems that do not use explicit world models (most systems that are used as front ends for data bases are examples)
- Type B* these are systems that use world models but do not keep track of the goals and beliefs of the user (TDUS and UC are examples).
- Type C* these are systems that model beliefs, and goals of the user (PAM, Possibly UC and our system are examples).

We claim that our system is truly a type C system but for a very specialized application. The reason for creating type C systems is as Waltz [82] says.

"Natural language systems must not simply understand the shallow surface meaning of language, but must understand the deeper implications and inferences that a user is likely to take from language. In order to do this, the systems must be capable of understanding user goals, intentions, and strategies, as well as the multiple purposes served by any piece of language."

If we can not create a type C system for a task environment we will certainly not be able to create a type C system for more complex environments. We will now survey some natural language systems and compare them to our system.

2.6.1 SHRDLU

This was one of the first completely integrated systems [Winograd, 82] for understanding natural language. We mean by completely integrated that it combined syntactic and semantic analysis with a body of world knowledge about a limited domain. The domain was the simple, closed domain of the blocks world. An example of the kind of conversation that the system could engage in follows.

Person Pick up the red block

System OK

Person Grasp the pyramid

System I don't understand which pyramid you mean

Person Find a block which is taller than the one you are holding
and put it in the box

System By "it", I assume you mean the block which is taller than

the one I am holding.

The differences between SHRLDU and our system are that:

- ours infers user goals;
- ours models the beliefs of the user and simulates his planning;
- ours does general common sense reasoning;
- ours imagines future situations and remembers past situations, and
- ours *separates* the knowledge used from the reasoning processes and is thus extendible to a *class* of applications

The approach taken by SHRLDU was abandoned because it was considered to be non-portable and restricted to the blocks world. We feel that this was premature and partially due to the inability in SHRLDU to separate the knowledge from the reasoning which made it very difficult to extend either one of them. We also feel that specialized areas like the task domain are environments simple enough to apply modern methods (speech acts, common sense reasoning, etc) to create a type C system, but still small enough to hope for some practical success.

2.6.2 TDUS

The Task Oriented Dialogue Understanding System was created by [Grosz, 80] and has similarities to our system since it works in the task environment. It uses procedural networks [Sacerdoti, 77] to store the task information and this information is used as a context for the dialogue. TDUS does not use the closed world assumption and can engage in fairly complicated dialogues.

The emphasis is on using the context to understand ambiguous parts of the dialogue. TDUS does not infer user goals nor does it model what the user believes. Therefore the system does not handle dialogues like:

Person: What's a ratchet wrench?

System: A wrench with a socket at the end. It's on the table.

This dialogue only makes sense in terms of the inferred goal of the person, which is to get the ratchet wrench. Our system can handle such dialogues because it infers user goals and models user beliefs.

2.6.3 PAM

The Plan Applier Mechanism is an early program by Wilensky [78] that tries to understand a story by determining the goals of the characters and then relating the events in the story to these goals. For example, consider the following story:

"Willie was hungry. She picked up the Michelin guide and got into her car."

If PAM was asked the question "Why did Willie get into her car?" it would respond "because Willie wanted to go to a restaurant". The answer (which is the obvious one) is given by PAM because it infers that Willie has the goal of satisfying her hunger and that getting into her car is part of a plan for achieving this goal by going to a restaurant. PAM does the following three things:

- First it infers user goals by using general rules that relate statements or events to goals. A typical rule would be "if someone is in a state and it is not a desirable state then they have a goal of changing this state". This rule allows PAM to infer that Willie being hungry means she has a goal of satisfying this hunger.
- Second it uses these goals to predict future events. If Willie has a goal of satisfying her hunger then PAM would predict events like going to the refrigerator or going to the restaurant.
- Third it fits actual events into predicted events. This would interpret Willie getting into her car as a predicted event in Willie's plan of going to the restaurant.

This process is called explanation driven understanding because it tries to understand a story by finding an explanation for every event in the story. The idea has limited applicability in stories but we use this idea in the task environment when we try to infer user goals from a single action.

PAM was one of the first programs that modelled understanding and reasoning as an explanation driven process.

2.6.4 UC.

The Unix Consultant [Wilensky, 84] is an intelligent natural language interface that allows naive users to ask questions about the Unix Operating System. The system has the following capabilities⁵:

- a robust language analyzer which almost never has a hard failure;
- a context and memory mechanism that determines the focus of attention;
- an experimental planning component that provides common sense reasoning in creating plans for the user, and
- a goal mechanism which tries to infer the user goals from the speech act that the input represents.

The ability to infer user goals and reply appropriately make this system different from more traditional data base front ends. An example of a conversation with UC is:

Person: Do you know how to list a file?

System: Typing "pr filename" will print the file with the name filename on your terminal.

The kind of conversation that should not take place (it is not clear from the article whether the system can handle this or not) is:

⁵ see Wilensky, 84, P.574

Person: I'm trying to get some more disk space.

System: Type "rm *".

The reason this is a bad answer is that executing this rm command would remove all of the user's files and thus it violates "common sense".

In order to analyze the situation the system must:

- Infer user goals, and
- Use common sense reasoning to create a sensible answer.

UC does the first by a goal analysis module which uses general rules in a way similar to PAM to infer the goals of the user. UC does the second by using a planner called PANDORA to create plans for solving the user's problem that do not violate any common sense restrictions. We want to avoid creating a system that takes everything literally but can use its knowledge of the situation and reasoning ability to produce sensible answers.

The context mechanism of UC is used to provide an environment which defines the current focus for the user's questions. In our system the task being undertaken provides an implicit focus. Another difference between UC and our system is that UC does not keep track of the user's beliefs. Our system can be used in a problem solving mode which is similar to the way current expert systems are used. For this reason we must keep track of our environment in a more thorough manner than UC which is simply an information providing system.

UC is an important system because it is the first to integrate the ideas of common sense reasoning and goal inference in a *practical* natural language interface. In his most recent article on UC Wilensky [84] concludes that analyzing user goals and having an explicit user model are the two most important areas of further research. We agree, and our system has both these characteristics though the environment is simpler than UC.

2.7 PROLOG

A logic program is a finite set of definite clauses of the form

$$A \leftarrow B_1, B_2, \dots, B_k \quad \text{with } k \geq 0$$

where the A 's and B 's are logical atoms, also called goals. The sentence reads " A is implied by the conjunction of the B 's" and is interpreted procedurally as "to satisfy goal A , satisfy goals $B_1, B_2, \dots, B_k$ ". A is called the clause's head and the B 's are the clause's body. If the B 's are missing the sentence reads " A is true" or "goal A is satisfied". Given a goal, or a conjunction of goals, a set of clauses can be executed as a program using this procedural interpretation.

A logic program is executed via non deterministic goal reduction. PROLOG, however is an implementation of logic programming where the execution method is sequential selection of clauses with backtracking on failure. It is important to realize that the relationship between PROLOG and Logic Programming is similar to the relationship between LISP and the Lambda Calculus. For a good introduction to PROLOG see Clocksin and Mellish [82].

We feel that the best use of PROLOG is to create *prototypes*. A *prototype* should demonstrate the *essential* ideas for solving a problem and makes it possible to test these ideas quickly. If successful it leads to a larger implementation in a more traditional language.

One problem with PROLOG as an implementation of Logic Programming is that the declarative semantics are essentially destroyed by the use of the cut. When a clause has a cut symbol (usually shown as !) the procedural semantics (backtracking) are affected without changing the declarative semantics of the program.

Another problem is the use of backtracking as the only control structure. A good analogy is to think of backtracking as fishing with a net; it is easy to catch something

and also easy to catch something that you don't want. We could have implemented the reasoning part of our system in LISP or PROLOG. PROLOG does force the use of predicates in the knowledge representation. This is not a problem since more complex schemes such as semantic networks can be implemented using predicates.

Fundamental Concepts

This chapter describes the basic concepts used in our system. It starts with a description of the PROLOG implementation which has been modified to represent possible worlds efficiently. The knowledge representation for beliefs and actions is described, along with how this knowledge is used during planning.

3.1 Possible Worlds Prolog

We are using an implementation of PROLOG which is similar to MICRO-PROLOG [McCabe & Clark, 80] but with some extensions to make the representation of possible worlds more efficient. In standard PROLOG a clause has the form

$$A :- B_1, B_2, \dots, B_n$$

where the A and the B 's are themselves clauses. The following example defines a list membership clause.

```
MEMBER (H, [H | _]) .
MEMBER (I, [_ | T]) :- MEMBER (I, T).
```

A list is represented by $[ ]$, the list constructor is $[_]$ and an unnamed variable is $_$. In MICRO-PROLOG a clause is represented as a list so the same example becomes

```
((MEMBER ?H (?H ?*))
((MEMBER ?I (?* ?T)) (MEMBER ?I ?T)))
```

The fact that a clause is represented as a list makes it possible to create a PROLOG clause that has a capability similar to LISP macros. An example is the following definition of an OR which uses this capability

((OR ?X ?Y) ?X)

((OR ?X ?Y) ?Y)

The variables ?X and ?Y become instantiated to a list of the form $((Clause_1) \dots (Clause_n))$ which becomes the rest of the OR clause. This is simpler than standard PROLOG which distinguishes clauses from lists since it must convert between the two to accomplish this task. Other differences between our PROLOG and standard PROLOG is that our symbol for the list constructor is " " instead of "[", our symbol for an unnamed variable is "?" instead of "_" and our variables all start with "?" instead of an upper case character.

The following notation is used for listing clauses.

- ¶ Clause heads are in upper case and all arguments are capitalized. For example in the clause (LIGHT On) the head is LIGHT and the argument is On.
- ¶ The individual clauses of a PROLOG statement are listed below the first clause in line with that clause. That is $A - B_1, B_2, \dots, B_n$ becomes.

((A)
(B₁)
(B₂)

(B_n))

- ¶ The arguments of a clause are listed on the same line as that clause in the form

(HEAD ARG₁ ARG₂ ... ARG_n)

If the arguments do not fit on one line they are listed on successive lines in the form

$$\begin{aligned}
 &(\text{HEAD} \\
 &\quad \text{ARG}_1 \dots \text{ARG}_i \\
 &\quad \text{ARG}_{i+1} \dots \text{ARG}_j \\
 &\quad \text{ARG}_{j+1} \dots \text{ARG}_k \\
 &\quad \dots \\
 &\quad \text{ARG}_{m+1} \dots \text{ARG}_n)
 \end{aligned}$$

If any of the arguments are lists this rule is invoked recursively with the first element of the list replacing the head of the clause.

The main problem is how to represent the possible worlds efficiently. Possible worlds are necessary if the system is to be able to "imagine" the future. A simple approach would require the additions and deletions that created each world to be explicitly associated with that world. This would have the following requirements:

- Space* Each fact would require at most a fixed number of bits to represent it and we assume that each action would have a fixed maximum number of facts that could be added or deleted. Therefore the space required to represent all the facts is of order N where N is the number of possible worlds.
- Time* A lookup requires a search through order N predicates since we must examine each world to see if the fact was deleted. Therefore the time required to lookup a fact is order N , where N is the number of possible worlds.

An alternative approach which is of the same order but is considerably more efficient because the constants are smaller is to use contexts as described in [Mcdermott, 1983]. A world contains a set of clauses. In this approach each world has a unique characteristic number which is a natural number labelled W_n for world number n . A world is described by this characteristic number, and the set of characteristic numbers of the worlds from which it inherits clauses. Since worlds are connected in a tree we know that for any world the set of characteristic numbers of the worlds from which it inherits is simply the characteristic number of all the worlds in the path from this world to the root of the tree. The form of this definition for world number i is $\langle W_i, \{W_{k1}, W_{k2}, \dots, W_{kn}\} \rangle$ where the set $\{W_{k1} \dots W_{kn}\}$ is the characteristic numbers of the worlds from which this world inherits clauses.

Every clause has an associated label which is a disjunction of conjunctions where each conjunction has the form $W_{i1} \wedge \neg W_{i2} \wedge \dots \wedge \neg W_{in}$. The label of a clause can be thought of as a function, which given the definition of a world as an argument returns true if this clause

is in that world, and false otherwise. The function takes each disjunction in the label of the clause and checks if the characteristic set of the world *satisfies* that disjunction. By *satisfies* we mean the following:

1. Take the first characteristic number of the conjunctions that make up the disjunction (those not negated). It must be a member of the characteristic set of the world.
2. Take each of the remaining characteristic numbers of the conjunction that makes up the disjunction (they are all negated). Each one must not be a member of the characteristic set of the world.

If any of the disjunctions are satisfied the clause is in that world, if none of the disjunctions are satisfied the clause is not in that world.

To add a clause to a world we must change the label L of that clause to $L \vee W_j$, where W_j is the characteristic number of the world. To delete a clause from a world we must change the label L of the clause to $L \wedge \neg W_j$, where W_j is the characteristic number of that world. This distributes the negation across each disjunction.

The order of this approach is still the same as the simple approach but the constants are less. If the sets are represented as a bit string then the membership operations become bit manipulation operations, which are very efficient. If there are N possible worlds for each clause we require at most N bits to represent each disjunction and there are at most $N/2$ disjunctions. This is the worst case which occurs when a clause is alternately deleted and added to a chain of worlds.

Consider the following example with clause (LIGHT On). The characteristic numbers of the worlds are labelled W_i , for i from one to seven.

Figure 3.1-Plan Tree For One Clause

World	Characteristic Set	Clause Label For (LIGHT On)
W_1	W_1	W_1
W_2	W_2, W_1	W_1
W_3	W_3, W_2, W_1	$W_1 \wedge \neg W_3$
W_4	W_4, W_3, W_2, W_1	$W_1 \wedge \neg W_3$
W_5	W_5, W_3, W_2, W_1	$(W_1 \wedge \neg W_3) \vee W_5$
W_6	W_6, W_4, W_3, W_2, W_1	$(W_1 \wedge \neg W_3) \vee W_5 \vee W_6$
W_7	W_7, W_5, W_3, W_2, W_1	$(W_1 \wedge \neg W_3 \wedge \neg W_7) \vee (W_5 \wedge \neg W_7) \vee (W_6 \wedge \neg W_7)$

Table 3.1-Clause Labels For One Clause

The clause label for each world is the label that exists for the clause (LIGHT On) after the creation of that world. The table shows how the clause label changes as this clause is added or deleted from a world. The final label and the characteristic set of the world are used to check if the clause exists in that world.

For example, consider world W_6 . To see if the clause (LIGHT On) exists in this world each disjunction of the label must be checked to see if the characteristic set $\{W_6, W_4, W_3, W_2, W_1\}$ satisfies that disjunction. Since the characteristic set satisfies the disjunction $W_6 \wedge \neg W_7$, the clause (LIGHT On) exists in world W_6 . Similarly for world W_4 , the characteristic set $\{W_4, W_3, W_2, W_1\}$ does not satisfy any disjunction so the clause (LIGHT On) does not exist in that world.

This next example has actions and worlds and uses them in the way a planner would. Consider an initial world where the following clauses exist:

(AT-BOX C)

(AT-CHAIR B)

(AT-ROBOT A)

The following actions when executed create a new world with the appropriate clauses added and deleted:

ACTION-ONE (GOTO B) ADD (AT-ROBOT B) DEL (AT-ROBOT A)

ACTION-TWO (GOTO C) ADD (AT-ROBOT C) DEL (AT-ROBOT A)

ACTION-THREE (PUSH-CHAIR B A) ADD (AT-ROBOT A) (AT-CHAIR A)

DEL (AT-ROBOT B) (AT-CHAIR B)

ACTION-FOUR (PUSH-BOX C A) ADD (AT-ROBOT A) (AT-BOX A)

DEL (AT-ROBOT C) (AT-BOX C)

Figure 3.2-Plan Tree for Planning Example

Clause	Label
A0. (AT-BOX C)	$W_1 \wedge \neg W_5$
A1 (AT-CHAIR B)	$W_1 \wedge \neg W_4$
A2 (AT-ROBOT A)	$(W_1 \wedge \neg W_2 \wedge \neg W_3) \vee W_4 \vee W_5$
A3 (AT-CHAIR A)	W_4
A4 (AT-ROBOT B)	$W_2 \wedge \neg W_4$
A5 (AT-ROBOT C)	$W_3 \wedge \neg W_5$
A6 (AT-BOX A)	W_5

Table 3.2-Clause Labels for Planning Example

To create a possible world we perform an EXECUTE-ACTION clause and to move from world to world we perform a VIEWPOINT or JUMP-WORLD clause. The EXECUTE-ACTION and VIEWPOINT clauses are special because when the system backtracks over them it changes the current world to be the original world in which the clause was executed. The PROLOG interpreter thus undoes some of the affects of the action which enables the PROLOG program to create and move around in the plan tree.

3.2 Knowledge Representation

3.2.1 Beliefs

There are two intelligences in the environment. The first, which we call the system, is the program that is simulating the expert assistant. The second, which we call the person, is whoever is trying to accomplish the given task(s). The beliefs of the system are represented by clauses of the following form

((BELIEVE ?Intelligence ?Belief ?Justification) . ?Belief-Body))

where the variable ?Intelligence is System or Person, the variable ?Belief is any belief and the variable ?Justification is the reason the intelligence has that belief. We take (BELIEVE System ?Belief ?Justification) to mean that the system privately believes ?Belief justified by ?Justification and (BELIEVE Person ?Belief ?Justification) to mean that the system believes that the person believes ?Belief justified by ?Justification. The justification is the one that the system has for these beliefs since the system does not know what justifications the person may have for his beliefs.

We do not use the *closed world assumption*. The *closed world assumption* is that any belief that can not be shown to be true is assumed to be false. Instead, the clause

((GET-TRUTH-VALUE ?Intelligence ?Belief ?Truth-Value))

when given some instantiations for the variables ?Intelligence and ?Belief sets the variable ?Truth-Value to True, False, Known or Unknown. A belief is unknown if the question "Is the belief true or false?" can not be answered affirmatively. It is necessary to have the truth value of unknown to represent the idea of an intelligence lacking knowledge. A belief is known if the system does not know but believes that the person does know some belief. These ideas are discussed in greater detail in the section on mutual beliefs.

The ?Belief part of the belief actually has the form (?Belief-Type ?Belief-Parameters). The meaning of each variable is

```

(Execute-Action
  (Action-Description
    Person
    (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    (Preconditions) (Body)
    (Additions (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    (Deletions (Believe Person (Physical-Fact (?Fact-Head ?*))))
    (Restrictions) (Sensors))))

```

After it is executed the beliefs of the system are changed so that the person has the appropriate beliefs.

We will use the situation where a light can be on or off as an example to illustrate how the inform speech act is processed. The processing starts with the following clause:

```

((PROCESS-SPEECH-ACT (Believe Person ?Belief))
 (GET-TRUTH-VALUE Person ?Belief ?Truth-Value)
 (PROCESS-POSSIBLE-INFORM ?Belief ?Truth-Value))

```

If the person knows whether the light is on or off, and the system does not (in our knowledge representation this has a truth value of known) we have a valid inform.

```

((PROCESS-POSSIBLE-INFORM ?Belief Known)
 (WRITE (Ok))
 (PERSON-INFORM ?Belief))

```

If the person should not have known whether the light was on or off, this may still be a valid inform but further processing is required.

```

((PROCESS-POSSIBLE-INFORM ?Belief Unknown)
 (GET-TRUTH-VALUE SYSTEM ?Belief ?Truth-Value)
 (FURTHER-INFORM-PROCESSING ?Belief ?Truth-Value))

```

The further processing is that the system checks whether it believes that the light is on or off. If it agrees with what the person has told it, or it does not know, it still accepts this as a valid inform.

```

((FURTHER-INFORM-PROCESSING ?Belief Unknown)
 (WRITE (I Did Not Know That And Did Not Think That You Did))
 (PERSON-INFORM ?Belief))

```

((BELIEVE Person (Generator-Shakes) ?Justification)
 (BELIEVE Person (Physical-Fact (Generator-Belts Loose)) ?*))

The ?Justification becomes instantiated to the particular body that is used in the proof of the inferred belief.

An inferred belief has a truth value of:

True If all the beliefs in the belief body are true

False If any of the beliefs in the belief body are false

Unknown If any of the beliefs in the belief body are unknown and none of the beliefs in the belief body are false.

An inferred belief can not be added by an action. It is only used to infer things about the current situation based on fact beliefs.

3.2.2 Mutual Beliefs

There are two intelligences in our environment, the system, which is simulating the expert assistant, and the person, who is performing some task. The system keeps track of its private beliefs and what it believes the person believes. The system has knowledge about performing the task at hand and possibly about the states of some physical facts. The person may know something about how to perform the task but the knowledge is deficient in some way, or else there would be no need for an assistant. The person does know (or can find out) the state of many physical facts that are unknown to the system. There may be some physical facts whose state can not be determined by either the system or the person, but which must be determined from the outcome of actions.

The possibilities with regard to the beliefs are as follows:

- 1 The system knows something that the person does not. This is represented by having the belief present in the system's beliefs but absent from the person's beliefs. The result of executing the clause GET-TRUTH-VALUE of that belief from the point of view of the system would be a truth value of TRUE, and from the point of view of the person the same belief would have a truth value of Unknown.

- 2 The person knows something that the system does not and the system knows that the person knows it. This is represented by having the belief present in the person's beliefs but absent from the system's beliefs. The actual state of the physical fact belief (it must be a fact since that is all the person could ever know that the system does not) is left as a free variable. For example, if the system believes the person knows whether the light is on or off the representation is the clause ((BELIEVE Person (Light ?*) Premise)). The result of executing the clause GET-TRUTH-VALUE for that belief from the point of view of the system or the person would be a truth value of Known.
- 3 The system and the person both have no idea of the state of some physical fact. This is represented by the belief being absent from both their private beliefs. The result of executing the clause GET-TRUTH-VALUE for that belief from the point of view of the system or the person would be a truth value of Unknown.
- 4 The system and the person disagree, that is they hold contradictory beliefs. Once detected this state will not last because we *assume* that the system and person *will eventually agree*. Any disagreement is temporary and will be resolved by the system or the person changing their beliefs until there is agreement.
- 5 The system and the person both have a belief and agree on it, which is the normal situation. Instead of duplicating things that are mutually believed by both the person and the system we include the equivalent of the clause:

((BELIEVE System ?Belief ?Justification)
 (BELIEVE Person ?Belief ?Justification))

which says that the system believes everything that it believes the person believes.

With this representation of mutual belief it is easy for the system to reason about what the person needs to *know* in order to perform some task, and to check whether a goal makes *sense* when considered from the context of these beliefs.

3.2.3 Knowledge

We define the concept of knowing by the clause:

```
((KNOW ?Intelligence ?Belief)
 (NOT GET-TRUTH-VALUE ?Intelligence ?Belief Unknown))
```

One has knowledge if one can say that the belief is true, false or known. Note that for physical facts to have knowledge of the current state implies that we also believe that the alternate states are not true since an obligation can be fulfilled in only one way. An example would be if we know that the light is on we also know that it is not off. Using obligations for physical facts allows us to answer some of the questions that would normally require explicit representation of negations.

Having knowledge about a situation is an important part of the planning and reasoning process and will be dealt with in more detail in the section on actions and planning.

3.2.4 World

The task environment contains physical objects which are manipulated by the person using other physical objects (tools). It is not surprising that the most important fact belief is:

```
((BELIEVE ?Intelligence (Physical-Fact (?Name ?Fact-State)) ?Justification))
```

The meaning of each variable is:

?Name   This is the name (or type) of the physical fact belief.

?Fact-State   This is the current state of that fact belief.

A physical object has a number of attributes such as size, location, etc. Each type of physical fact defines an attribute of some object. For example, the physical fact with ?Name of Socket-Wrench-Location defines the location attribute of the socket wrench.

Each type of Physical-Fact has an associated obligation clause of the form ((OBLIGATION ?Name . ?Possible-States)). The obligation means that ?Fact-State must be a member of ?Possible-States for that ?Type and that these states are mutually exclusive.

For example ((OBLIGATION Light On Off)) means that the following possibilities exist:

- ((BELIEVE ?Intelligence (Physical-Fact (Light On)) ?Justification)) which means that the ?Intelligence believes that the light is on.
- ((BELIEVE ?Intelligence (Physical-Fact (Light Off)) ?Justification)) which means that the ?Intelligence believes that the light is off.
- The other alternative is that there is no belief of the form ((BELIEVE ?Intelligence (Physical-Fact (Light ?State)) ?Justification)) which means that the ?Intelligence has no idea whether the light is on or off.

We assume that all the intelligences (system and person) believe the OBLIGATION clause and are aware that there is a light and that it must be in one of the possible states, even if the current state is not known. Therefore, if a physical fact clause has the truth value UNKNOWN it means that the intelligence does not know the state of the associated physical obligation.

A physical object has a description the form:

(Description ?Object ?Description)

The meaning of each variable is:

?Object	The name of the object.
?Description	An english like description of the object.

An example of a description clause is.

```
((BELIEVE
  System
  (Description Socket-Wrench (A Wrench With A Socket At The End)
  Premise)))
```

In summary the environment, which we call a world, has objects with attributes. The state of these attributes is defined by a Physical-Fact clause and these objects are described by a Description clause.

3.3 Actions

3.3.1 Action Definition

An initial world is given to the system as a premise, and future worlds are created by executing actions. An action, when executed, transforms one world to another world by adding or deleting clauses. One can think of the worlds as nodes, the actions as arcs, and a plan as a tree of worlds connected by actions with a root node which is the initial world. The action arc would be directed towards the world created by executing the action.

A BELIEVE clause of type Actions-For-Obligation lists all the actions that could affect a particular obligation. Such clauses have the form:

```
(BELIEVE
  ?Intelligence
  (Actions-For-Obligation
    ?Obligation ?Type ?Action ?Action-Descriptions ?Justification))
```

If the clause is present with an instantiation for the variable ?Action-Descriptions then the intelligence believes that these actions can be used to modify the state of that obligation. There can be a number BELIEVE clauses of type Actions-For-Obligation for each obligation. The reason is that the person may be aware of some, but not all of the ways to accomplish something and the system must be able to represent this situation.

3.3.2 Action Description

The form of each Action-Description contained in the ?Action-Descriptions of the BELIEVE clause of type Actions-For-Obligation is as follows.

```
(Action-Description
  ?Class ?Actor ?Short-Description ?Preconditions
  ?Body ?Restrictions ?Deletions)
```

This is the standard STRIPS format with some extra fields. The meaning of each variable is as follows:

- ¶ The variable ?Class is the type of action, Physical or Knowledge. A physical action is one that physically changes the world along with what we believe about the world. It can add and delete only fact beliefs of type physical fact. A knowledge action is one that does not physically change the world but changes what we believe about the world. It can add and delete any kind of fact belief.
- ¶ The variable ?Actor is the intelligence that can perform the action, which is either System or Person.
- ¶ The variable ?Short-Description is the action description in the form (?Action-Name . ?Action-Parameters) where ?Action-Name is the name of the action and ?Action-Parameters is a list of the variables that are used in the action description. The variables can be thought of as the parameters of the action. The description of the action completely defines the action if the parameters are instantiated.
- ¶ The variable ?Preconditions is the list of preconditions clauses which are beliefs that must be true in the world before the execution of the body part can be attempted. In certain situations we want to specify that an object have certain characteristics without naming the object explicitly. For this reason the type of physical fact for a precondition may have the form (?Characteristics ?Attribute). This means that any object that has the proper characteristics is acceptable. There is an ISA hierarchy that relates objects to characteristics with the objects at the bottom of the hierarchy.

- ¶ The variable ?Body is either (Serial-And ?Actions) or (Code ?Body). In the first form ?Actions is a list of short descriptions of actions that must be successfully executed in the order given for the containing action to be executed. In the second form ?Body is any PROLOG clause that must be executed for the containing action to be executed.
- ¶ The variable ?Restrictions is a list of clauses that must be true in order for this action to be executed. These clauses are said to restrict the execution of the action because if they are not true the action will fail. They are different from preconditions because they are not goals that must be satisfied if they are not true because there are no actions that can change the state of the associated obligation (or if there were they would certainly not be used).
- ¶ The variable ?Additions is a list of clauses that are added to the old world, to make the new world if the action is successful.
- ¶ The variable ?Deletions is a list of clauses that are deleted from the old world to make the new world if the action is successful.

An example of an action description of a physical action is

```
(Action-Description Physical Person (Move-Self ?From ?To)
 (Preconditions (Believe Person (Physical-Fact (At-Person ?From)))
 (Body)
 (Restrictions (Not Eq ?From ?To))
 (Additions (Believe Person (Physical-Fact (At-Person ?To))))
 (Deletions (Believe Person (Physical-Fact (At-Person ?From)))))
```

The restriction field for this action stops the planner from using this action as a useless no-op of the form (Move-Self ?Place ?Place). Another example of a physical action is

```
(Action-Description
 Physical Person (Install-Generator) (Preconditions)
 (Body
  Serial-And
  ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws))
 (Restrictions (Believe Person (Physical-Fact (Generator Not-Installed))))
 (Additions (Believe Person (Physical-Fact (Generator Installed))))
 (Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))
```

The body field for this action contains the steps necessary for the generator to be installed successfully. It contains the skeleton of a plan that can be used to execute the action.

Without a body field more searching would be necessary during planning because this information would have to be put into the preconditions of the action.

An example of an action description of a knowledge action is

```
(Action-Description
 Knowledge System (Assume-System At-Wrench Corner)
 (Preconditions) (Body) (Restrictions)
 (Additions (Believe System (Physical-Fact (At-Wrench Corner)))) (Deletions))
```

This action is an assumption action, which means that when necessary the system is prepared to assume that the wrench is in the corner. There may be other assumptions about the wrench location that the system can make. Assumptions are made when the knowledge is required and the state of the associated obligation is unknown. We restrict assumptions to adding only fact beliefs of type physical fact.

Another example of knowledge action is the following clause:

```
(Action-Description
 Knowledge Person (Check Power On)
 (Preconditions
 (Believe Person (Physical-Fact (At-Power-Cable ?Place)))
 (Believe Person (Physical-Fact (At-Person ?Place)))
 (Believe Person (Physical-Fact (At-Voltmeter ?Place)))
 (Body) (Restrictions)
 (Additions (Believe Person (Physical-Fact (Power On))))
 (Deletions (Believe Person (Physical-Fact (Power ?*))))))
```

```
(Action-Description
 Knowledge Person (Check Power Off)
 (Preconditions
 (Believe Person (Physical-Fact (At-Power-Cable ?Place)))
 (Believe Person (Physical-Fact (At-Person ?Place)))
 (Believe Person (Physical-Fact (At-Voltmeter ?Place)))
 (Body) (Restrictions)
 (Additions (Believe Person (Physical-Fact (Power Off))))
 (Deletions (Believe Person (Physical-Fact (Power ?*))))))
```

These actions would be performed to check whether the power is on or off at the end of a power cable. These are knowledge actions because in executing them the person has not changed the physical world, only the beliefs about the physical world.

A final example of a knowledge action is a speech action which is

```

(Action-Description
  Knowledge System (Inform Person ?Belief) (Preconditions)
  (Body) (Restrictions (Believe System ?Belief))
  (Additions (Believe Person ?New-Belief)) (Deletions))

```

A speech action is used to model human communication. The execution of the Inform action results in an utterance and in the modification of the beliefs of the speaker and the hearer. In the above speech action the system informs the person of some ?Belief and records this by changing what it believes the person believes. In this way the system can reason about what the person needs to know since it keeps track of the person's beliefs.

A BELIEVE clause of type Requirement states the requirements on an action and has the following form:

```
(BELIEVE ?Intelligence (Requirement ?Action-Name ?Requirement) ?Justification)
```

The meaning of each variable is:

?Action-Name	The name of the action.
?Requirement	A list of requirements where each requirement has the form (?Characteristics ?Acceptable-Objects). This means that for this action, only the named objects are acceptable as objects that fulfill the given characteristic requirements.

For example the clause:

```

((BELIEVE
  Person
  (Requirement Tighten-Generator-Bolts ((Wrench) (Socket-Wrench)))))

```

means that when tightening generator bolts the person should use a socket wrench instead of just any wrench. The fact that the socket wrench should be used is not mentioned explicitly in the action because the person might know the general action but not the requirement for a specific situation. Having a requirement clause allows the system to model this condition.

A BELIEVE clause of type reversable tells whether an action can be easily reversed and has the following form:

((BELIEVE ?Intelligence (Reversible ?Action-Description ?Flag) ?Justification))

The meaning of each variable is:

?Action-Description	The name of the action.
?Flag	Either Yes or No. If it is Yes then the action can be reversed, if No it can not be reversed.

The reason for having a Reversible clause is the same reason as that for having a Requirement clause. That is we want to represent the situation where the person knows the basic action but does not know whether the action is reversible. This information would be important because it warns the person to be careful to perform this action in the proper sequence.

3.4 Plans

3.4.1 Plan Definition

Planning is the process of creating a plan tree, which we define as a tree of worlds connected by actions, to achieve a goal. Our planner is based on the usual backward chaining method described in STRIPS but is able to handle the simple intensional idea of knowing something. This is why our planner produces a tree of worlds with each branch corresponding to a different state of some unknown.

A goal is a clause of the following form:

(GOAL ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)

The meaning of each variable is:

?Type	This is either Physical or Knowledge. A goal of type Physical is the kind of goal that normal planners try to achieve. This is done by producing a plan containing only physical actions that when executed modify the physical world as well as the beliefs held about the physical world. The result of executing the plan is a world where the desired ?Belief is true.

A goal of type Knowledge by contrast, is achieved when the question "Is the belief true or false?" can be answered. This is done by creating a plan containing knowledge actions, that when executed

modify the beliefs held about the world. The result of executing such a plan is a world where the above question can be answered

?Anyone   This is either Person or System
 ?Belief   If the goal is of type Knowledge this is a fact belief or an inferred belief. If the goal is of type Physical this is a fact belief
 ?Justification   This is the reason for the goal in some descriptive form

3.4.2 Planning Method

We will describe the planning from the point of view of the person. By this we mean that we will consider what planning methods the person would use to achieve a particular goal. We ignore the system and assume that the person has all the knowledge necessary to achieve this goal. In the §4.4 we will embellish this planning process by showing how the system would perform cooperative behavior.

The basic planning method for the person starts with the clause

```
((PLAN-FOR-GOAL ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)
 (GET-TRUTH-VALUE ?Anyone ?Belief ?Current-Truth-Value ?Justification)
 (ACHIEVE-GOAL ?Type ?Anyone ?Belief ?Current-Truth-Value ?Justification))
```

To achieve a goal we use an ACHIEVE-GOAL clause which has a number of cases

```
((ACHIEVE-GOAL ?Type ?Anyone ?Belief True ?Justification))
```

The goal has already been achieved if the belief is true so nothing more need be done

```
((ACHIEVE-GOAL Knowledge ?Anyone ?Belief False ?Justification))
```

A knowledge goal is achieved even if the belief is False because the clause (KNOW Person ?Belief) is satisfied

```
((ACHIEVE-GOAL
  ?Type ?Anyone
  (Physical-Fact (?Fact-Head ?Fact-Rest)) ?Truth-Value ?Justification)
 (OR
  (EQ (?Type ?Truth-Value) (Physical False))
  (EQ (?Type ?Truth-Value) (Knowledge Unknown)))
 (PLAN-FOR-GOAL
  Knowledge ?Anyone
  (Believe
    ?Anyone
    (Actions-For-Obligation ?Fact-Head ?Type ?*) (Know-Action ?Fact-Head)))
```

```

(GET-TRUTH-VALUE
  'Anyone
  (Actions-For-Obligation 'Fact-Head 'Type 'Action-Descriptions) True)
(EVERY-MEMBER
  (Action-Description
    'Type 'Anyone 'D 'P 'B 'R
    (Additions
      (Believe
        'Anyone
        (Physical-Fact ('Fact-Head 'Fact-Rest))) 'Rest-Additions)
      'Rest-Action)
    'Action-Descriptions))
(TRY-EXECUTE-ACTION
  (Action-Description
    'Type 'Anyone 'D 'P 'B 'R
    (Additions
      (Believe
        'Anyone
        (Physical-Fact ('Fact-Head 'Fact-Rest))) 'Rest-Additions)
      'Rest-Action)))

```

This clause handles two situations. The first is when the person wants to achieve a physical goal which is a physical fact belief that is false, the second is when the person wants to achieve a knowledge goal which is a physical fact belief that is currently unknown. For both of these the person needs to find the actions for the appropriate obligation and then to execute at least one of these actions. In trying to execute the action, the clause TRY-EXECUTE-ACTION will invoke the clause

```
((PLAN-FOR-GOALS 'Type 'Anyone 'Preconditions (Want-Preconditions 'Action))
```

This invokes the goal clause for each member of the preconditions list thus recursively invoking the planner

```

((ACHIEVE-GOAL
  Physical 'Anyone (Physical-Fact ('Fact-Head 'Fact-Rest)) Unknown 'Justification)
  (PLAN-FOR-GOAL
    Knowledge 'Anyone
    (Believe 'Anyone (Physical-Fact ('Fact-Head 'Fact-Rest))) (Know-State 'Fact-Head))
  (PLAN-FOR-GOAL
    Physical 'Anyone
    (Believe 'Anyone (Physical-Fact ('Fact-Head 'Fact-Rest))) 'Justification))

```

This clause handles the situation where the person wants to achieve a physical goal for a belief which is currently unknown. This forces the person to first have a knowledge goal

of knowing the state of this belief. This is necessary because the belief may already be true and the person will not do any planning unless the current state of the associated obligation is known. For example, if you want to have the light on but you don't know whether it is on or off you would find out before you considered ways of turning it on.

We have not described how one plans for fact beliefs which are not of type Physical-Fact. The reason is that in our situation this is not necessary because of an assumption. The assumption is that only a Request knowledge action can be used to modify such beliefs and this is done by the system when presented with such a goal by the user. In other words, if the person wants to know how to do something, then he has no way of finding out except to ask the system. This is covered in more detail in the §4.4 which describes the complete planning process.

The result of planning is a tree of possible worlds connected by actions which we call a plan tree. A plan is a path through a plan tree. Another way to think of this is that a plan tree is a set of possible future worlds and a plan is a particular path through the plan tree. Each world is a unique environment for the PROLOG interpreter. This means that when the interpreter is executing in that world only the clauses that exist in the world are visible. The interpreter can move from world to world by the VIEWPOINT clause which has the form

```
((VIEWPOINT Future-Action ?Action-Description))
((VIEWPOINT Past-Action ?Action-Description))
```

There are clauses of the form (FUTURE-ACTION ?Action-Description) and (PAST-ACTION ?Action-Description) present in each world. The FUTURE-ACTION clauses describe the actions that when executed in the present world create a future world. The PAST-ACTION clauses describes the action which was executed in the past world to create the present world. These clauses link together the worlds into a plan tree. Every attempt to produce another plan by reprovving the PLAN-FOR-GOAL clause adds worlds to the plan tree. Using this method we can save the results of planning in a simple and efficient way to be

used later by other reasoning processes

System Description

4.1 Overview

The basic idea of the system is that it simulates the planning of the person using the person's beliefs, and whenever necessary it behaves cooperatively. Cooperative behavior means that the system tells the person something that is necessary to accomplish the task, or warns the person about some situation that should be avoided during the task. The prolog code for the main loop is as follows:

```
((MAIN)
  (FOR-ALL
    ((GET-ENGLISH-INPUT ?English-Input))
    ((PROCESS-INPUT ?English-Input))))

((PROCESS-INPUT ?English-Input)
  (ENGLISH-TO-SEMANTIC-FORM ?English-Input ?Semantic-Form)
  (VALIDATE-SPEECH-ACT ?Semantic-Form)
  (PROCESS-POSSIBLE-DIRECT-REQUEST ?Semantic-Form)
  (PROCESS-POSSIBLE-INDIRECT-REQUEST ?Semantic-Form))

((PROCESS-INPUT ?*))

((PROCESS-POSSIBLE-DIRECT-REQUEST (Goal ?Type Person (Believe Person ?Belief))
  (PROVE-EVERY
    (Plan-For-Goal
      ?Type Person (Believe Person ?Belief)
      (Person-Said (Inform System (Goal ?Type Person (Believe Person ?Belief))))))
  (EXECUTE-PREDICTIONS)))

((PROCESS-POSSIBLE-DIRECT-REQUEST ?*))

((PROCESS-POSSIBLE-INDIRECT-REQUEST ?Semantic-Form)
  (INFER-REQUEST ?Semantic-Form ?Inferred-Goal))
```

(PROVE-EVERY
  (Plan-For-Goal
    Physical Person ?Inferred-Goal (Inferred-Goal ?Semantic-Form)))
(EXECUTE-PREDICTIONS))

((PROCESS-POSSIBLE-INDIRECT-REQUEST ?*))

The steps in the PROCESS-INPUT clause are

1. The english input is converted to a semantic form. We do not describe in detail how this is done because we *assume* that it can be done using standard techniques.
2. The speech act which this input represents is validated. This is done using speech act rules that examine the system's beliefs to decide on the type and validity of the speech act. If the input statement is in conflict with the system's beliefs the system engages in a dialogue with the person to adjust its beliefs.
3. If the speech act is a request then the system simulates the planning of the person to decide where it should behave cooperatively. The result of this is a plan tree, which when executed produces a dialogue with the person.
4. Further processing is done to infer an indirect speech act using speech act rules and the system's beliefs. Some of these rules are independent of the type of task environment while others would be used only for a particular task. If an indirect request speech act has been inferred then the system once again simulates the planning of the person to decide where it should behave cooperatively. The resulting plan is executed which produces another dialogue with the person.

4.2 Parse

Parsing takes the english input and produces a semantic form. There are two possible semantic forms.

The first semantic form is the clause (BELIEVE Person ?Belief). This semantic form means the system has been told something that the person believes to be true. For example

"I am cold"

⇒ (BELIEVE Person (Physical-Fact (Person-State Cold)))

The second semantic form is the clause (GOAL ?Type Person ?Belief). This semantic form means that the system has been told of some goal of the person. For example:

"Where is the wrench"

⇒ (GOAL
     Knowledge Person
     (Believe Person (Physical-Fact (Wrench-Location ?*))))

"I want the wrench at the generator base"

⇒ (GOAL
     Physical Person
     (Believe Person (Physical-Fact (Wrench-Location Generator-Base))))

Note that the semantic form can have any propositional content, that is the variable ?Belief can be any predicate in our knowledge representation

The translation process is not automatic, but we believe that there are standard methods¹ that can be used which are reasonably successful

¹ see Schank. 81

4.3 Speech Act Processing

There are two kinds of speech acts that the system processes. The first is (INFORM System ?Belief) which means that the person is telling the system something that the person believes. In order for the semantic form of type ?Belief to be a valid inform the following conditions must hold

1. The system must believe the person has this knowledge. The only thing the person can know is the state of some physical fact. This condition can be violated if the system does not have this knowledge, and the inform will still be valid. However, the system will show surprise when presented with such an inform since the person should not have known the state of the physical fact (this implies that *neither* the person nor the system should have known it).
2. The person must believe the system does not have this knowledge. The system assumes that
  - the person is aware of the initial state of the system's knowledge base. For example, the person may have been told that the system knows everything about how to perform a given task and everything about the location of certain kinds of tools.
  - the person is aware that the system knows (or can infer) what actions have taken place and thus knows the state of any physical facts that have been modified by the execution of these actions by the person.

These assumptions mean that the person is aware of what the system knows. If any of the above conditions are violated the statement is rejected as an inform. If the semantic form of type BELIEVE meets both these requirements it is a true inform and the inform speech act is executed. It has the following definition:

((PERSON-INFORM (Physical-Fact (*Fact-Head ?Fact-State))))

```

(Execute-Action
  (Action-Description
    Person
    (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    (Preconditions) (Body)
    (Additions (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    (Deletions (Believe Person (Physical-Fact (?Fact-Head ?*))))
    (Restrictions) (Sensors))))

```

After it is executed the beliefs of the system are changed so that the person has the appropriate beliefs.

We will use the situation where a light can be on or off as an example to illustrate how the inform speech act is processed. The processing starts with the following clause:

```

((PROCESS-SPEECH-ACT (Believe Person ?Belief))
 (GET-TRUTH-VALUE Person ?Belief ?Truth-Value)
 (PROCESS-POSSIBLE-INFORM ?Belief ?Truth-Value))

```

If the person knows whether the light is on or off, and the system does not (in our knowledge representation this has a truth value of known) we have a valid inform.

```

((PROCESS-POSSIBLE-INFORM ?Belief Known)
 (WRITE (OK))
 (PERSON-INFORM ?Belief))

```

If the person should not have known whether the light was on or off, this may still be a valid inform but further processing is required.

```

((PROCESS-POSSIBLE-INFORM ?Belief Unknown)
 (GET-TRUTH-VALUE SYSTEM ?Belief ?Truth-Value)
 (FURTHER-INFORM-PROCESSING ?Belief ?Truth-Value))

```

The further processing is that the system checks whether it believes that the light is on or off. If it agrees with what the person has told it, or it does not know, it still accepts this as a valid inform.

```

((FURTHER-INFORM-PROCESSING ?Belief Unknown)
 (WRITE (I Did Not Know That And Did Not Think That You Did))
 (PERSON-INFORM ?Belief))

```

```
((FURTHER-INFORM-PROCESSING ?Belief True)
(WRITE (I Agree And Did Not Think You Knew That))
(PERSON-INFORM ?Belief))
```

If on the other hand, it disagrees with what the person has said about the state of the light then the system must adjust its beliefs via a dialogue with the person

```
((FURTHER-INFORM-PROCESSING ?Belief False)
(WRITE (I Disagree)) !
(ADJUST-SYSTEM-BELIEFS ?Belief))
```

This completes the further processing for the case where the person should not have known the state of the physical fact (whether the light was on or off)

If the system believes they both disagree with what the person has just said about the state of the light, the system must adjust its beliefs via a dialogue with the person

```
((PROCESS-POSSIBLE-INFORM ?Belief False)
(WRITE (I Disagree And Thought You Did Too)) !
(ADJUST-SYSTEM-BELIEFS ?Belief))
```

If the person has told the system the state of the light, and the person believes the system already knows this, then the system ignores the utterance (no comment is made because it may later be processed as an indirect speech act)

```
((PROCESS-POSSIBLE-INFORM ?Belief True))
```

This completes the processing for an inform speech act

The second type of speech act is (REQUEST System ?Goal) which means that the person is telling the system of his goal. In order for the semantic form of type GOAL to be a valid request the following conditions must hold

1. If the goal is of type physical then the system must not believe that the person believes that this physical state already exists (the goal has been achieved)
2. If the goal is of type knowledge then system must not believe that the person already has this information (the goal has been achieved)

If a semantic form of type GOAL meets these requirements then it is a valid request and the request speech act is executed. It has the following definition.

```
((PERSON-REQUEST ?Type ?Belief)
  (EXECUTE-ACTION
    (Action-Description
      Person (Request System (Goal ?Type Person (Believe Person ?Belief)))
      (Preconditions) (Body)
      (Additions (Believe Person (Goal ?Type Person (Believe Person ?Belief))))
      (Deletions) (Restrictions) (Sensors))))
```

After it is executed the beliefs of the system are changed so that the person has the appropriate goal.

Consider the same situation where there is a light that can be on or off to illustrate how a request speech act is processed. The processing begins with the following clause:

```
((PROCESS-SPEECH-ACT (Goal ?Type Person (Believe Person ?Belief)))
  (GET-TRUTH-VALUE PERSON ?Belief ?Truth-Value)
  (PROCESS-POSSIBLE-REQUEST ?Type ?Belief ?Truth-Value))
```

If the person wants to know whether the light is on or off and the system believes the person already knows; or the person wants the light to be on or off and the system believes the person believes it is already in the desired state, then the system must adjust its beliefs by engaging in a dialogue with the person.

```
((PROCESS-POSSIBLE-REQUEST PHYSICAL ?Belief True)
  (WRITE (I Think This Is Already True)) !
  (ADJUST-SYSTEM-BELIEFS ?Belief))
```

```
((PROCESS-POSSIBLE-REQUEST KNOWLEDGE ?Belief True)
  (WRITE (I Think You Already Know That This Is True)) !
  (ADJUST-SYSTEM-BELIEFS ?Belief))
```

```
((PROCESS-POSSIBLE-REQUEST KNOWLEDGE ?Belief False)
  (WRITE (I Think You Already Know That Is Not True)) !
  (ADJUST-SYSTEM-BELIEFS ?Belief))
```

```
((PROCESS-POSSIBLE-REQUEST KNOWLEDGE ?Belief Known)
  (WRITE (I Do Not Know But I Think That You Know)) !
  (ADJUST-SYSTEM-BELIEFS ?Belief))
```

If none of the above conditions are true this is a valid request speech act. This leaves the situation where either the person wants the light to change state and believes the light is in the opposite state, or he wants to know what state the light is in and does not have this knowledge.

```
((PROCESS-POSSIBLE-REQUEST ?Type ?Belief ?Truth-Value)
  (WRITE (Hm...)))
  (PERSON-REQUEST ?Type ?Belief))
```

4.4 Plan

We assume that the system always accepts the person's goals as valid. This means the system need not have its own goals which would force it to decide whether to accept or reject the person's goals. The system simulates the planning process of the person using what it believes the person knows to guide it in performing cooperative behavior. We have defined cooperative behavior as the system telling the person something that is necessary to accomplish the task, or warning the person about some situation that should be avoided. The idea is not for the system to tell the person what to do but to make sure that the person has the knowledge to be able to create a reasonable plan to accomplish the goal (reasonable means one that works and does not lead to disaster).

Here we concentrate on how the system helps the person by telling the person something he does not know that is necessary to perform the task. The basic idea is that during the simulation of the person's planning if the person does not know something that is necessary the system has the goal of obtaining this knowledge, and telling the person.

The clause that is invoked when the person does not have the appropriate knowledge is as follows:

```

((ACHIEVE-GOAL Knowledge Person ?Belief Unknown ?Justification)
 (FACT-BELIEF ?Belief)
 (NOT EQ ?Belief (Actions-For-Obligation ?Type Knowledge . ?*))
 (PLAN-FOR-GOAL
  Knowledge System (Believe System ?Belief)
  ((Help-Person-Know-State ?Belief) (Goal Knowledge Person (Believe Person ?Belief))))
 (GET-CURRENT-FACT-STATE System ?Belief ?New-Belief)
 (EXECUTE-ACTION
  (Action-Description
   System (Inform Person ?New-Belief) (Preconditions) (Body)
   (Additions (Believe Person ?New-Belief)) (Deletions)
   (Restrictions) (Goal Knowledge Person (Believe Person ?Belief))))
 (WRITE (System))
 (WRITE (Inform Person ?New-Belief))
 (DELETE-CLAUSE (Goal Knowledge Person (Believe Person ?Belief) ?*)))

```

This clause forces the system to have the same knowledge goal as the person, and then to tell the person the answer using an inform speech act.

If the system has no knowledge about a physical fact that the person wants to know the state of (this implies the person does not know either) then the system is prepared to make the appropriate assumption.

```

((ACHIEVE-GOAL
 Knowledge System (Physical-Fact (?Fact-Head ?Fact-State)) Unknown ?Justification)
 (NOT BELIEVE System (Actions-For-Obligation ?Fact-Head Knowledge . ?*) ?*)
 (OBLIGATION ?Fact-Head . ?Possible-States)
 (MEMBER ?Fact-State ?Possible-States)
 (EXECUTE-ACTION
  (Action-Description
   System (Assume ?Fact-Head ?Fact-State) (Preconditions) (Body)
   (Additions (Believe System (Physical-Fact (?Fact-Head ?Fact-State))))
   (Deletions) (Restrictions)
   (Goal Knowledge System (Physical-Fact (?Fact-Head ?Fact-State))))
 (WRITE (System))
 (WRITE (Assume ?Fact-Head ?Fact-State))
 (DELETE-CLAUSE
  (Goal
   Knowledge
   System
   (Believe System (Physical-Fact (?Fact-Head ?Fact-State))) ?Justification)))

```

Thus allows the system to imagine several different alternatives in the plan tree. For example, if the person needs to know if the light is on or off and neither the person nor the system know, then the system will imagine *both* alternatives and continue planning in each of them.

The system's simulation of the person's planning has knowledge goals inserted in strategic places. This makes it possible for the system to help the person if the person does not have the appropriate knowledge. The main ACHIEVE clause for the person now becomes:

```
((ACHIEVE-GOAL
  ?Type Person (Physical-Fact (?Fact-Head . ?Fact-Rest)) ?Truth-Value ?Justification)
 (LEGAL-TYPE-TRUTH-VALUE-COMBINATION ?Type ?Truth-Value)
 (BELIEVE
  System
  (Actions-For-Obligation ?Fact-Head ?Type ?Action-Descriptions) Premise)
 (PLAN-FOR-GOAL
  Knowledge Person
  (Believe Person (Actions-For-Obligation ?Fact-Head ?Type ?Action-Descriptions))
  ((Know-Action ?Fact-Head)
   (Goal ?Type Person (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))))))
 (EVERY-MEMBER
  (Action-Description
   ?Type Person ?Description (Preconditions . ?P) (Body . ?B) (Restrictions . ?R)
   (Additions
    (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest)))
    ?Rest-Additions)
   ?Rest-Action)
  ?Action-Descriptions)
 (TRY-EXECUTE-ACTION
  (Action-Description
   ?Type Person ?Description (Preconditions . ?P) (Body . ?B) (Restrictions . ?R)
   (Additions
    (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest)))
    ?Rest-Additions)
   ?Rest-Action)
  ?Justification))

((LEGAL-TYPE-TRUTH-VALUE-COMBINATION Physical False))

((LEGAL-TYPE-TRUTH-VALUE-COMBINATION Knowledge Unknown))
```

This is the same clause that was described in §3.4. The only change is that the person now has an additional goal of knowing how to accomplish the task since the person presently may not have this knowledge.

Another place where the knowledge goals are added is the following clause:

```
((TRY-EXECUTE-ACTION
  (Action-Description
   ?Type ?Actor (?Name . ?Arguments) (Preconditions . ?P) (Body . ?B)
   (Restrictions . ?R) (Additions . ?A) (Deletions . ?D))
  ?Justification)
 (PFIRST ?Goal ?A) !)
```

```

(CHECK-IF-DESCRIPTIONS-KNOWN ?P (Goal ?Type ?Actor ?Goal))
(PRODUCE-WARNINGS ?Type ?Name (Goal ?Type ?Actor ?Goal))
(CLASSIFY-OBLIGATIONS ?P ?Free-Obligations ?Fixed-Obligations)
(PLAN-FOR-GOALS
  ?Type ?Actor ?Fixed-Obligations
  ((Want-Precondition (?Name ?Arguments)) (Goal ?Type ?Actor ?Goal)))
(FIX-FREE-OBLIGATIONS ?Actor ?Name ?Free-Obligations ?New-Fixed-Obligations)
(PLAN-FOR-GOALS
  ?Type ?Actor ?New-Fixed-Obligations
  ((Want-Precondition (?Name ?Arguments)) (Goal ?Type ?Actor ?Goal)))
(GET-BELIEFS-IN-RESTRICTIONS ?R ?Restriction-Beliefs)
(PLAN-FOR-GOALS
  Knowledge ?Actor ?Restriction-Beliefs
  ((Want-Know-Restrictions (?Name ?Arguments)) (Goal ?Type ?Actor ?Goal)))
(CHECK-RESTRICTIONS ?R)
(TRY-EXECUTE-BODY ?Actor ?B (?Name ?Arguments) ?Goal)
(CHECK-OBLIGATIONS ?A)
(EXECUTE-ACTION
  (Action-Description
    ?Actor (?Name ?Arguments) (Preconditions ?P) (Body ?B)
    (Additions ?A) (Deletions ?D) (Restrictions ?R)
    (Goal ?Type ?Actor ?Goal)))
(DELETE-CLAUSE (Goal ?Type ?Actor ?Goal ?Justification))
(WRITE ?Actor)
(WRITE (?NAME ?ARGUMENTS)))

```

This clause invokes the clauses PRODUCE-WARNINGS and CHECK-IF-DESCRIPTIONS-KNOWN which both force various knowledge goals for the person. It is the clause which tries to execute the action which reinvokes the planner in a backwards chaining fashion.

The result of planning for the person's goals is a plan tree which contains all the alternative ways of achieving the goal with the system's actions. The plan tree is interpreted with the person's actions. The plan tree provides expectations about the future, memory about the past, and controls the execution phase.

4.5 Warn

We have not implemented this capability but we feel confident enough in the ideas we present to include them in this chapter. We have defined warning as the system telling the person of some situation that should be avoided. The basic idea is to define constraints on

every possible world using inferred beliefs. A constraint would be a condition that should not be true in any world.

During the planning process the system assumes that if the person is aware that a world violates a constraint the world will be abandoned and some other alternative will be taken. The system would warn the person by telling him the constraint if he did not already know it. This should be sufficient for the person to avoid the undesired world. It is simpler to consider the constraints as separate from actions than to define the constraints as restrictions on each action. The reason is that the same restriction would have to be applied to a number of actions to be an effective constraint.

An example of a constraint is the situation where a person should never touch a power cable when the power is on. This would be defined with the following inferred belief:

```
((BELIEVE ?Anyone (Person-Electrocuted) ?*)  
  (BELIEVE ?Anyone (At-Hand ?Place) ?*)  
  (BELIEVE ?Anyone (At-Cable ?Place) ?*)  
  (BELIEVE ?Anyone (Power On) ?*)))
```

If this constraint is ever satisfied in a world then that world should be abandoned. The person may have a knowledge goal of finding out whether a constraint is true or false if the truth value of the constraint is unknown.

Constraints can also be classed according to the effects that would occur if they are violated and a possible world in which a constraint is satisfied becomes the real world. The more detrimental the effect, the more serious the constraint. A very serious constraint would force the system to warn the person even if the person knew that constraint. The person being electrocuted is an example of a very serious constraint and the system would warn the person to avoid this situation whenever it might occur.

Implementing these ideas is not a serious undertaking since the basic mechanisms that they use (inferred beliefs, plan tree, etc.) are already in place.

4.6 Infer

This is basically a plan recognition process, by which we mean that the system tries to infer the person's goals from his utterances. The reason this step is necessary is that the person does not always specify his goal explicitly, but expects the system to use its knowledge of the person's beliefs along with general speech act rules to *infer* these goals. This corresponds to the idea of an indirect speech act, which is the use of one speech act which also has the indirect meaning of another speech act, besides its own direct meaning (if there is one). The reason that human conversation has this characteristic is that it makes speech more economical. This is because the participants depend on each others common knowledge and reasoning processes to produce the complete meaning (all the speech acts) of an utterance.

The infer processing starts with the following two clauses

```
((INFER-REQUEST (Goal ?Type Person (Believe Person ?Belief)) ?Inferred-Goal)
 (PROVE-AND-COUNT
  (?Inferred-Goal)
  (Infer-Higher-Goals-Given-Goal ?Type (Believe Person ?Belief) ?Inferred-Goal))
 (VERIFY-INFERENCE ?Inferred-Goal))
```

```
((INFER-REQUEST (Believe Person ?Belief) ?Inferred-Goal)
 (PROVE-AND-COUNT
  (?Inferred-Goal) (Infer-Higher-Goals-Given-Fact ?Belief ?Inferred-Goal)))
```

The clause PROVE-AND-COUNT proves the clause in every possible way and counts the number of ways. It is used to verify that only *one* indirect user goal can be inferred. The system *assumes* the person would expect the system to infer his goal *only* if the situation led to an *unambiguous* (which means one) inferred goal. If more than one goal can be inferred the choice is ambiguous and the system will not infer any user goal from the utterance because it believes the person does not *want* any goal to be inferred.

The following clauses are invoked when the person's original utterance was a statement of some belief

```

((INFER-HIGHER-GOALS-GIVEN-FACT
  (Physical-Fact (?Fact-Head ?Fact-State))
  (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
(OBLIGATION ?Fact-Head ?Fact-State ?Inferred-Fact-State)
(UNDESIRABLE-STATE-OF-OBLIGATION ?Fact-Head ?Fact-State) !))

((INFER-HIGHER-GOALS-GIVEN-FACT
  (Physical-Fact (?Fact-Head ?Fact-State))
  (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
(OBLIGATION ?FACT-HEAD ?Inferred-Fact-State ?Fact-State)
(UNDESIRABLE-STATE-OF-OBLIGATION ?Fact-Head ?Fact-State) !))

((INFER-HIGHER-GOALS-GIVEN-FACT
  (Physical-Fact (?Fact-Head ?Fact-State))
  (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
(OBLIGATION ?FACT-HEAD ?Inferred-Fact-State ?Fact-State)
(NOT PAST-ACTION
  (Action-Description
    Person
    (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    ?*)) !))

((INFER-HIGHER-GOALS-GIVEN-FACT
  (Physical-Fact (?Fact-Head ?Fact-State))
  (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
(OBLIGATION ?Fact-Head ?Fact-State ?Inferred-Fact-State)
(NOT PAST-ACTION
  (Action-Description
    Person
    (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
    ?*)) !))

```

The first two clauses say that if the person tells the system the state of some physical fact and this is an undesirable state, then the person is really telling the system that he has a goal of achieving the opposite of this state. For example, consider the situation where the person tells the system "I am hungry". The system would use these clauses to infer that the person is really telling the system "I want to be full (not hungry)". Note that it *does* not matter whether the original utterance was a valid inform or not. The fact that the system did not know the person was hungry does not affect the conclusion that the person wants to be full because the state of being hungry is an undesirable state.

The second two clauses say that if the person tells the system the state of some physical fact that the system already knows, then the person is really telling the system he has a

goal of achieving the opposite of this state. These two clauses are used if the current state is not obviously an undesirable state. If this is the case then the fact that the system already knows this state exists (since there was no valid inform) means there is strong evidence that the person wants this physical fact to be in some other state. For example, consider the case where the person has said "The light in the corner is on". Now it is not clear that this is an undesirable state, but if the person believes the system already knows this, then the person's utterance makes *no sense* unless it is an indirect speech act. The system uses these clauses to infer that the person is really telling the system "I want the light in the corner to be turned off".

This completes the processing where the original utterance was a statement of the person's belief.

The following clause is used if the original utterance was a statement of the person's goal.

```
((INFER-HIGHER-GOALS-GIVEN-GOAL ?Type ?Goal ?Higher-Goal)
 (FIND-INFERRED-PHYSICAL-GOAL ?Type ?Goal ?Inferred-Goal) !
 (FIND-HIGHER-GOALS ?Inferred-Goal ?Higher-Goal)
 (NOT EQ ?Goal ?Higher-Goal))
```

The final goal that is inferred is always a physical goal while the original goal may have been a knowledge goal. For this reason the clause FIND-INFERRED-PHYSICAL-GOAL is used to infer what the person could want to do from what the person wants to know.

This is done using some general goal rules that are actually heuristics since they are not guaranteed to be correct, but only suggest an inference that can be made.

The following clause says that if someone wants to know how to achieve a particular state then he wants to achieve that state. For example, if the person says "How do I install the generator?" he means that besides knowing how to install it he actually wants to install the generator.

```
((FIND-INFERRED-PHYSICAL-GOAL
 Knowledge
 Believe
```

```

Person
(Actions-For-Obligation
  ?* Physical
  ((Action-Description
    Physical Person ?* ?* ?* ?*
    (Additions (Believe Person (Physical-Fact ?Fact))) ?* ?*)))
(Believe Person (Physical-Fact ?Fact)))

```

The following clause says that if the person wants to know whether some physical fact is in a particular state then he actually wants the physical fact to be in that state. For example, if the person says "Is the oven hot?" then besides wanting to know the answer he actually wants the oven to be hot.

```

((FIND-INFERRED-PHYSICAL-GOAL
  Knowledge
  (Believe Person (Physical-Fact ?Fact)) (Believe Person (Physical-Fact ?Fact))))

```

The following clause says that if the person wants to know the description of some object then he actually wants that object to be located where he is currently located. The object is normally a tool that is necessary for some task. For example if the person says "What's a socket wrench" and he is located at the generator then besides wanting the answer he actually wants the socket wrench at the generator.

```

((FIND-INFERRED-PHYSICAL-GOAL
  Knowledge
  (Believe Person (Description ?Object ?Description))
  (Believe Person (Physical-Fact (?Obligation ?State))))
(CONVERT-BETWEEN-OBLIGATION-AND-OBJECT ?Obligation ?Object)
(GET-TRUTH-VALUE Person (Physical-Fact (At-Person ?State)) True))

```

This clause says that if the original goal was a physical goal instead of a knowledge goal, then this goal is taken as the current inferred physical (it is processed further and can still be rejected).

After the FIND-INFERRED-PHYSICAL-GOAL clause has been executed the system has inferred some physical goal from the original utterance. The system now tries to find higher level goals that subsume this goal. This is done by the clause find higher goals, which finds an action that has the inferred goal as a precondition, and then infers higher level actions that have this action in their body field.

```

((FIND-HIGHER-GOALS ?Inferred-Goal (Execute-Action ?Highest-Action-Description))
 (FIND-ACTION-WITH-GOAL-AS-PRECONDITION
  ?Inferred-Goal
  (Action-Description
   ?* Person ?First-Action-Description ?* ?* ?* (Additions ?A) ?* ?*))
 (NOT ADDITIONS-TRUE ?A)
 (INFER-HIGHER-ACTIONS-GIVEN-DESCRIPTION
  ?First-Action-Description ?Rest-Actions-Description)
 (NOT EQ ?Rest-Actions-Description ())
 (FINAL
  (?First-Action-Description ?Rest-Actions-Description)
  ?Highest-Action-Description) ')

```

At each step the system rejects any actions whose affects are already true since there is no reason that the person would have a goal of executing such actions

This process finds the highest level goal that can be inferred in this situation. For example if the person says "Is the socket wrench at the generator" then the clause FIND-INFERRED-PHYSICAL-GOAL would conclude that the person wants the socket wrench at the generator. The clause FIND-HIGHER-GOALS would conclude that the inferred goal is a precondition of the action TIGHTEN-GENERATOR-BOLTS, and that this action is in the body of the action INSTALL-GENERATOR. Therefore the conclusion is that the person really wants to install the generator. Since these inferences are not guaranteed to be correct the system verifies the inference with the person

4.7 Execute

The plan tree created by the planning process is used to predict the future and thus to guide the system during its interactions with the person. There are two modes of interacting with the person during plan execution

<i>Novice Mode</i>	in this mode the system tells the person what he needs to know right before he needs to know it. In order to do this the system asks the person to tell the system when a particular action has been completed. The action is the one the person must perform right before the system must tell the person the next thing he needs to know. This process is repeated until the task is complete
<i>Expert Mode</i>	in this mode the system waits for the person to ask a question. If the question is one of the predicted questions then the system

will answer it without comment. It will also make assumptions about the actions taken by the person since the last question.

Consider a world in a plan tree, a future action in that world, and the intelligence that executes that action. Informally we define a *region* as all the future worlds (not including this world) from this world whose past action was executed by the same intelligence. The plan tree is divided into alternating regions where the intelligence executing the actions is the person, and then the system.

The execution process for the novice mode starts with the following clause:

```
((EXECUTE-PLAN ?Previous-Assumptions)
 (FUTURE-ACTION (Action-Description Person ?Rest-Action))
 (VIEWPOINT Future (Action-Description Person ?Rest-Action))
 (VERIFY-PERSON-ACTION (Action-Description Person ?Rest-Action) ?Previous-Assumptions)
 (EXECUTE-PLAN ?Previous-Assumptions))
```

This clause skips over each of the person's actions in a region verifying the action at the same time.

The verification process starts with the following clause. If the action is based on an assumption that the system made (remember this only happens because *neither* the system nor the person know, or can discover anything about this physical fact) then this clause checks if this action was successful. If so the action is accepted, if not some other assumption is checked (the assumptions covers every possible state of the physical fact).

```
((VERIFY-PERSON-ACTION (Action-Description Person ?Name ?Rest-Action) ?Assumptions)
 (WORLD-IDENTIFIER ?Current-World)
 (ACTION-BASED-ON-ASSUMPTION ?Answer)
 (JUMP-WORLD ?Current-World)
 (EQ ?Answer Yes)
 (INPUT (Did You Successfully ?Name) ?Response)
 (EQ ?Response Yes))
```

The following clause is invoked if the future action is a system action. This implies that this is a synchronization point because we must be at the end of the person's region. The system tells the person to tell the system when a synchronization point is reached. When this occurs the system will tell the person what he needs to know (by telling him everything in the system's region) in order to proceed with this task.

```
((VERIFY-PERSON-ACTION (Action-Description Person ?Name . ?Rest-Action) ?Assumptions)
(FUTURE-ACTION (Action-Description System . ?*))
(NOT EQ ?Name (Inform . ?*))
(NOT EQ ?Name (Check . ?*))
(INPUT (Tell Me When You Have Done Action ?Name) ?Response) !)
```

The following clause is invoked if the person's action is a knowledge action, in which case the system checks the result of the execution of that action with the person. The knowledge action will have an alternative instance for each possible state of the physical fact. Eventually the system will find out which knowledge action was executed successfully because the number of possible alternatives is limited.

```
((VERIFY-PERSON-ACTION
  (Action-Description Person (Check ?Obligation ?State) . ?Rest-Action)
  ?Assumptions)
(INPUT (Is The ?Obligation In The State ?State) ?Response) !
(EQ ?Response Yes))
```

If none of the above cases are true the action is considered to be verified automatically by the following clause. This is because the system assumes there is only one way of accomplishing a task so if the action is not a knowledge action the person must execute it. This assumption could be removed by having the system verify any physical action that is one of a number of possible physical actions that could achieve the same goal.

```
((VERIFY-PERSON-ACTION (Action-Description Person ?Name . ?Rest-Action) ?Assumptions))
```

The execution for the novice mode continues with the following clause:

```
((EXECUTE-PLAN ?*)
(FUTURE-ACTION (Action-Description System . ?*))
(FUTURE-ACTION (Action-Description System . ?Rest-Action))
(PERFORM-ALL-SYSTEM-ACTIONS (Action-Description System . ?Rest-Action) ?New-Assumptions)
(EXECUTE-PLAN ?New-Assumptions))
```

This clause is invoked only when the execution is in the system's region and the person's region has been processed. The clause PERFORM-SYSTEM-ACTIONS executes all the actions that the system must perform in the region. Since the system can only do an INFORM or an ASSUME the result of executing these actions is that the system tells the person something.

The execution terminates if there is no future action which is checked by the following clause

```
((EXECUTE-PLAN ?*)
 (NOT FUTURE-ACTION ?*))
```

The execution process for the expert mode starts with the following clauses

```
((SILENT-EXECUTE-PLAN)
 (FUTURE-ACTION (Action-Description System ?Rest-Action))
 (PERFORM-ALL-SYSTEM-ACTIONS (Action-Description System ?Rest-Action))
 (WORLD-IDENTIFIER ?Current-Region)
 (PROCESS-REGION-INPUT ?Current-Region ()))
```

```
((SILENT-EXECUTE-PLAN)
 (FUTURE-ACTION (Action-Description Person ?*))
 (WORLD-IDENTIFIER ?Current-Region)
 (PROCESS-REGION-INPUT ?Current-Region ()))
```

The first clause initializes the region identifier, which is the world identifier of the first world in that region. The system checks the validity of the question and then responds by invoking the following clauses:

```
((RESPOND-TO-INPUT
 (Goal Knowledge Person (Believe Person ?Belief))
 ?Current-Region ?Current-Region
 ?Currently-Answered (?Belief ?Currently-Answered))
 (ANSWER-QUESTION ?Belief)
 (JUMP-WORLD ?Current-Region))
```

```
((RESPOND-TO-INPUT
 (Goal Knowledge Person (Believe Person ?Belief))
 ?Current-Region ?New-Region
 ?Currently-Answered ?New-Currently-Answered)
 (END-ACTIVE-REGION ?Currently-Answered)
 (NEXT-ACTIVE-REGION ?New-Region ?Persons-Actions)
 (DESCRIBE-ASSUMED-ACTIONS ?Persons-Actions)
 (RESPOND-TO-INPUT
 (Goal Knowledge Person (Believe Person ?Belief))
 ?New-Region ?New-Region () ?New-Currently-Answered))
```

The system attempts to answer the question by finding the answer in the current system region. If the answer is not found in the current region the system searches the next region for the answer. If this region contains the answer it becomes the current region and the

system then describes the actions that it assumes were made in the person region between the two system regions

4.8 Justify

One of the reasons for having justifications is that it allows the system to explain *why* it has performed some action, which in our case means *why* it has told the person something. Justifications are the last thing in a clause regardless of the type of the clause.

A fact/belief is justified by "Premise" or by a short description of the action that created this fact. The action could be a physical action or a knowledge action (a speech action is classed as a knowledge action)

An action can be justified by "Sensors" if it is a speech action that was executed by the person. The justification comes from the fact that the system has knowledge of the action from its sensors. All other actions are justified by the goals that caused the action to be undertaken.

A goal is justified by a list of the form (?Goal-Rule ?Goal-Justification). The variable ?Goal-Rule is some description of the rule (reasoning) that contributed to this goal. An example of a goal rule is (WANT-PRECONDITION ?Action-Description) which means that this goal exists because it is a precondition of some action described by ?Action-Description. The variable ?Goal-Justification is normally the goal which along with the goal rule caused this goal. The goal justifications follow a path through goals and terminate with a goal that is justified by a speech action of the person, or to a goal that is justified by a fact which is itself justified by a speech action of the person. In other words, the system does something only because the person tells it some belief or goal. The justification of goals are added by the reasoning processes (PROLOG code) when this reasoning takes place.

If at any point the person asks *why* the system performed some action the following

clause is invoked.

```
((SHOW-JUSTIFICATIONS ?Action-Description)
 (FIND-GOAL-JUSTIFICATION-FOR-ACTION ?Action-Description ?Goal-Justification)
 (PRINT-GOAL-JUSTIFICATION-CHAIN ?Goal-Justification))
```

This clause finds the action that occurred in the past from its short description (sometimes called the action name) and then prints the goal justification chain.

The goal justification chain is printed by the following clauses

```
((PRINT-GOAL-JUSTIFICATION-CHAIN (Goal ?Type ?Actor (Believe ?Actor ?Belief)))
 (GOAL ?Type ?Actor (Believe ?Actor ?Belief) (?Rule ?Parent-Goal))
 (WRITE (Goal ?Type ?Actor ?Belief Because ?Rule))
 (INPUT (Further Justifications) ?Response) !
 (EQ ?Response Yes)
 (PRINT-GOAL-JUSTIFICATION-CHAIN ?Parent-Goal))
```

```
((PRINT-GOAL-JUSTIFICATION-CHAIN (Inform . ?*))
 (WRITE (Because You Told Me)))
```

```
((PRINT-GOAL-JUSTIFICATION-CHAIN (Believe Person (Physical-Fact ?Fact)))
 (BELIEVE Person (Physical-Fact ?Fact) ?Justification)
 (WRITE (Believe Person (Physical-Fact ?Fact) Because ?Justification)))
```

These clauses print the goal justification chain which terminates with a system inform or a belief which is justified by a system inform.

4.9 Debug

This comes about when the person has made a mistake, which is usually that he has forgotten to perform some action. For this reason the person reaches a situations where:

- there are some parts left over that have not been used, or
- some action that the person should be able to perform can not be performed,
- or
- a fault exists.

When the person tells the system of any of these conditions the system should try to help the person by identifying the missing action and explaining how to fix the current situation.

This is done in two parts.

- 1 The first is a process of belief revision. This is necessary because the system believes the person has done everything properly, but the person has actually forgotten to perform some action. This implies that something that the system believes happened in the past did not really happen, so the past worlds in the plan tree must be modified. For each of the above situations the process of identifying the missing action(s) is as follows:

- The system asks the person whether the action that should have used the left over part was performed.
- The system looks at the preconditions of the failing action(s) and asks the person whether the action that justified these preconditions was performed.
- The system looks at the fault and asks the person whether the action that would have negated this fault condition was performed.

This dialogue results in the system identifying the missing action(s) and adjusting its beliefs about the past actions.

2. The second part is that the system must create a plan to rectify the situation. If the action is reversible the plan could be as simple as reversing some actions, performing the missing action, and then re-doing the other actions. If there are actions that can not be directly reversed the system would produce a plan to reverse these actions.

We have implemented a simple version of the debugging process for fault conditions.

The faults are described by inferred beliefs which list the fact beliefs that support the fault.

An example is the following clause:

- 3 If the person is hungry he can satisfy his hunger by eating food
- 4 Food is defined to be an apple or a banana
- 5 The person can get a stick and use it to knock down bananas
- 6 There is a stick somewhere
- 7 The person knows whether he is hungry or not and the system knows that he knows

The following facts are known only by the system

- 1 The apples are not good to eat even though they are classified as food (the reason is not stated - a good reason would be that they are poisonous)
- 2 The stick is by the rubber tree

5.1.1.1 Dialogue

I Am Hungry

(Believe Person (Physical-Fact (Person-State Hungry)))

*I Think You Need To Know That The Apples Are Not Good
So You Should Eat The Bananas*

(Requirement Eat-Food (((Food) (Banana))))

I Think You Need To Know That The Stick Is By The Rubber Tree
(Physical-Fact (Stick-Location Rubber-Tree))

Why Did You Tell Me That?

(Justify (Inform Person (Physical-Fact (Stick-Location Rubber-Tree))))

I Told You Where The Stick Was Because You Wanted To Know Its Location
(Goal Knowledge Person (Physical-Fact (Stick-Location Rubber-Tree))
Because (Know-State Stick-Location))

Further Justifications?

Yes

Because You Need The Stick To Knock Down The Bananas
(Goal Physical Person (Physical-Fact (Stick-Location Bottom-Banana-Tree))
Because (Want-Precondition (Knock-Down-Bananas)))

Further Justifications?

This approach falls short of the mark because

- it does not remove the past beliefs
- it does not create a plan to fix the situation because it assumes actions are easily reversible
- it only handles the situation where an explicit fault condition exists

However, the information necessary for implementing better debugging methods does exist in the plan tree. In particular the justifications and the record of the past world are the key things that would be used by a more complete debugging process.

The system simulates the person's planning process and realizes that the person does not have the following information and that this information is necessary to create a successful plan

- How to install the generator
- The fact that only the socket wrench should be used to tighten the generator bolts
- The fact that the socket wrench is in the corner

The system tells the person these things so that he can succeed in achieving his goal of installing the generator (see §4.4.5.4.7)

5.2.1.3 Expert Dialogue

This dialogue uses the expert mode of the system. This means that the system guides the person indirectly during the execution of the task since it waits for the person to ask what he needs to know, instead of giving him the information when he needs to know it

The Generator Is Not Installed

(Believe Person (Physical-Fact (Generator Not-Installed)))

I Think You Need To Know That To Install Generator You Must Lift It,
Then Tighten Bolts Then Tighten Screws

(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person (Install-Generator)

(Preconditions)

(Body

Serial-And

((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))

(Restrictions

(Believe Person (Physical-Fact (Generator Not-Installed)) Premise))

(Additions (Believe Person (Physical-Fact (Generator Installed))))

(Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))

Questions?

- 3 If the person is hungry he can satisfy his hunger by eating food
- 4 Food is defined to be an apple or a banana
- 5 The person can get a stick and use it to knock down bananas
- 6 There is a stick somewhere
- 7 The person knows whether he is hungry or not and the system knows that he knows.

The following facts are known only by the system

- 1 The apples are not good to eat even though they are classified as food (the reason is not stated, a good reason would be that they are poisonous)
- 2 The stick is by the rubber tree

5.1.1.1 Dialogue

I Am Hungry

(Believe Person (Physical-Fact (Person-State Hungry)))

I Think You Need To Know That The Apples Are Not Good

So You Should Eat The Bananas

(Requirement Eat-Food (((Food) (Banana))))

I Think You Need To Know That The Stick Is By The Rubber Tree

(Physical-Fact (Stick-Location Rubber-Tree))

Why Did You Tell Me That?

(Justify (Inform Person (Physical-Fact (Stick-Location Rubber-Tree))))

I Told You Where The Stick Was Because You Wanted To Know Its Location

(Goal Knowledge Person (Physical-Fact (Stick-Location Rubber-Tree))

Because (Know-State Stick-Location))

Further Justifications?

Yes

Because You Need The Stick To Knock Down The Bananas

(Goal Physical Person (Physical-Fact (Stick-Location Bottom-Banana-Tree))

Because (Want-Precondition (Knock-Down-Bananas)))

Further Justifications?

Yes

Because You Want To Eat The Bananas
 (Goal Physical Person (Physical-Fact (Banana-Location Bottom-Banana-Tree))
 Because (Want-Precondition (Eat-Food)))
 Further Justifications?

Yes

Because I Inferred You Want To Satisfy Your Hunger
 (Goal Physical Person (Physical-Fact (Person-State Not-Hungry))
 Because Inferred-Goal)
 Further Justifications?

Yes

Because You Told Me You Were Hungry
 (Believe Person (Physical-Fact (Person-State Hungry))
 Because (Inform System (Believe Person (Physical-Fact (Person-State Hungry))))

5.1.1.2 Explanation

The system accepts the person's statement that he is hungry as a valid inform speech act because the person knows whether he is hungry or not and the system does not. This is an undesirable state so the system infers that the person has a goal of satisfying his hunger (see §4.3.4.6).

The system simulates the person's planning process and realizes that the person does not have the following information and that this information is necessary to create a successful plan:

- The apples are not good to eat.
- The stick is by the rubber tree.

The system tells the person these things so that he can succeed in achieving the goal of satisfying his hunger (see §4.4.4.7).

The person asks the system to justify its action. The system takes the term "that" to mean the *last* action it has performed. The justifications consist of a chain of goals which lead back to the speech act performed by the person (§4.9)

5.2 Simple Generator Installation

5.2.1 Situation

The beliefs for this situation are in Appendix B. This is a version of the pump example of the TDUS system [Robinson, 80] which is described in detail in [Hendrix & Sacerdoti, 81]². The TDUS system has capabilities similar to ours and this example demonstrates that our system can equal and surpass these capabilities³. The following facts are known by both the person and the system.

1. There exists a generator, some bolts, some screws, a screwdriver, a socket wrench and an adjustable wrench.
2. The generator is not installed and this is an undesirable state.
3. The generator is currently at the bottom of the generator base.
4. The bolts and screws are at the top of the generator base.
5. The screwdriver is at the generator base.
6. The person knows how to fetch the wrenches and the screwdriver.
7. The person knows how to lift the generator, tighten bolts and tighten screws.

The following facts are known only by the system:

1. To install the generator it must be lifted on its base, the bolts must be tightened and then the screws must be tightened.
2. The socket wrench and the adjustable wrench are both in the corner.
3. Only the socket wrench should be used to tighten the generator bolts.

² see listing 6 of the article

³ at least in its reasoning capabilities

5.2.1.1 Novice Dialogue

This dialogue uses the novice mode of the system. This means the system guides the person directly during the execution of the task by telling him exactly what he needs to know, when he needs to know it.

The Generator Is Not Installed

(Believe Person (Physical-Fact (Generator Not-Installed)))

I Think You Need To Know That To Install The Generator You Must Lift It. Then Tighten The Bolts Then Tighten The Screws

(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person (Install-Generator) (Preconditions)

(Body

Serial-And

((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))

(Restrictions

(Believe Person (Physical-Fact (Generator Not-Installed)) Premise))

(Additions (Believe Person (Physical-Fact (Generator Installed))))

(Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))

Tell Me When You Have Lifted The Generator

Done

I Think You Need To Know That To Tighten The Generator Bolts

You Should Use A Socket Wrench

(Requirement Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))

I Think You Need To Know That The Socket Wrench Is In The Corner

(Physical-Fact (At-Socket-Wrench Corner))

5.2.1.2 Novice Explanation

The system does not accept the person's statement that the generator is not installed as an inform speech act. The reason is that the person believes the system already knows this fact. It is also the case that the generator not being installed is an undesirable state. Either of these two facts causes the system to infer that the person has a goal of installing the generator (see §4.3, §4.6).

The system simulates the person's planning process and realizes that the person does not have the following information and that this information is necessary to create a successful plan

- How to install the generator
- The fact that only the socket wrench should be used to tighten the generator bolts.
- The fact that the socket wrench is in the corner

The system tells the person these things so that he can succeed in achieving his goal of installing the generator (see §4.4.5.4.7)

5.2.1.3 Expert Dialogue

This dialogue uses the expert mode of the system. This means that the system guides the person indirectly during the execution of the task since it waits for the person to ask what he needs to know, instead of giving him the information when he needs to know it.

The Generator Is Not Installed

(Believe Person (Physical-Fact (Generator Not-Installed)))

I Think You Need To Know That To Install Generator You Must Lift It,
Then Tighten Bolts Then Tighten Screws

(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person (Install-Generator)

(Preconditions)

(Body

Serial-And

((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))

(Restrictions

(Believe Person (Physical-Fact (Generator Not-Installed)) Premise))

(Additions (Believe Person (Physical-Fact (Generator Installed))))

(Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))

Questions?

Which Wrench Should I Use To Tighten The Generator Bolts?

(Goal

Knowledge Person

(Believe

Person

(Requirement

Tighten-Generator-Bolts

((Wrench) <@ V2712 5571606 >@) <@ V2713 5571624 >@))))

I Assume That You Have Lifted The Generator

I Think You Need To Know That To Tighten The Generator Bolts

You Should Use A Socket Wrench

(Requirement Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))

Questions?

Where Is The Socket Wrench?

(Goal

Knowledge Person

(Believe Person (Physical-Fact (At-Socket-Wrench <@ V2722 5572175 >@))))

I Think You Need To Know That The Socket Wrench Is In The Corner

(Physical-Fact (At-Socket-Wrench Corner))

Questions?

Where Is The Screwdriver?

(Goal

Knowledge Person

(Believe Person (Physical-Fact (At-Screwdriver <@ V2723 5572211 >@))))

I Thought You Already Knew That The Screwdriver Is In The Corner

(Physical-Fact (At-Screwdriver Corner))

5.2.1.4 Expert Explanation

The system assumes that the person has performed certain actions if the person has asked something which implies that these actions have taken place. Thus the system assumes that the person has lifted the generator when he asks which wrench should be used to tighten the bolts because it believes the person would not want to know this unless the generator was on its base (see §4.3, §4.6).

If the system is asked something that it believes the person already knows it makes a comment. This explains the system's reply when the person asks where the screwdriver is located (see §4.3).

5.3 Inferring Generator Installation Goal

This is a sequence of examples that show the ability of the system to infer user goals. Only the first utterance in the dialogue is listed since this is what the system uses when inferring goals. In each case the inferred goal is that the person wants to install the generator and the processing would continue in a similar fashion to the previous example.

5.3.1 Situation One

The beliefs for this situation are the same as the situation in the previous example and are found in Appendix B. During the dialogue the inferred goal should actually be listed in the form (GOAL Person ?Goal ?Justification). It is treated in this fashion by the system but to maintain consistency with our listings in the Appendices we will not change our dialogue listings.

5.3.1.1 Dialogue

How Do I Get The Generator Installed?

(Goal

Knowledge Person

(Believe

Person

(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person <#:V2714 5571736 *> <#:V2715 5571754 *>

<#:V2716 5571772 *> <#:V2717 5572010 *>

(Additions (Believe Person (Physical-Fact (Generator Installed))))

<#:V2718 5572034 *> <#:V2719 5572052 *>))))))

I Think You Need To Know That To Install the Generator You
 Must Lift It Then Tighten The Bolts Then Tighten The Screws
 (ACTIONS-FOR-OBLIGATION Generator Physical
 ((Action-Description
 Physical Person
 (Install-Generator) (Preconditions)
 (Body
 Serial-And
 ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
 (Restrictions
 (Believe Person (Physical-Fact (Generator Not-Installed)) Premise))
 (Additions (Believe Person (Physical-Fact (Generator Installed))))
 (Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))
 I Inferred That You Want To Achieve A State With Generator Installed
 (Believe Person (Physical-Fact (Generator Installed)))
 If You Agree Say Yes

5.3.1.2 Explanation

The system accepts the person's goal as a valid goal since he does not have this information, so it tells the person what he wants to know. Since knowing how to do something often leads to wanting to do that thing the system infers that the person wants the generator installed (see §4.3. §4.6).

5.3.2 Situation Two

The beliefs for this situation are found in Appendix C. They are the same as the previous example with following exceptions.

- The person does not know (and the system does) where the screwdriver is located and does not know the description of the socket wrench.
- The person does know how to install the generator.

5.3.2.1 Dialogue

Where's The Screwdriver?

(Goal

Knowledge

Person (Believe Person (Physical-Fact (At-Screwdriver . <#:V0864 4352321 *>))))

I Think You Need To Know That The Screwdriver Is In The Corner
 (Physical-Fact (At-Screwdriver Corner))
 I Inferred That You Want To Execute The Action Of Install Generator
 (Execute-Action (Install-Generator))
 If You Agree Say Yes

5.3.2.2 Explanation

The system accepts the persons goal as a valid goal since he does not have this information so it tells the person what he wants to know. Knowing the screwdriver's location is a precondition of fetching the screwdriver, having the screwdriver is a result of fetching the screwdriver and is a precondition of tightening the generator screws. This is a part of the body of the action to install the generator so the system infers that the person wants to install the generator (see §4.3.4.6)

5.3.2.3 Dialogue

Is The Screwdriver At The Generator Base?
 (Goal Knowledge Person (Believe Person (Physical-Fact (At-Screwdriver Generator-Base))))

I Think You Need To Know That The Screwdriver Is In The Corner
 (Physical-Fact (At-Screwdriver-Corner))
 I Inferred That You Want To Execute The Action Of Install Generator
 (Execute-Action (Install-Generator))
 If You Agree Say Yes

5.3.2.4 Explanation

The system accepts the persons goal as a valid goal since he does not have this information, so it tells the person what he wants to know. Since knowing whether the screwdriver is at a particular location often leads to a goal of having the screwdriver at that location the system infers that the person wants the screwdriver at the generator base. This is a precondition of tightening the generator bolts which is part of the body of the action to install the generator, so the system infers the person wants the generator installed. (see §4.3.4.6).

5.3.2.5 Dialogue

What Is A Socket Wrench?

(Goal

Knowledge

Person (Believe Person (Description Socket-Wrench <#:V0854 4352000 +>)))

I Think You Need To Know That A Socket-Wrench Is

A Wrench With Removable Sockets

(Description Socket-Wrench (Wrench With Removable Sockets At End))

I Inferred That You Want To Execute The Action Of Install Generator

(Execute-Action (Install-Generator))

If You Agree Say Yes

Yes

Tell Me When You Have Lifted The Generator

Done

I Think You Need To Know That The Socket Wrench Is In The Corner

(Physical-Fact (At-Socket-Wrench Corner))

5.3.2.6 Explanation

The system accepts the persons goal as a valid goal since he does not have this information, so it tells the person what he wants to know. Since knowing the description of the socket wrench often leads to a goal of having the socket wrench at the place where the person is currently located the system infers the person wants the socket wrench to be at the generator base (the person's current location). This is a precondition of tightening the generator bolts which is part of a the body of the action to install the generator, so the system infers the person wants the generator installed. We include the rest of the execution to show that the system does eventually tell the person where the socket wrench is located which a human assistant would probably say right away (see §4.3, §4.6).

5.3.3 Situation Three

Consider the same situation as Situation Two but with the person not knowing how to install the generator. We do not list the beliefs in the Appendices since the only difference between this set of beliefs and those Appendix C is this one clause.

5.3.3.1 Dialogue

What Is A Socket Wrench?

(Goal

Knowledge Person

(Believe Person (Description Socket-Wrench <#:V0854 4352000 *>)))

I Think You Need To Know That A Socket-Wrench Is
A Wrench With Removable Sockets

(Description Socket-Wrench (Wrench With Removable Sockets At End))

I Inferred That You Want To Achieve A State With
The Socket Wrench At The Generator Base

(Believe Person (Physical-Fact (At-Socket-Wrench Generator-Base)))

If You Agree Say Yes

Yes

I Think You Need To Know That The Socket Wrench Is In The Corner

5.3.3.2 Explanation

The system accepts the persons goal as a valid goal since he does not have this information, so it tells the person what he wants to know. Since the person does not know how to install the generator the highest goal that can be inferred from this question is that the person wants the socket wrench to be at the generator base (which is where he is currently located). Since knowing where the socket wrench is located is a precondition of fetching the socket wrench the system gives the person this information (see §4.3, §4.6).

2 2
OF / DE

5.4 Complex Generator Installation

The beliefs for this situation are found in Appendix D. This example shows the way the system deals with assumptions and knowledge actions (actions whose purpose is to gain some knowledge, not to change the world). Though the example is based in part on the previous sequence of generator examples it demonstrates further capabilities of our system.

5.4.1 Situation

This is a variation of the generator scenario of the previous two examples with the following changes. The following extra information is known by both the person and the system.

1. The action to install the generator is the same as before except the last step in the body is now "attach the power cable" instead of "tighten generator screws" (this step still needs the screwdriver).
2. There is a forklift at the generator base.
3. The power cable can not be attached unless the power is known to be off.
4. It is not known whether the power is on or off; however, the person can check which is the case (we do not describe how).
5. The generator is heavy or light and there is no way of knowing which is the case by observation.
6. The person can lift the generator only if it is light.
7. There is a pump which is already installed.
8. The inferred fact "everything is done" is defined to mean that the pump is installed and the generator is installed.

The system believes the following facts which the person does not know:

1. The fork lift should be used to lift a heavy generator.
2. The generator is not installed.

3. The power switch is in the corner and can be used to turn off the power before attaching the power cable to the generator..
4. The generator shakes if the bolts are loose.

5.4.1.1 Dialogue

Is Everything Done?

(Goal Knowledge Person (Believe Person (Everything-Done)))

I Think You Need To Know That The Generator Is Not Installed
(Physical-Fact (Generator Not-Installed))

How Do I Get The Generator Installed?

(Goal

Knowledge Person

(Believe Person

(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person <#:V4889 7434027 *> <#:V4890 7434045 *>

<#:V4891 7434063 *> <#:V4892 7434101 *>

(Additions (Believe Person (Physical-Fact (Generator Installed))))

<#:V4893 7434125 *> <#:V4894 7434143 *>))))))

I Think You Need To Know That To Install The Generator
You Must Lift It Then Tighten The Bolts Then Attach The Power Cable
(Actions-For-Obligation

Generator Physical

((Action-Description

Physical Person (Install-Generator)

(Preconditions

(Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))

(Body Serial-And ((Tighten-Generator-Bolts) (Attach-Power-Cable)))

(Restrictions

(Believe

Person (Physical-Fact (Generator Not-Installed)) <#:V4900 7434462 *>))

(Additions (Believe Person (Physical-Fact (Generator Installed))))

(Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))

I Inferred That You Want To Achieve A State With The Generator Installed

If You Agree Say Yes

(Believe Person (Physical-Fact (Generator Installed)))

Yes

I Am Assuming That The Generator Is Light
(Generator-Type Light)

Did You Successfully Lift The Generator By Hand?

No

I Think You Need To Know That To Lift A Heavy Generator Use A Fork Lift

(Actions-For-Obligation

At-Generator Physical

(Action-Description

Physical Person

(Fork-Lift-Generator)

(Preconditions

(Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))

(Believe Person (Physical-Fact (At-Fork-Lift Generator-Base)))

(Body)

(Restrictions

(Believe

Person (Physical-Fact (Generator-Type Heavy)) <\$:V4902 7434516 *>))

(Additions

(Believe Person (Physical-Fact (At-Generator Generator-Base-Top)))

(Deletions

(Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))))

I Am Assuming That Generator-Type Heavy

(Generator-Type Heavy)

Did You Successfully Fork-Lift The Generator?

Yes

I Think You Need To Know That To Tighten The Generator Bolts You Should Use A Socket Wrench

(Requirement Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))

I Think You Need To Know That The Socket Wrench Is In The Corner

(Physical-Fact (At-Socket-Wrench Corner))

Is The Power In The State On?

Yes

I Think You Need To Know That To Turn The Power Off
You Should Use The Switch In The Corner

(Actions-For-Obligation

Power Physical

((Action-Description

Physical Person (Turn-Off-Power)

(Preconditions (Believe Person (Physical-Fact (At-Person Corner))))

(Body)

(Restrictions

(Believe Person (Physical-Fact (Power On)) <\$:V4901 7434500 *>))

(Additions (Believe Person (Physical-Fact (Power Off))))

(Deletions (Believe Person (Physical-Fact (Power On))))))

Well?

The Generator Shakes

(Believe Person (Generator-Shakes))

I Disagree And Thought You Did Too
Did You Do Tighten The Generator Bolts?

5.4.1.2 Explanation

When the person asks if everything is done he is really asking if the generator is installed. The reason is that this the state of this physical fact is what he needs to know to be able to answer that question (according to the definition of the inferred belief). When the person asks how to get the generator installed the system tells him and infers that he wants to install the generator (see §4.3, §4.6).

The system simulates the person's planning process and realizes that the person needs (in fact, *might* need) to know that:

- if the generator is heavy the person should use a forklift;
- if the power is on the person can turn it off by using the switch in the corner,
and
- to tighten the generator bolts he should use a socket wrench.)

This information is necessary for the person to create a successful plan to install the generator so the system gives it to the person (see §4.4, §4.9).

During the dialogue the system asks if the person has successfully lifted the generator. The reason is this action depends on a precondition which is that the generator is light, and this precondition is an assumption. The system verifies an assumption by asking whether the first action that depends on it was successful. The system also verifies the result of the person checking whether the power is on or off. The reason for these checks is that what the person needs to know depends on the outcome of these questions (see §4.7).

When the person says the generator shakes the system does not agree because it thought everything was done correctly. The system knows that the generator shakes if the bolts are loose so it asks the person if he has tightened the generator bolts (see §4.3, §4.7).

Conclusions

This chapter evaluates our work, describes some future improvements and summarizes everything.

6.1 Evaluation

Recently there has been some criticism of the methodology used in experimental AI work (see [McCarthy, 84] and [Ritchie & Hanna, 84]). In evaluating our work we will attempt to answer some of these criticisms, but first we will discuss an important principle which has guided our work.

This is the functional justification principle, by which we mean that every computational mechanism must be justified because it adds to the functionality (performance) of the system. This may seem self evident but it has a number of implications. The function of our system is to simulate a human expert in the task environment. To say that this goal has been achieved we would have to pass our version of the Turing test, which we can not do at this time. However, what we have done is take some typical dialogues (mostly from the TDUS system) that occur in this environment, and show that our system can engage in these dialogues. If any other mechanisms are to be added to the system then they must be justified in the same way as the original mechanisms; that is by the dialogues which make these mechanisms necessary.

The final decision as to whether a dialogue is typical depends on how often it occurs in an operational system. However, since we do not have such a system we depend on common sense and our previous experience with such environments to judge what "typical" means. It is important to relate new mechanisms to dialogues. If this is not done it is possible to justify the addition of any mechanism that appears to be plausible. We perform such a justification for each of our suggestions for future improvements to the system (where

these improvements are major ones).

To get back to our first point we believe that these criticisms of experimental AI work can be summed up in the following two requirements.

First, the theory and implementation should agree with each other. In their critique of the AM program (really the AM description) Ritchie and Hanna say:

"The central point in our criticism is the apparent discrepancy between published claims (about performance and techniques) and implementation details."

The reviewer of such work should clearly see that the claims made for the theory are implemented in the program. Another way of saying this is that the reviewer should have enough information to personally duplicate the program in order to verify the claims made for its performance.

We feel that our system passes this test since with PROLOG the structure of the program closely matches the description of the algorithms. Even though we use a modified PROLOG, an experienced programmer should be able to duplicate our program using standard PROLOG. We also feel that our description shows how the system functions in enough detail so that the reviewer can mentally verify that these theories are implemented in the program, and that the program can produce our examples. This was not the case with the original AM description which left one impressed with the performance, but unable to see how it was achieved from the description of the program.

Second, there should be a description of the class of problems that the program is capable of solving and the mechanisms embodied in the program should be shown to be necessary or sufficient for this class.

For our system the class of problems is the task environment. We relate this question of necessary and sufficient back to the idea of functionality. In other words, the mechanisms are necessary if there is a dialogue that the system should engage in that uses these

mechanisms. We have shown the mechanisms to be necessary by our example dialogues. To show that the mechanisms are completely sufficient (which we do not believe) would force us to pass the Turing Test. We describe some other mechanisms that we consider necessary for future work, along with the dialogues that justify them in the next section.

6.2 Future Work

6.2.1 Short Term

These are short term changes which are straightforward enough that they do not require justification by a dialogue. They are listed below.

1. Warning should be implemented in the manner described in §4.5.
2. Actions should be more general in the sense that they should work on elements of a hierarchy, not just on specific objects. For example, there should not be a separate fetch screwdriver and fetch wrench action. Instead, there should be only one fetch tool action since both these objects are tools.
3. Debugging should be expanded to cover all the different faults described in §4.9.

6.2.2 Long Term

These are things that are normally justified by a situation or a dialogue (unless they are of a structural nature). They are listed below.

1. The system should have goals of its own. This would force it to reason as to whether the person's goal conflicts with its own goals. Consider the situation where the person wants to kill himself and asks the system where the gun is located. If the system has a goal of keeping people alive it should not answer. This is a goal conflict situation which requires methods described in [Wilensky, 83] to detect and resolve.

2. The system should switch from a simple STRIPS planner to a more modern planner like NOAH. These planners try work in parallel until some constraint is noticed that forces an ordering of the actions in a plan. Consider the situation where the person asks whether a particular step can be done next, and if not, why not. If the ordering is discovered by the system during the planning process this question can be answered. If the ordering is arbitrarily given to the system this question can not be answered. An improved planner would make it possible for the system to be more of an assistant than a boss. An assistant realizes there are many ways to accomplish a goal, and leaves it to the person to choose one. A boss forces the person to do things in one way, which is the way he wants them done.
3. Different levels of expertise in the person should be modelled by changing what the system believes the person knows. This should be done in real time in the sense that evidence should accumulate for or against the current level of expertise assigned to the person. Consider the situation where the person says he is expert in a task but asks questions whose answers he should know. If these questions persist the system should downgrade the person's level of expertise.
4. The system should be able to handle multiple interacting agents. Consider the situation where there are two people working on a task and the system must guide both so that they work together properly. Very little work has been done in this area (see [Gregoff,83] for an exception) but it is necessary for realistic situations.
5. We have conveniently ignored the natural language parsing phase. This phase must be integrated into our system before it can be considered to be practical. Work has already been done in this area to see how various knowledge sources are used during the parsing process [Gershman, 79], and this work should be relevant.
6. The system should not accept everything that the person tells it as true. It may be that the person is wrong or is not telling the truth. Consider the situation where the person is dishonest and tells the system a lie for some ulterior motive. In order to be able to reason about these situations

the system needs to keep the person's beliefs completely separate from its own in order to allow contradictions between the two. Currently the system accepts what it believes the person believes as part of its own beliefs. It also needs to add more levels of belief, which must include at least what it believes the person believes the system believes. This level is necessary for counterplanning situations if the person is not cooperative.

7. The system is not able to know when the person is digressing, or has switched tasks. Consider the situation where the person has asked a question that implies he is changing tasks, or is not relevant to the current task. This problem (and some of the others listed) will require a change in the control structure from a centralized control to a distributed control. One way of doing this is by using object oriented programming with each object having a particular task. The theory is that all the objects *interacting together* would behave intelligently. The idea has promise but it is not clear how the objects should interact to produce intelligent behavior at a global level.

6.3 Summary

The system is not in any sense complete but does have some unique capabilities. These capabilities support the claims that are made about the adequacy of the underlying mechanisms that include the use of speech acts, plan recognition and plan simulation. The system must model the person's beliefs in order for these mechanisms to work properly. It could be extended in both its reasoning and parsing abilities and stands as a bridge between current natural language systems and current expert systems.

References

The following abbreviations are used throughout:

- AAAI-*i*   Proceedings of the *i*th American Artificial Intelligence Conference.
IJACI-*i*   Proceedings of the *i*th International Joint Conference on Artificial Intelligence.

- ALLEN, J., *A Plan Based Approach To Speech Act Recognition*,
TR 121/79, Department Of Computer Science,
University of Toronto, Toronto, 1979.
- APPELT, D.E., *Planning Natural Language Utterances to Satisfy Multiple Goals*,
Department of Computer Science,
Stanford University, 1980.
- AUSTIN, J.L., *How to Do Things With Words*,
Harvard University Press, 1962.
- CARBERRY, S., *Tracking User Goals in an Information Seeking Environment*,
AAAI-3, 1983.
- CLOCKSIN, W., MELLISH, C., *Programming in Prolog*,
Springer-Verlag, 1981.
- COHEN, P., *On Knowing What to Say, Planning Speech Acts*,
TR 118/79, Department Of Computer Science,
University of Toronto, Toronto, 1979.
- DYER, G., *Some Perceptions on Artificial Intelligence*,
Journal of Department of Computer Science,
UCLA, Volume 11, #2, 1983.
- FALETTI, J., *Pandora-A program for Doing Common Sense Planning
In Complex Situations*,
AAAI-2, 1982.
- FIKES, J., NILSSON, N., *Strips: A New Approach to the Application
of Theorem Proving to Problem Solving*,
Artificial Intelligence, Vol. 3, No. 4, 1972.
- GEORGEFF, M., *Communication and Interaction in Multi-Agent Planning*,
AAAI-3, 1983.
- GERSHMAN, A., *Knowledge Based Parsing*,
Yale Computer Science Research Report #156, Yale University, 1979.
- GROSZ, B.J., *The representation and use of Focus
In a System For Understanding Dialogues*,
IJACI-5, 1977.
- GROSZ, B.J., *Utterance and Objective: Issues in Natural Language Communication*,
The AI Magazine, Volume 1, #1, 1980.

- HENDRIX, J., SACERDOTI, E., *Natural Language Field in Perspective*,
BYTE Magazine, September 1981.
- HINTIKKA, J., *Knowledge and Belief*,
Cornell University Press, Ithica, 1962.
- KOWALSKI, R. *Logic For Problem Solving*,
North-Holland, 1979.
- KRIPKE, S., *Semantical Considerations on Modal Logic*,
In Linsky (Ed.), *Reference and Modality*,
Oxford University Press, London, 1971.
- MANN, W.C., *Towards a Speech Act Theory for Natural Language Processing*,
Information Sciences Institute, ISI/RR-79-75,
University of Southern California, 1980.
- MCCABE, F.G., CLARK, K.L., *Micro-Prolog 3.0 Programmer's Reference Manual*,
Logic Programming Associates, September, 1980.
- MCCARTHY, J., *Presidents Message: We Need Better Standards for AI Research*,
The AI Magazine, Volume 5, #3, 1984.
- MCDERMOTT, D., *Contexts and Data Dependencies: A Synthesis*,
IEEE Transactions on Pattern Analysis and Machine Intelligence,
Vol. 5, No. 3, MAY 1983.
- MOORE, R.C., *Reasoning about Knowledge and Action*,
SRI International, AI Center Technical Report,
No. 191, 1980.
- PERRAULT, R., COHEN, P., ALLEN, J., *Beyond Question Answering*,
in Lenhart (Ed.), *Strategies for Natural Language Processing*,
Lawrence Erlbaum Associates, 1982.
- RITCHIE, G.D., HANNA, F.K., *AM:A Case Study in AI Methodology*,
Artificial Intelligence, Vol. 23, P. 249-268, 1984.
- ROBINSON, A.E., *Interpreting Natural Language
Utterances in Dialogues about Tasks*,
Technical Note 210, SRI International,
Menlo Park CA, February 1980.
- SACERDOTI, E., *A Structure for Plans and Behaviour*,
North-Holland, New York, 1977.
- SACERDOTI, E., *Planning in a Hierarchy of Abstraction Spaces*,
Artificial Intelligence, Volume 5, 1974.
- SCHANK, R., *Inside Computer Understanding*,
San Francisco, W.H. Freeman, 1981.
- SEARLE, J.R., *Indirect Speech Acts*,
In Cole P., and Morgan, J. (Eds)
Syntax and Semantics, Volume 3, Academic Press, 1975.
- SHORTCLIFFE, E., *Computer Based Medical Consultations: MYCIN*,
Amsterdam Elsevier, 1976.

- SIDNER, J., *Recognizing Intended Meaning in Speakers Plans*,
IJACI-7, 1981.
- TURING, A.M., *Computing Machinery and Intelligence*,
In Feigenbaum, E. and Feldman, J. (Eds.)
Computers and Thought, NY:Mgraw-Hill, 1963.
- VERE, S., *Planning in Time: Windows and durations for Activities and Goals*,
Jet Propulsion Lab., Pasadena, CA. 1981.
- WALTZ, D., *The State of the Art in Natural Language Understanding*,
in Lenhart (Ed.), *Strategies for Natural Language Processing*,
Lawrence Erlbaum Associates, 1982.
- WARREN, D.H.D., *Warplan: A System For Generating Plans*,
DAI #76, Department of Artificial Intelligence,
University of Edinburgh, 1974.
- WILENSKY, R., *Talking to UNIX in English: An Overview of UC*,
Communications of the ACM, Volume 27, Number 6, 1984.
- WILENSKY, R., *Understanding Goal Based Stories*,
Yale Computer Science Research Report #140, Yale University, 1978.
- WILENSKY, R., *Planning and Understanding*,
A Computational Approach to Human Reasoning, Addison-Wesley, 1983.
- WILKINS, D.E., *Domain Independent Planning*,
Artificial Intelligence, Vol 22, No. 3, 1984.
- WINOGRAD, T., *Understanding Natural Language*,
New York, Academic Press, 1972.
- WINOGRAD, T., *Language as a Cognitive Process*,
Reading, Mass, Addison-Wesley.

Monkey And Bananas

A1 Beliefs

(Defun Beliefs ())

:: Systems Beliefs

(Pr (Remove-Clauses (Believe Obligation Undesired-State-Of-Obligation)))

:: Obligations On Physical Facts

((Obligation Apple-Location Top-Banana-Tree Bottom-Banana-Tree))

((Obligation Banana-Location Top-Banana-Tree Bottom-Banana-Tree))

((Obligation Stick-Location Rubber-Tree Bottom-Banana-Tree))

((Obligation Person-Location Rubber-Tree Bottom-Banana-Tree))

((Obligation Person-State Hungry Not-Hungry))

((Undesired-State-Of-Obligation Person-State Hungry))

:: System Beliefs

((Believe System (Requirement Eat-Food (((Food) (Banana)))) Premise))

((Believe System (Physical-Fact (Stick-Location Rubber-Tree)) Premise))

((Believe System (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification) ; System Agrees With Person
(Believe Person (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification)
(Non-Var ?Fact-State))

((Believe System (Actions-For-Obligation . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Actions-For-Obligation . ?Rest) ?Justification))

((Believe System (Requirements . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Requirements . ?Rest) ?Justification))

((Believe System (Reversible . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Reversible . ?Rest) ?Justification))

((Believe System (Description . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Description . ?Rest) ?Justification))

:: Persons Beliefs

((Believe Person (Physical-Fact (Person-State ?*)) Premise))

((Believe Person (Physical-Fact (Person-Location Bottom-Banana-Tree)) Premise))

((Believe Person (Physical-Fact (Banana-Location Top-Banana-Tree)) Premise))

((Believe Person (Physical-Fact (Apple-Location Bottom-Banana-Tree)) Premise))

```

((Believe Person
 (Actions-For-Obligation
  Person-Location Physical
  ((Action-Description
   Physical Person
   (Move-Self ?From ?To)
   (Preconditions (Believe Person (Physical-Fact (Person-Location ?From))))
   (Body)
   (Restrictions (Not Eq ?From ?To))
   (Additions (Believe Person (Physical-Fact (Person-Location ?To))))
   (Deletions (Believe Person (Physical-Fact (Person-Location ?From)))))) Premise))

```

```

((Believe Person
 (Actions-For-Obligation
  Stick-Location Physical
  ((Action-Description
   Physical Person
   (Get-Stick ?From ?To)
   (Preconditions (Believe Person (Physical-Fact (Stick-Location ?From))))
   (Believe Person (Physical-Fact (Person-Location ?From))))
   (Body)
   (Restrictions (Not Eq ?From ?To))
   (Additions
    (Believe Person (Physical-Fact (Stick-Location ?To)))
    (Believe Person (Physical-Fact (Person-Location ?To))))
   (Deletions
    (Believe Person (Physical-Fact (Stick-Location ?From)))
    (Believe Person (Physical-Fact (Person-Location ?From)))))) Premise))

```

```

((Believe Person
 (Actions-For-Obligation
  Person-State Physical
  ((Action-Description
   Physical Person
   (Eat-Food)
   (Preconditions (Believe Person (Physical-Fact (Person-Location ?Place)))
    (Believe Person (Physical-Fact (((Food) Location) ?Place))))
   (Body)
   (Restrictions (Believe Person (Physical-Fact (Person-State Hungry)) ?*))
   (Additions (Believe Person (Physical-Fact (Person-State Not-Hungry))))
   (Deletions (Believe Person (Physical-Fact (Person-State Hungry)))))) Premise))

```

```

((Believe Person
 (Actions-For-Obligation
  Banana-Location Physical
  ((Action-Description
   Physical Person
   (Knock-Down-Bananas)
   (Preconditions (Believe Person (Physical-Fact (Stick-Location Bottom-Banana-Tree)))
    (Believe Person (Physical-Fact (Person-Location Bottom-Banana-Tree))))
   (Body)
   (Restrictions (Believe Person (Physical-Fact (Banana-Location Top-Banana-Tree)) ?*))
   (Additions (Believe Person (Physical-Fact (Banana-Location Bottom-Banana-Tree))))
   (Deletions (Believe Person (Physical-Fact (Banana-Location Top-Banana-Tree)))))) Premise))

```

A2 Examples

```

(Pr (Main))
(Well)
% (I Am Hungry)
(Believe Person (Physical-Fact (Person-State Hungry)))
(Ok)
System
(Inform Person (Prohibition Eat-Food (((Food) (Banana)))))
System
(Inform Person (Physical-Fact (Stick-Location Rubber-Tree)))
Person
(Move-Self Bottom-Banana-Tree Rubber-Tree)
Person
(Get-Stick Rubber-Tree Bottom-Banana-Tree)
Person
(Knock-Down-Bananas)
Person
(Eat-Food)
(Prohibition Eat-Food (((Food) (Banana)))))
(I Think You Need To Know That The Apples Are Not Good So You Should Eat The Bananas)
(Physical-Fact (Stick-Location Rubber-Tree))
(I Think You Need To Know That The Stick Is By The Rubber Tree)
(Well)
% (Why Did You Tell Me That)
(Justify (Inform Person (Physical-Fact (Stick-Location Rubber-Tree)))))
(Goal Knowledge Person (Physical-Fact (Stick-Location Rubber-Tree)) Because (Know-State Stick-Location))
(Further Justifications)
% Yes

(Goal Physical Person (Physical-Fact (Stick-Location Bottom-Banana-Tree)) Because (Want-Precondition (Knock-Down-
Bananas)))
(Further Justifications)
% Yes

(Goal Physical Person (Physical-Fact (Banana-Location Bottom-Banana-Tree)) Because (Want-Precondition (Eat-Food)))
(Further Justifications)
% Yes

(Goal Physical Person (Physical-Fact (Person-State Not-Hungry)) Because Inferred-Goal)
(Further Justifications)
% Yes

(Believe Person (Physical-Fact (Person-State Hungry)) Because (Inform System (Believe Person (Physical-Fact (Person-
State Hungry)))))
(Well)
%
Back To Dribbling Lisp Listener.

(Dribble-End)

```

Simple Generator Installation

B1 Beliefs

(Defun Beliefs ())

:: Systems Beliefs

((Pr (Remove-Clauses (Believe Obligation Undesired-State-Of-Obligation))))

:: Obligations On Physical Facts

((Obligation At-Person Corner Pump-Base Generator-Base Roof))

((Obligation At-Socket-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Adjustable-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Screwdriver Corner Pump-Base Generator-Base Roof))

((Obligation At-Generator Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Generator-Bolts Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Generator-Screws Generator-Base-Bottom Generator-Base-Top))

((Obligation Generator-Bolts Tight Loose))

((Obligation Generator Installed Not-Installed))

((Undesired-State-Of-Obligation Generator Not-Installed))

((Obligation Generator-Screws Loose Tight))

:: Systems Private Beliefs

((Believe System (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))) Premise))

((Believe System (Physical-Fact (At Socket-Wrench Corner)) Premise))

((Believe System (Physical-Fact (At-Adjustable-Wrench Corner)) Premise))

((Believe System
 (Actions-For-Obligation
 Generator Physical
 ((Action-Description
 Physical Person
 (Install-Generator)
 (Preconditions)
 (Body Serial-And ((Lit-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
 (Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ?*))
 (Additions (Believe Person (Physical-Fact (Generator Installed))))
 (Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))) Premise))

((Believe System (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification) ; System Agrees With Person
 (Believe Person (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification)
 (Non-Var ?Fact-State))

```
((Believe System (Actions-For-Obligation . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Actions-For-Obligation . ?Rest) ?Justification))
```

```
((Believe System (Prohibitions . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Prohibitions . ?Rest) ?Justification))
```

```
((Believe System (Reversible . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Reversible . ?Rest) ?Justification))
```

```
((Believe System (Description . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Description . ?Rest) ?Justification))
```

```
:: Mutual Beliefs
```

```
((Believe ?Everyone (Generator-Shakes)
((Believe ?Everyone (Physical-Fact (Generator-Bolts Loose))))))
(Believe ?Everyone (Physical-Fact (Generator-Bolts Loose)) ?*)
```

```
((Believe ?Everyone (Generator-Shakes)
((Believe ?Everyone (Physical-Fact (Generator-Screws Loose))))))
(Believe ?Everyone (Physical-Fact (Generator-Screws Loose)) ?*)
```

```
:: Persons Beliefs
```

```
((Believe Person (Physical-Fact (At-Screwdriver Corner)) Premise))
```

```
((Believe Person (Physical-Fact (At-Person Pump-Base)) Premise))
```

```
((Believe Person
  (Actions-For-Obligation
    At-Person Physical
    ((Action-Description
      Physical Person
      (Move-Self ?From ?To)
      (Preconditions (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```
((Believe Person
  (Actions-For-Obligation
    At-Socket-Wrench Physical
    ((Action-Description
      Physical Person
      (Get-Socket-Wrench ?From ?To)
      (Preconditions (Believe Person (Physical-Fact (At-Socket-Wrench ?From))))
      (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Socket-Wrench ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Socket-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```
((Believe Person
  (Actions-For-Obligation
    At-Adjustable-Wrench Physical
    ((Action-Description
      Physical Person
      (Get-Adjustable-Wrench ?From ?To)
      (Preconditions
        (Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Adjustable-Wrench ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```

((Believe Person
  (Actions-For-Obligation
    At-Screwdriver Physical
    ((Action-Description
      Physical Person
      (Get-Screwdriver ?From ?To)
      (Preconditions (Believe Person (Physical-Fact (At-Screwdriver ?From)))
        (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Screwdriver ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Screwdriver ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

;; Generator
((Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)) Premise))
((Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)) Premise))

((Believe Person (Physical-Fact (Generator-Bolts Loose)) Premise))

((Believe Person (Physical-Fact (Generator Not-Installed)) Premise))

((Believe Person (Physical-Fact (At-Generator-Screws Generator-Base-Top)) Premise))

((Believe Person (Physical-Fact (Generator-Screws Loose)) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Generator Physical
    ((Action-Description
      Physical Person
      (Lift Generator)
      (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))
      (Body)
      (Restrictions)
      (Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
      (Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))))) Premise))

((Believe Person (Reversible Tighten-Generator-Bolts No) Premise))

((Believe Person (Description Socket-Wrench (Wrench With Removable Sockets At End))))

((Believe Person
  (Actions-For-Obligation
    Generator-Bolts Physical
    ((Action-Description
      Physical Person
      (Tighten-Generator-Bolts)
      (Preconditions (Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)))
        (Believe Person (Physical-Fact (((Wrench) Location) Generator-Base))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Bolts Loose)) ?*))
      (Additions (Believe Person (Physical-Fact (Generator-Bolts Tight))))
      (Deletions (Believe Person (Physical-Fact (Generator-Bolts Loose)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    Generator-Screws Physical
    ((Action-Description Physical Person
      (Tighten-Generator-Screws)
      (Preconditions (Believe Person (Physical-Fact (At-Generator-Screws Generator-Base-Top)))
        (Believe Person (Physical-Fact (At-Screwdriver Generator-Base))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Screws Loose)) ?*))
      (Additions (Believe Person (Physical-Fact (Generator-Screws Tight))))
      (Deletions (Believe Person (Physical-Fact (Generator-Screws Loose)))))) Premise))

```

B2 Examples

```

(Pr (Main))
(Well)
% (The Generator Is Not Installed)
(Believe Person (Physical-Fact (Generator Not-Installed)))
System
(Inform Person (Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) i#:V2707 5612760 *2))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))))
Person
(Lift-Generator)
System
(Inform Person (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))))
System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Pump-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
Person
(Tighten-Generator-Bolts)
Person
(Move-Self Generator-Base Corner)
Person
(Get-Screwdriver Corner Generator-Base)
Person
(Tighten-Generator-Screws)
Person
(Install-Generator)
(Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) Premise))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))
(I Think You Need To Know That To Install Generator You Must Lift It Then Tighten Bolts Then Tighten Screws)
(Tell Me When You Have Done Action (Lift-Generator))
% Done

(Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))))
(I Think You Need To Know That To Tighten The Generator Bolts You Should Use A Socket Wrench)
(Physical-Fact (At-Socket-Wrench Corner))
(I Think You Need To Know That The Socket Wrench Is In The Corner)
(Well)
%
Back To Dribbling Lisp Listener.

(Dribble-End)

```

```

(Pr (Expert-Model))
More? (Y Or N) No
Nil
(Pr (Main))
(Well)
% (The Generator Is Not Installed)
(Believe Person (Physical-Fact (Generator Not-Installed)))
System
(Inform Person (Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed))) ;#:V2707 7312404 *;))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))))))
Person
(Lift-Generator)
System
(Inform Person (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))))
System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Pump-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
Person
(Tighten-Generator-Bolts)
Person
(Move-Self Generator-Base Corner)
Person
(Get-Screwdriver Corner Generator-Base)
Person
(Tighten-Generator-Screws)
Person
(Install-Generator)
(Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed))) Premise))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))))))
(I Think You Need To Know That To Install Generator You Must Lift It Then Tighten Bolts Then Tighten Screws)
(Questions)
% (Which Wrench Should I Use To Tighten The Generator Bolts)
(Goal Knowledge Person (Believe Person (Prohibition Tighten-Generator-Bolts (((Wrench) ;#:V2712 5571606 *;) . ;#:V2713
5571624 *;))))
(I Assumed That You Have Done Action Lift-Generator)
(Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))))
(I Think You Need To Know That To Tighten The Generator Bolts You Should Use A Socket Wrench)
(Questions)
% (Where Is The Socket Wrench)
(Goal Knowledge Person (Believe Person (Physical-Fact (At-Socket-Wrench . ;#:V2722 5572173 *;))))
(Physical-Fact (At-Socket-Wrench Corner))
(I Think You Need To Know That The Socket Wrench Is In The Corner)
(Questions)
% (Where Is The Screwdriver)
(Goal Knowledge Person (Believe Person (Physical-Fact (At-Screwdriver . ;#:V2723 5572211 *;))))
(Physical-Fact (At-Screwdriver Corner))
(I Thought You Already Knew The Screwdriver Is In The Corner)
(Questions)
Back To Dribbling Lisp Listener.

(Dribble-End)

```


Inferring Generator Installation Goals

C1 Beliefs

((Defun Beliefs (

;; Systems Beliefs

((Pr (Remove-Clauses (Believe Obligation Undesired-State-Of-Obligation)))

;; Obligations On Physical Facts

((Obligation At-Person Corner Pump-Base Generator-Base Roof))

((Obligation At-Socket-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Adjustable-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Screwdriver Corner Pump-Base Generator-Base Roof))

((Obligation At-Generator Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Generator-Bolts Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Generator-Screws Generator-Base-Bottom Generator-Base-Top))

((Obligation Generator-Bolts Tight Loose))

((Obligation Generator Installed Not-Installed))

((Undesired-State-Of Obligation Generator Not-Installed))

((Obligation Generator-Screws Loose Tight))

;; Systems Private Beliefs

((Believe System (Description Socket-Wrench (Wrench With Removable Sockets At End))))

((Believe System (Physical-Fact (At-Screwdriver Corner)) Premise))

((Believe System (Physical-Fact (At-Socket-Wrench Corner)) Premise))

((Believe System (Physical-Fact (At-Adjustable-Wrench Corner)) Premise))

((Believe System (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification) ; System Agrees With Person
(Believe Person (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification)
(Non-Var ?Fact-State))

((Believe System (Actions-For-Obligation . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Actions-For-Obligation . ?Rest) ?Justification))

((Believe System (Prohibitions . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Prohibitions . ?Rest) ?Justification))

((Believe System (Reversible . ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Reversible . ?Rest) ?Justification))

```
((Believe System (Description : ?Rest) ?Justification) ; System Agrees With Person
(Believe Person (Description : ?Rest) ?Justification))
```

```
:: Mutual Beliefs
```

```
((Believe ?Everyone (Generator-Shakes)
((Believe ?Everyone (Physical-Fact (Generator-Bolts Loose))))
(Believe ?Everyone (Physical-Fact (Generator-Bolts Loose)) ?*))
```

```
((Believe ?Everyone (Generator-Shakes)
((Believe ?Everyone (Physical-Fact (Generator-Screws Loose))))
(Believe ?Everyone (Physical-Fact (Generator-Screws Loose)) ?*))
```

```
:: Persons Beliefs
```

```
((Believe Person (Physical-Fact (At-Person Generator-Base)) Premise))
```

```
((Believe Person
(Actions-For-Obligation
At-Person Physical
((Action-Description
Physical Person
(Move-Self ?From ?To)
(Preconditions (Believe Person (Physical-Fact (At-Person ?From))))
(Body)
(Restrictions (Not Eq ?From ?To))
(Additions (Believe Person (Physical-Fact (At-Person ?To))))
(Deletions (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```
((Believe Person
(Actions-For-Obligation
At-Socket-Wrench Physical
((Action-Description
Physical Person
(Get-Socket-Wrench ?From ?To)
(Preconditions
(Believe Person (Physical-Fact (At-Socket-Wrench ?From)))
(Believe Person (Physical-Fact (At-Person ?From))))
(Body)
(Restrictions (Not Eq ?From ?To))
(Additions
(Believe Person (Physical-Fact (At-Socket-Wrench ?To)))
(Believe Person (Physical-Fact (At-Person ?To))))
(Deletions
(Believe Person (Physical-Fact (At-Socket-Wrench ?From)))
(Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```
((Believe Person
(Actions-For-Obligation
At-Adjustable-Wrench Physical
((Action-Description
Physical Person
(Get-Adjustable-Wrench ?From ?To)
(Preconditions
(Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
(Believe Person (Physical-Fact (At-Person ?From))))
(Body)
(Restrictions (Not Eq ?From ?To))
(Additions
(Believe Person (Physical-Fact (At-Adjustable-Wrench ?To)))
(Believe Person (Physical-Fact (At-Person ?To))))
(Deletions
(Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
(Believe Person (Physical-Fact (At-Person ?From)))))) Premise))
```

```
((Believe Person
(Actions-For-Obligation
At-Screwdriver Physical
((Action-Description
Physical Person
(Get-Screwdriver ?From ?To)
(Preconditions (Believe Person (Physical-Fact (At-Screwdriver ?From)))
(Believe Person (Physical-Fact (At-Person ?From))))
(Body)
(Restrictions (Not Eq ?From ?To))
```

```

(Additions
  (Believe Person (Physical-Fact (At-Screwdriver ?To)))
  (Believe Person (Physical-Fact (At-Person ?To))))
(Deletions
  (Believe Person (Physical-Fact (At-Screwdriver ?From)))
  (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

;; Generator

((Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)) Premise))

((Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)) Premise))

((Believe Person (Physical-Fact (Generator-Bolts Loose)) Premise))

((Believe Person (Physical-Fact (Generator Not-Installed)) Premise))

((Believe Person (Physical-Fact (At-Generator-Screws Generator-Base-Top)) Premise))

((Believe Person (Physical-Fact (Generator-Screws Loose)) Premise))

((Believe Person
  (Actions-For-Obligation
    Generator-Screws Physical
    ((Action-Description
      Physical Person
      (Tighten-Generator-Screws)
      (Preconditions
        (Believe Person (Physical-Fact (At-Generator-Screws Generator-Base-Top)))
        (Believe Person (Physical-Fact (At-Screwdriver Generator-Base))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Screws Loose)) ?*))
      (Additions (Believe Person (Physical-Fact (Generator-Screws Tight))))
      (Deletions (Believe Person (Physical-Fact (Generator-Screws Loose)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    Generator Physical
    ((Action-Description
      Physical Person
      (Install-Generator)
      (Preconditions
        (Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws)))
        (Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ?*))
        (Additions (Believe Person (Physical-Fact (Generator Installed))))
        (Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Generator Physical
    ((Action-Description
      Physical Person
      (Lift-Generator)
      (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))
      (Body)
      (Restrictions)
      (Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
      (Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))))) Premise))

((Believe Person (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))) Premise))

((Believe Person (Reversible Tighten-Generator-Bolts No) Premise))

((Believe Person
  (Actions-For-Obligation
    Generator-Bolts Physical
    ((Action-Description Physical Person
      (Tighten-Generator-Bolts)
      (Preconditions (Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)))
      (Believe Person (Physical-Fact (((Wrench) Location) Generator-Base))))
      (Body)

```

(Restrictions (Believe Person (Physical-Fact (Generator-Bolts Loose)) ?*))
(Additions (Believe Person (Physical-Fact (Generator-Bolts Tight))))
(Deletions (Believe Person (Physical-Fact (Generator-Bolts Loose)))))) Premise))

C2 Examples

```

(Pr (Main))
(Well)
% (How Do I Get The Generator Installed)
(Goal Knowledge Person (Believe Person (Actions-For-Obligation Generator Physical
((Action-Description Physical Person ;#:V2714 5571736 *; #:V2715 5571754 *; #:V2716 5571772 *; #:V2717 5572010
(Additions (Believe Person (Physical-Fact (Generator Installed)))) ;#:V2718 5572034 *; #:V2719 5572052 *))))))
(Hm...)
System
(Inform Person (Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws))))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ;#:V2707 7416350 *;))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))) Nil)))
(Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator) (Preconditions)
(Body Serial-And ((Lift-Generator) (Tighten-Generator-Bolts) (Tighten-Generator-Screws))))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) Premise))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))) Nil)))
(I Think You Need To Know That To Install Generator You Must Lift It Then Tighten Bolts Then Tighten Screws)
(Believe Person (Physical-Fact (Generator Installed)))
(I Inferred That You Want To Achieve State With Generator Installed)
(If You Agree Say Yes)
%
Back To Dribbling LISP Listener.

```

```

(Dribble-End)

```

```

(Init)
(T)
(Pr (Main))
(Well)
% (Where Is The Screwdriver)
(Goal Knowledge Person (Believe Person (Physical-Fact (At-Screwdriver :V0864 4352321 *))))
(Hm...)
System
(Inform Person (Physical-Fact (At-Screwdriver Corner)))
(Physical-Fact (At-Screwdriver Corner))
(I Think You Need To Know That The Screwdriver Is In The Corner)
(Execute-Action (Install-Generator))
(I Inferred That You Want To Execute The Action Of Install Generator)
(If You Agree Say Yes)
%
Back To Dribbling Lisp Listener.

```

```

(Init)
(T)
(Pr (Main))
(Well)
% (Is The Screwdriver At The Generator Base)
(Goal Knowledge Person (Believe Person (Physical-Fact (At-Screwdriver Generator-Base))))
(Hm...)
System
(Inform Person (Physical-Fact (At-Screwdriver Corner)))
(Physical-Fact (At-Screwdriver Corner))
(I Think You Need To Know That The Screwdriver Is In The Corner)
(Execute-Action (Install-Generator))
(I Inferred That You Want To Execute The Action Of Install Generator)
(If You Agree Say Yes)
%
Back To Dribbling Lisp Listener.

```

```

(Init)
(T)
(Pr (Main))
(Well)
% (What Is A Socket Wrench)
(Goal Knowledge Person (Believe Person (Description Socket-Wrench :V0854 4352000 *))))
(Hm...)
System
(Inform Person (Description Socket-Wrench (Wrench With Removable Sockets At End)))
(Description Socket-Wrench (Wrench With Removable Sockets At End))
(I Think You Need To Know That A Socket-Wrench Is Wrench With Removable Sockets At End)
(Execute-Action (Install-Generator))
(I Inferred That You Want To Execute The Action Of Install Generator)
(If You Agree Say Yes)
% Yes

```

```

Person
(Lift-Generator)
System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Generator-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
Person
(Tighten-Generator-Bolts)
System
(Inform Person (Physical-Fact (At-Screwdriver Corner)))
Person
(Move-Self Generator-Base Corner)
Person
(Get-Screwdriver Corner Generator-Base)
Person
(Tighten-Generator-Screws)
(Tell Me When You Have Done Action (Lift-Generator))
% Done

```

```

(Physical-Fact (At-Socket-Wrench Corner))
(I Think You Need To Know That The Socket Wrench Is In The Corner)
(Tell Me When You Have Done Action (Tighten-Generator-Bolts))
% Done

```

```

(Physical-Fact (At-Screwdriver Corner))
(I Think You Need To Know That The Screwdriver Is In The Corner)
(Well)
%
Back To Dribbling Lisp Listener.

```

```
(Init)
(T)
(Pr (Main))
(Well)
% (What Is A Socket Wrench)
(Goal Knowledge Person (Believe Person (Description Socket-Wrench (#:V0854 4352000 *2)))
(Hm..))
System
(Inform Person (Description Socket-Wrench (Wrench With Removable Sockets At End)))
(Description Socket-Wrench (Wrench With Removable Sockets At End))
(I Think You Need To Know That A Socket-Wrench Is Wrench With Removable Sockets At End)
(Believe Person (Physical-Fact (At-Socket-Wrench Generator-Base)))
(I Inferred That You Want To Achieve State With Socket Wrench Is At The Generator Base)
(If You Agree Say Yes)
% Yes

System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Generator-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
(Physical-Fact (At-Socket-Wrench Corner))
(I Think You Need To Know That The Socket Wrench Is In The Corner)
(Well)
%
Back To Dribbling Lisp Listener.

(Dribble-End)
```

Complex Generator Installation

D1 Beliefs

(Defun Beliefs ())

:: Systems Beliefs

(Pr (Remove-Clauses (Believe Obligation Undesired-State-Of-Obligation)))

:: Obligations On Physical Facts

((Obligation At-Person Corner Pump-Base Generator-Base Roof))

((Obligation At-Socket-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Adjustable-Wrench Corner Pump-Base Generator-Base Roof))

((Obligation At-Fork-Lift Corner Pump-Base Generator-Base Roof))

((Obligation At-Screwdriver Corner Pump-Base Generator-Base Roof))

((Obligation At-Generator Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Generator-Bolts Generator-Base-Bottom Generator-Base-Top))

((Obligation At-Power-Cable Corner Pump-Base Generator-Base Roof))

((Obligation Power-Cable Attached Not-Attached))

((Obligation Generator-Type Heavy Light))

((Obligation Generator-Bolts Tight Loose))

((Obligation Generator Installed Not-Installed))

((Undesired-State-Of-Obligation Generator Not-Installed))

((Obligation Pump Installed Not-Installed))

((Obligation Power On Off))

:: Systems Private Beliefs

((Believe System (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))) Premise))

((Believe System (Physical-Fact (Generator Not-Installed)) Premise))

((Believe System (Physical-Fact (At-Socket-Wrench Corner)) Premise))

((Believe System (Physical-Fact (At-Adjustable-Wrench Corner)) Premise))

((Believe System
(Actions-For-Obligation
Generator Physical


```

((Action-Description
  Physical Person
  (Install-Generator)
  (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
  (Body Serial-And ((Tighten-Generator-Bolts) (Attach-Power-Cable)))
  (Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ?*))
  (Additions (Believe Person (Physical-Fact (Generator Installed))))
  (Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))) Premise))

((Believe System (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification) ; System Agrees With Person
 (Believe Person (Physical-Fact (?Fact-Head ?Fact-State)) ?Justification)
 (Non-Var ?Fact-State))

((Believe System (Actions-For-Obligation . ?Rest) ?Justification) ; System Agrees With Person
 (Believe Person (Actions-For-Obligation . ?Rest) ?Justification))

((Believe System (Prohibitions . ?Rest) ?Justification) ; System Agrees With Person
 (Believe Person (Prohibitions . ?Rest) ?Justification))

((Believe System (Reversible . ?Rest) ?Justification) ; System Agrees With Person
 (Believe Person (Reversible . ?Rest) ?Justification))

((Believe System (Description . ?Rest) ?Justification) ; System Agrees With Person
 (Believe Person (Description . ?Rest) ?Justification))

;; Mutual Beliefs

((Believe ?Everyone (Generator-Shakes)
 ((Believe ?Everyone (Physical-Fact (Generator-Bolts Loose))))
 (Believe ?Everyone (Physical-Fact (Generator-Bolts Loose)) ?*))

((Believe ?Everyone (Generator-Shakes)
 ((Believe ?Everyone (Physical-Fact (Generator-Screws Loose))))
 (Believe ?Everyone (Physical-Fact (Generator-Screws Loose)) ?*))

((Believe ?Everyone (Everything-Done)
 ((Believe ?Everyone (Physical-Fact (Generator Installed)))
 (Believe ?Everyone (Physical-Fact (Pump Installed)))))
 (Believe ?Everyone (Physical-Fact (Generator Installed)) ?*)
 (Believe ?Everyone (Physical-Fact (Pump Installed)) ?*))

;; Persons Beliefs

((Believe Person (Physical-Fact (At-Screwdriver Generator-Base)) Premise))

((Believe Person (Physical-Fact (At-Person Pump-Base)) Premise))

((Believe Person (Physical-Fact (At-Fork-Lift Generator-Base)) Premise))

((Believe Person
 (Actions-For-Obligation
  Power-Cable Physical
  ((Action-Description
    Physical Person
    (Attach-Power-Cable)
    (Preconditions (Believe Person (Physical-Fact (Power Off)))
      (Believe Person (Physical-Fact (At-Power-Cable Generator-Base)))
      (Believe Person (Physical-Fact (At-Screwdriver Generator-Base))))
    (Body)
    (Restrictions (Believe Person (Physical-Fact (Power-Cable Not-Attached)) ?*))
    (Additions (Believe Person (Physical-Fact (Power-Cable Attached))))
    (Deletions (Believe Person (Physical-Fact (Power-Cable Not-Attached))))) Premise))

((Believe System
 (Actions-For-Obligation
  Power Physical
  ((Action-Description
    Physical Person
    (Turn-Off-Power)
    (Preconditions (Believe Person (Physical-Fact (At-Person Corner))))
    (Body)

```

```

(Restrictions (Believe Person (Physical-Fact (Power On)) ?*))
(Additions (Believe Person (Physical-Fact (Power Off))))
(Deletions (Believe Person (Physical-Fact (Power On)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    Power Knowledge
    ((Action-Description
      Knowledge Person
      (Check Power On)
      (Preconditions
        (Believe Person (Physical-Fact (At-Power-Cable ?Place)))
        (Believe Person (Physical-Fact (At-Person ?Place))))
      (Body)
      (Restrictions)
      (Additions (Believe Person (Physical-Fact (Power On))))
      (Deletions (Believe Person (Physical-Fact (Power ?*))))))
    (Action-Description Knowledge Person
      (Check Power Off)
      (Preconditions
        (Believe Person (Physical-Fact (At-Power-Cable ?Place)))
        (Believe Person (Physical-Fact (At-Person ?Place))))
      (Body)
      (Restrictions)
      (Additions (Believe Person (Physical-Fact (Power Off))))
      (Deletions (Believe Person (Physical-Fact (Power ?*)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Person Physical
    ((Action-Description
      Physical Person
      (Move-Self ?From ?To)
      (Preconditions (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Socket-Wrench Physical
    ((Action-Description
      Physical Person
      (Get-Socket-Wrench ?From ?To)
      (Preconditions
        (Believe Person (Physical-Fact (At-Socket-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Socket-Wrench ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Socket-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Adjustable-Wrench Physical
    ((Action-Description
      Physical Person
      (Get-Adjustable-Wrench ?From ?To)
      (Preconditions (Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Adjustable-Wrench ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Adjustable-Wrench ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Screwdriver Physical
    ((Action-Description
      Physical Person

```

```

(Get-Screwdriver ?From ?To)
(Preconditions
  (Believe Person (Physical-Fact (At-Screwdriver ?From)))
  (Believe Person (Physical-Fact (At-Person ?From))))
(Body)
(Restrictions (Not Eq ?From ?To))
(Additions
  (Believe Person (Physical-Fact (At-Screwdriver ?To)))
  (Believe Person (Physical-Fact (At-Person ?To))))
(Deletions
  (Believe Person (Physical-Fact (At-Screwdriver ?From)))
  (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Forklift Physical
    ((Action-Description
      Physical Person
      (Get-Forklift ?From ?To)
      (Preconditions
        (Believe Person (Physical-Fact (At-Forklift ?From)))
        (Believe Person (Physical-Fact (At-Person ?From))))
      (Body)
      (Restrictions (Not Eq ?From ?To))
      (Additions
        (Believe Person (Physical-Fact (At-Forklift ?To)))
        (Believe Person (Physical-Fact (At-Person ?To))))
      (Deletions
        (Believe Person (Physical-Fact (At-Forklift ?From)))
        (Believe Person (Physical-Fact (At-Person ?From)))))) Premise))

;; Generator

((Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)) Premise))

((Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)) Premise))

((Believe Person (Physical-Fact (Generator-Bolts Loose)) Premise))

((Believe Person (Physical-Fact (At-Power-Cable Generator-Base)) Premise))

((Believe Person (Physical-Fact (Power-Cable Not-Attached)) Premise))

((Believe Person
  (Actions-For-Obligation
    At-Generator Physical
    ((Action-Description
      Physical Person
      (Person-Lift-Generator)
      (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Type Light)) ?*))
      (Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
      (Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))))) Premise))

((Believe System
  (Actions-For-Obligation
    At-Generator Physical
    ((Action-Description
      Physical Person
      (Fork-Lift-Generator)
      (Preconditions
        (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))
        (Believe Person (Physical-Fact (At-Fork-Lift Generator-Base))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Type Heavy)) ?*))
      (Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
      (Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))))) Premise))

((Believe Person (Reversible Tighten-Generator-Bolts No) Premise))

((Believe Person (Description Socket-Wrench (Wrench With Removable Sockets At End))))

((Believe Person

```

```

(Actions-For-Obligation
  Generator-Bolts Physical
  ((Action-Description Physical Person
    (Tighten-Generator-Bolts)
    (Preconditions
      (Believe Person (Physical-Fact (At-Generator-Bolts Generator-Base-Top)))
      (Believe Person (Physical-Fact (((Wrench) Location) Generator-Base))))
    (Body)
    (Restrictions (Believe Person (Physical-Fact (Generator-Bolts Loose)) ?*))
    (Additions (Believe Person (Physical-Fact (Generator-Bolts Tight))))
    (Deletions (Believe Person (Physical-Fact (Generator-Bolts Loose)))))) Premise))

```

```

:: Pump
::

```

```

((Believe Person (Physical-Fact (Pump Installed)) Premise))

```

```

)

```

D2 Examples

```

(Pr (Main))
(Well)
% (Is Everything Done)
(Goal Knowledge Person (Believe Person (Everything-Done)))
(Hm.)
System
(Inform Person (Physical-Fact (Generator Not-Installed)))
(Physical-Fact (Generator Not-Installed))
(I Think You Need To Know That The Generator Is Not Installed)
(Well)
% (How Do I Get The Generator Installed)
(Goal Knowledge Person (Believe Person
(Actions-For-Obligation Generator Physical ((Action-Description Physical Person ;#:V4889 7434027 *2 ;#:V4890 7434945
;#:V4891 7434063 *2 ;#:V4892 7434101 *2
(Additions (Believe Person (Physical-Fact (Generator Installed)))) ;#:V4893 7434125 *2 ;#:V4894 7434143 *2))))
(Hm.)
System
(Inform Person (Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator)
(Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
(Body Serial-And ((Tighten-Generator-Bolts) (Attach-Power-Cable)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ;#:V4875 7466212 *2))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))) Nil))))
(Actions-For-Obligation Generator Physical
((Action-Description Physical Person (Install-Generator)
(Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
(Body Serial-And ((Tighten-Generator-Bolts) (Attach-Power-Cable)))
(Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ;#:V4900 7434462 *2))
(Additions (Believe Person (Physical-Fact (Generator Installed))))
(Deletions (Believe Person (Physical-Fact (Generator Not-Installed)))) Nil))))
(I Think You Need To Know That To Install Generator You Must Lift It Then Tighten Bolts Then Attach Power Cable)
(Believe Person (Physical-Fact (Generator Installed)))
(I Inferred That You Want To Achieve State With Generator Installed)
(If You Agree Say Yes)
% Yes

(System)
(Assume Generator-Type Light)
System
(Inform Person (Physical-Fact (Generator-Type Light)))
Person
(Person-Lift-Generator)
System
(Inform Person (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))))
System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Pump-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
Person
(Tighten-Generator-Bolts)
Person
(Check Power On)
System
(Inform Person (Actions-For-Obligation Power Physical
((Action-Description Physical Person (Turn-Off-Power) (Preconditions (Believe Person (Physical-Fact (At-Person Corner))))
(Body) (Restrictions (Believe Person (Physical-Fact (Power On)) ;#:V4881 7544536 *2))
(Additions (Believe Person (Physical-Fact (Power Off))))
(Deletions (Believe Person (Physical-Fact (Power On))))))))))
Person
(Move-Self Generator-Base Corner)
Person
(Turn-Off-Power)
Person
(Attach-Power-Cable)
Person
(Install-Generator)
Person
(Check Power Off)
Person
(Attach-Power-Cable)
Person
(Install-Generator)
System
(Inform Person (Actions-For-Obligation At-Generator Physical
((Action-Description Physical Person (Fork-Lift-Generator)
(Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))) (Believe Person (Physical-Fact (At-Fork-Lift Generator-Base))))
(Body)

```

```

(Restrictions (Believe Person (Physical-Fact (Generator-Type Heavy)) (#:V4885 7376674 #2))
(Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
(Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))))
(System)
(Assume Generator-Type Heavy)
System
(Inform Person (Physical-Fact (Generator-Type Heavy)))
Person
(Fork-Lift-Generator)
System
(Inform Person (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))))
System
(Inform Person (Physical-Fact (At-Socket-Wrench Corner)))
Person
(Move-Self Pump-Base Corner)
Person
(Get-Socket-Wrench Corner Generator-Base)
Person
(Tighten-Generator-Bolts)
Person
(Check Power On)
System
(Inform Person (Actions-For-Obligation Power Physical
((Action-Description Physical Person (Turn-Off-Power) (Preconditions (Believe Person (Physical-Fact (At-Person Corner))))
(Body)
(Restrictions (Believe Person (Physical-Fact (Power On)) (#:V4881 7644017 #2))
(Additions (Believe Person (Physical-Fact (Power Off))))
(Deletions (Believe Person (Physical-Fact (Power On))))))
Person
(Move-Self Generator-Base Corner)
Person
(Turn-Off-Power)
Person
(Attach-Power-Cable)
Person
(Install-Generator)
Person
(Check Power Off)
Person
(Attach-Power-Cable)
Person
(Install-Generator)
(Generator-Type Light)
(I Am Assuming That Generator-Type Light)
(Did You Successfully Person-Lift-Generator)
% No

(Actions-For-Obligation At-Generator Physical
((Action-Description Physical Person (Fork-Lift-Generator)
(Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))) (Believe Person (Physical-Fact (At-
Fork-Lift-Generator-Base))))
(Body)
(Restrictions (Believe Person (Physical-Fact (Generator-Type Heavy)) (#:V4902 7434316 #2))
(Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
(Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom))))))
(I Think You Need To Know That To Lift A Heavy Generator Use A Fork Lift)
(Generator-Type Heavy)
(I Am Assuming That Generator-Type Heavy)
(Did You Successfully Fork-Lift-Generator)
% Yes

(Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench))))
(I Think You Need To Know That To Tighten The Generator Bolts You Should Use A Socket Wrench)
(Physical-Fact (At-Socket-Wrench Corner))
(I Think You Need To Know That The Socket Wrench Is In The Corner)
(Is The Power In The State On)
% Yes

(Actions-For-Obligation Power Physical
((Action-Description Physical Person (Turn-Off-Power)
(Preconditions (Believe Person (Physical-Fact (At-Person Corner))))
(Body)
(Restrictions (Believe Person (Physical-Fact (Power On)) (#:V4901 7434500 #2))
(Additions (Believe Person (Physical-Fact (Power Off)))) (Deletions (Believe Person (Physical-Fact (Power On))))))
(I Think You Need To Know That To Turn The Power Off Use The Switch In The Corner)
(Well)
% (The Generator Shakes)
(Believe Person (Generator-Shakes))
(I Disagree And Thought You Did Too)
(Did You Do Tighten-Generator-Bolts)
% No

```

```
((Then Do (Transformed-Description 3426655 *2 And Redo ((Check Power On)
(Inform Person (Actions-For-Obligation Power Physical
((Action-Description Physical Person (Turn-Off-Power)
(Preconditions (Believe Person (Physical-Fact (At-Person Corner)))) (Body)
(Restrictions (Believe Person (Physical-Fact (Power On)) ;#:V4661 7644017 *2)))
(Additions (Believe Person (Physical-Fact (Power Off))))
(Deletions (Believe Person (Physical-Fact (Power On)))))) (Move-Self Generator-Base Corner) (Turn-Off-Power) (Attach-
Power-Cable) (Install-Generator)))
(Well)
%)
```

Back To Dribbling Lisp Listener.

(Dribble-End)

Program Listing

E1 Builtin

```
(Defun Builtin ())
```

```
;; Builtin Utilities
```

```
(Pr (Remove-Clauses (Succeed Prove-Every Pfirst Final Lreverse Append Prove Not
For-All Permutations Intersect Or If Member Every-Member Remove-Clauses Eq Apply)))
```

```
((Succeed))
```

```
;; Proves Clause Every Way Possible
```

```
((Prove-Every ?Clause) ?Clause (Fail))
((Prove-Every ?Clause))
```

```
((Pfirst ?F (?F . ?*))
```

```
((Final ?X) ?X))
((Final (?* . ?Rest) ?X) (Final ?Rest ?X))
```

```
((Lreverse () ()))
```

```
((Lreverse (?H . ?T) ?L)
 (Lreverse ?T ?R)
 (Append ?R (?H) ?L))
```

```
((Append () ?L ?L))
((Append (?H . ?T) ?L (?H . ?U)) (Append ?T ?L ?U))
```

```
((Prove ?X) . ?X)
```

```
((Not . ?X) ?X ! (Fail))
((Not . ?X))
```

```
((For-All ?X ?Y) (Not Prove ((Prove ?X) (Not Prove ?Y))))
```

```
((Permutations ?L (?H . ?T))
 (Append ?V (?H . ?U) ?L) (Append ?V ?U ?W) (Permutations ?W ?T))
((Permutations () ()))
```

```
((Intersect () ?* ()))
((Intersect ?X . ?R) ?Y (?X . ?Z)) (Member ?X ?Y) ! (Intersect ?R ?Y ?Z))
((Intersect ?X . ?R) ?Y ?Z) (Intersect ?R ?Y ?Z))
```

```
((Or ?X ?Y) . ?X)
((Or ?X ?Y) . ?Y)
```

```
((If ?X ?Y ?Z) ?X ! . ?Y)
((If ?X ?Y ?Z) . ?Z)
```

```
((Member ?H (?H . ?*)) !
((Member ?H (?* . ?T)) (Member ?H ?T))
```

```
((Every-Member ?H (?H . ?*)) ; A Member With No Cut
((Every-Member ?H (?* . ?T)) (Every-Member ?H ?T))
```



```
((Remove-Clauses (?First-Clause-Head ?Rest-Clause-Heads))  
 (Delete-All-Clauses ?First-Clause-Head)  
 (Remove-Clauses ?Rest-Clause-Heads))
```

```
((Eq ?X ?X))
```

```
((Apply ?X ()))  
((Apply ?X (?Y . ?Z)) (?X ?Y) (Apply ?X ?Z)))
```

E2 Facts

```
(Defun Facts ())
```

```
  ;; Fact Manipulation
```

```
  (Pr (Remove-Clauses
       (Inferred-Belief Fact-Belief Collect-Fact-Beliefs
        Collect-Fact-Beliefs-In-Clause-Body)))
```

```
  ;; Checks If Head Clause Is A Fact Belief As Opposed To An Inferred Belief
```

```
((Fact-Belief (Physical-Fact . ?*)))
```

```
((Fact-Belief (Actions-For-Obligation . ?*)))
```

```
((Fact-Belief (Requirement . ?*)))
```

```
((Fact-Belief (Reversible . ?*)))
```

```
((Fact-Belief (Description . ?*)))
```

```
  ;; Checks If Head Clause Is An Inferred Belief As Opposed To An Fact Belief
```

```
((Inferred-Belief ?Belief)
 (Not Fact-Belief ?Belief))
```

```
  ;; Gets Justifications For Belief Inference
```

```
((Get-Belief-Inference-Justification ?Anyone ?Belief ?Justification)
 (Believe ?Anyone ?Belief ?Justification) !)
```

```
  ;; Collects All Fact Beliefs In Clause With That Head
```

```
((Collect-Fact-Beliefs ?Anyone (?Head . ?Rest) ?Fact-Beliefs)
 (Clause ((Believe ?Anyone (?Head . ?Rest) ?*) . ?Clause-Body))
 (Collect-Fact-Beliefs-In-Clause-Body ?Clause-Body ?Fact-Beliefs))
```

```
  ;; Collects Fact Beliefs In Clause Body
```

```
((Collect-Fact-Beliefs-In-Clause-Body ((Believe ?Anyone ?Belief) . ?Rest-Beliefs)
 ((Believe ?Anyone ?Belief) . ?Rest-Fact-Beliefs))
 (Fact-Belief ?Belief) !
 (Collect-Fact-Beliefs-In-Clause-Body ?Rest-Beliefs ?Rest-Fact-Beliefs))
```

```
((Collect-Fact-Beliefs-In-Clause-Body () ()))
```

```
((Collect-Fact-Beliefs-In-Clause-Body (?* . ?Rest-Beliefs) ?Rest-Fact-Beliefs) !
 (Collect-Fact-Beliefs-In-Clause-Body ?Rest-Beliefs ?Rest-Fact-Beliefs))
)
```

E3 Knowrep

```
(Defun Knowrep ())
```

```
;; Knowledge Representation
```

```
(Pr (Remove-Clauses (Novice Expert Get-Current-Fact-State Get-Truth-Value Get-Conjunction-Truth-Values Get-True-Or-False)))
```

```
((Novice)) ; Person Is Novice By Default
```

```
;; Get State Of Current Fact Belief
```

```
((Get-Current-Fact-State ?Anyone (Physical-Fact (?Belief-Head . ?Belief-Rest))
  (Physical-Fact (?Belief-Head . ?Current-Belief-Rest)))
  (Get-Truth-Value ?Anyone (Physical-Fact (?Belief-Head . ?Current-Belief-Rest)) True))
```

```
((Get-Current-Fact-State
  ?Anyone
  (Actions-For-Obligation ?Obligation ?Type . ?Actions-For-Obligation) ; These Are Special
  (Actions-For-Obligation ?Obligation ?Type . ?Current-Actions-For-Obligation))
  (Get-Truth-Value ?Anyone (Actions-For-Obligation ?Obligation ?Type . ?Current-Actions-For-Obligation) True) !)
```

```
((Get-Current-Fact-State ?Anyone (Actions-For-Obligation ?Obligation ?Type . ?Actions-For-Obligation)
  (Actions-For-Obligation ?Obligation ?Type . ?Current-Actions-For-Obligation))
  (Get-Truth-Value ?Anyone (Actions-For-Obligation ?Obligation ?Type . ?Current-Actions-For-Obligation) True))
```

```
((Get-Current-Fact-State ?Anyone (Requirement ?Action-Name ?Unacceptable-Tools)
  (Requirement ?Action-Name ?Current-Unacceptable-Tools))
  (Get-Truth-Value ?Anyone (Requirement ?Action-Name ?Current-Unacceptable-Tools) True))
```

```
((Get-Current-Fact-State ?Anyone (Reversible ?Action-Name ?Reversible-Flag)
  (Reversible ?Action-Name ?Current-Reversible-Flag))
  (Get-Truth-Value ?Anyone (Reversible ?Action-Name ?Current-Reversible-Flag) True))
```

```
((Get-Current-Fact-State ?Anyone (Description ?Object ?Description)
  (Description ?Object ?Current-Description))
  (Get-Truth-Value ?Anyone (Description ?Object ?Current-Description) True))
```

```
;; Gets Truth Value Of Belief Which Is True, False Or Unknown
```

```
((Get-Truth-Value Person (Physical-Fact (?Belief-Head . ?Belief-Rest)) Known)
  (Believe Person (Physical-Fact (?Belief-Head ?Current-Belief-State)) ?Justification)
  (Var ?Current-Belief-State) !)
```

```
((Get-Truth-Value ?Anyone (Physical-Fact (?Belief-Head . ?Belief-Rest)) Unknown)
  (Not Believe ?Anyone (Physical-Fact (?Belief-Head . ?*)) ?Justification) !)
```

```
((Get-Truth-Value ?Anyone (Actions-For-Obligation ?Obligation ?Type ?Action-Descriptions) Unknown)
  (Not Believe ?Anyone (Actions-For-Obligation ?Obligation ?Type . ?*) ?Justification) !)
```

```
((Get-Truth-Value ?Anyone (Requirement ?Action-Name ?Unacceptable-Tools) Unknown)
  (Not Believe ?Anyone (Requirement ?Action-Name . ?*) ?Justification) !)
```

```
((Get-Truth-Value ?Anyone (Reversible ?Action-Name ?Reversible-Flag) Unknown)
  (Not Believe ?Anyone (Reversible ?Action-Name . ?*) ?Justification) !)
```

```
((Get-Truth-Value ?Anyone (Description ?Object ?Description) Unknown)
  (Not Believe ?Anyone (Description . ?*) ?Justification) !)
```

```
((Get-Truth-Value ?Anyone ?Belief True)
  (Fact-Belief ?Belief)
  (Believe ?Anyone ?Belief ?Justification) !)
```

```
((Get-Truth-Value ?Anyone (Physical-Fact ?Fact) False) !)
```

```

((Get-Truth-Value ?Anyone ?Belief Unknown)
 (Fact-Belief ?Belief) !)

:: Inferred Belief
:: True If All Parts True
:: False If Any Part False
:: Unknown If Any Part Unknown And No Part False

((Get-Truth-Value ?Anyone ?Belief ?Truth-Value)
 (Inferred-Belief ?Belief)
 (Collect-Fact-Beliefs ?Anyone ?Belief ?Fact-Beliefs)
 (Get-Conjunction-Truth-Values ?Anyone ?Fact-Beliefs ?Conjunction-Truth-Values)
 (If (Prove ((Member Unknown ?Conjunction-Truth-Values)
 (Not Member False ?Conjunction-Truth-Values)))
 ((Eq ?Truth-Value Unknown))
 ((Get-True-Or-False ?Anyone ?Belief ?Truth-Value))))

:: Gets List Of Truth Values For Each Clause In Conjunction

((Get-Conjunction-Truth-Values ?Anyone () ()))

((Get-Conjunction-Truth-Values ?Anyone ((Believe ?Anyone ?First-Belief) . ?Rest-Beliefs) (?First-Truth-Value . ?Rest-
Truth-Values))
 (Get-Truth-Value ?Anyone ?First-Belief ?First-Truth-Value) !
 (If (Eq ?First-Truth-Value False)
 ((Eq ?Rest-Truth-Values Nil))
 ((Get-Conjunction-Truth-Values ?Anyone ?Rest-Beliefs ?Rest-Truth-Values))))

:: Gets Truth Value Of Belief Which Has No Unknowns

((Get-True-Or-False ?Anyone ?Belief True)
 (Believe ?Anyone ?Belief ?*) !)

((Get-True-Or-False ?Anyone ?Belief False))

)

```

E4 Warn

```
(Defun Warn ())
```

```
;; Warn Person Of Important But Not Essential Facts
```

```
(Pr (Remove-Clauses (Infer-Possible-Obligations Convert-Between-Obligation-And-Object Find-Object-With-Proper-Characteristics
Check-If-Descriptions-Known Infer-Possible-Obligations Classify-Obligations
Fix-Free-Obligations Warn-Of-Object-Requirements
Warn-Of-Unreversible-Action Produce-Warnings Test-Warn)))
```

```
((Convert-Between-Obligation-And-Object At-Socket-Wrench Socket-Wrench))
```

```
;; Goes Through List Of Beliefs And Makes Sure Person Knows Object Descriptions (Uniquely Associated With Obligation).
;; If They Exist
```

```
((Check-If-Descriptions-Known () ?*) !)
```

```
((Check-If-Descriptions-Known ((Believe ?Anyone (Physical-Fact (?Obligation ?State))) . ?Rest-Beliefs) ?Parent-Goal)
(Convert-Between-Obligation-And-Object ?Obligation ?Object) !
(Plan-For-Goal Knowledge ?Anyone (Believe ?Anyone (Description ?Object ?Description))
((Action-Uses-Object ?Object) ?Parent-Goal))
(Check-If-Descriptions-Known ?Rest-Descriptions ?Parent-Goal))
```

```
((Check-If-Descriptions-Known ((Believe ?Anyone (Physical-Fact (?Obligation ?State))) . ?Rest-Beliefs) ?Parent-Goal) !
(Check-If-Descriptions-Known ?Rest-Descriptions ?Parent-Goal))
```

```
;; Uses Requirements To Filter Out Objects That Would Otherwise Have The Proper Characteristics.
;; These Characteristics Are In An Isa Hierarchy With Objects At Bottom, Characteristics On Top
```

```
((Infer-Possible-Obligations ?Actor ?Action-Name ?Characteristics ?Obligation-Type ?Obligation)
(Get-Truth-Value ?Actor (Requirement ?Action-Name ?All-Acceptable-Objects) True) !
(Find-Object-With-Proper-Characteristics ?Characteristics ?Object)
(Member ?Characteristics ?Acceptable-Object) ?All-Acceptable-Objects)
(Member ?Object ?Acceptable-Object)
(Get-Obligation-Name ?Object ?Obligation-Type ?Obligation))
```

```
((Infer-Possible-Obligations ?Actor ?Action-Name ?Characteristics ?Obligation-Type ?Obligation)
(Find-Object-With-Proper-Characteristics ?Characteristics ?Object)
(Get-Obligation-Name ?Object ?Obligation-Type ?Obligation))
```

```
;; Returns Object That Has Proper Characteristics
```

```
((Find-Object-With-Proper-Characteristics () ?Object))
```

```
((Find-Object-With-Proper-Characteristics (?First-Characteristic . ?Rest-Characteristics) ?Object)
(Is ?Object ?First-Characteristic)
(Find-Object-With-Proper-Characteristics ?Rest-Characteristics ?Object))
```

```
;; Some Obligations In A Belief Are Free Which Means They Have The Form (?Characteristics ?Obligation-Type).
;; Any Obligation Of Proper Type That Describes
;; An Object With These Characteristics Is Acceptable,
;; That Is There May Be An Acceptable Set Of Obligations.
;; Normal Obligations Are Fixed.
;; This Clause Classifies The Beliefs In A Belief List Into Those That Contain Free-Obligations And Fixed-Obligations
```

```
((Classify-Obligations ((Believe ?Anyone
(Physical-Fact ((?Characteristics ?Obligation-Type) ?Obligation-State))) . ?Rest-Beliefs)
((Believe ?Anyone
(Physical-Fact ((?Characteristics ?Obligation-Type) ?Obligation-State))) . ?Rest-Free-Obligations)
?Fixed-Obligations) !
(Classify-Obligations ?Rest-Beliefs ?Rest-Free-Obligations ?Fixed-Obligations))
```

```
((Classify-Obligations ((Believe ?Anyone
(Physical-Fact (?Obligation ?Obligation-State))) . ?Rest-Beliefs)
?Free-Obligations
((Believe ?Anyone
(Physical-Fact (?Obligation ?Obligation-State))) . ?Rest-Fixed-Obligations) !
(Classify-Obligations ?Rest-Beliefs ?Free-Obligations ?Rest-Fixed-Obligations))
```

```
((Classify-Obligations () () ()))
```

```
:: Uses The Isa Hierarchy And Requirements To Return
:: A Set Of Possible Obligations That Are Acceptable,
:: Hence The Name Fix-Free-Obligations
```

```
((Fix-Free-Obligations ?Actor ?Action-Name
  ((Believe ?Actor
    (Physical-Fact ((?Characteristics ?Obligation-Type) ?Obligation-State))) . ?Rest-Free-Obligations)
  ((Believe ?Actor
    (Physical-Fact (?Obligation ?Obligation-State)) . ?Rest-Fixed-Obligations)))
  (Infer-Possible-Obligations ?Actor ?Action-Name ?Characteristics ?Obligation-Type ?Obligation)
  (Fix-Free-Obligations ?Actor ?Rest-Free-Obligations ?Rest-Fixed-Obligations))
```

```
((Fix-Free-Obligations ?Actor ?Action-Name () ()))
```

```
:: If Object Requirement Exists Make Sure The Person Knows
```

```
((Warn-Of-Object-Requirements Physical ?Action-Name ?Parent-Goal)
  (Get-Truth-Value System (Requirement ?Action-Name . ?*) True) !
  (Plan-For-Goal Knowledge Person (Believe Person (Requirement ?Action-Name ?All-Acceptable-Objects))
    ((Person-Know-Requirements ?Action-Name) ?Parent-Goal)))
```

```
((Warn-Of-Object-Requirements ?Action-Type ?Action-Name ?Parent-Goal))
```

```
:: If Action Is Irreversible Make Sure The Person Knows
```

```
((Warn-Of-Unreversible-Action Physical ?Action-Name ?Parent-Goal)
  (Get-Truth-Value System (Reversible ?Action-Name No) True) !
  (Plan-For-Goal Knowledge Person (Believe Person (Reversible ?Action-Name No))
    ((Person-Know-Action-Not-Reversible ?Action-Name) ?Parent-Goal)))
```

```
((Warn-Of-Unreversible-Action ?Action-Type ?Action-Name ?Parent-Goal))
```

```
:: Produce All The Warnings Necessary
```

```
((Produce-Warnings ?Action-Type ?Action-Name ?Parent-Goal)
  (Warn-Of-Unreversible-Action ?Action-Type ?Action-Name ?Parent-Goal)
  (Warn-Of-Object-Requirements ?Action-Type ?Action-Name ?Parent-Goal))
```

```
)
```

E5 Plan

```

(Defun Plan ()

  ;; Plan For A Goal

  (Pr (Remove-Clauses (Check-Obligations Check-Restrictions Try-Execute-Action
    Try-Execute-Body Legal-Type-Truth-Value
    Plan-For-Goals Plan-For-Goal Achieve-Goal
    Show-Plan Show-All-Plans Get-Associated-Action
    Get-Beliefs-In-Restrictions Get-Action-Descriptions Know-By-System)))

  ;; Gets Beliefs Clauses In Restrictions
  ;; This Is Necessary Because A Restriction Can Include Arbitrary Code.

  ((Get-Beliefs-In-Restrictions () ()) !)

  ((Get-Beliefs-In-Restrictions ((Believe ?Anyone ?Belief ?Justification) . ?Rest-Restrictions)
    ((Believe ?Anyone ?Belief) . ?Rest-Belief-Restrictions)) !
    (Get-Beliefs-In-Restrictions ?Rest-Restrictions ?Rest-Belief-Restrictions))

  ((Get-Beliefs-In-Restrictions (?* . ?Rest-Restrictions) ?Rest-Belief-Restrictions) !
    (Get-Beliefs-In-Restrictions ?Rest-Restrictions ?Rest-Belief-Restrictions))

  ;; Check That Obligations Are Met

  ((Check-Obligations
    ((Believe ?* (Physical-Fact (?Physical-Fact-Head ?Physical-Fact-State))) . ?Rest-Additions))
    (Obligation ?Physical-Fact-Head . ?Possible-State)
    (Member ?Physical-Fact-State ?Possible-States) !
    (Check-Obligations ?Rest-Additions))

  ((Check-Obligations ((Believe ?* (?Fact-Head . ?*)) . ?Rest-Additions))
    (Not Eq ?Fact-Head Physical-Fact)
    (Check-Obligations ?Rest-Additions))

  ((Check-Obligations ()))

  ;; Check That Restrictions On Action Are Met

  ((Check-Restrictions ?Restrictions)
    ?Restrictions)

  ;; Tries To Execute The Action.
  ;; This Is A Lot Of Work Because System Must Make Sure Person Knows Appropriate Descriptions, Prohibitions, Re-
  versable Clauses
  ;; And Is Aware Of Any Restrictions On The Execution Of The Action

  ((Try-Execute-Action (Action-Description ?Type ?Actor (?Name . ?Arguments)
    (Preconditions . ?P) (Body . ?B) (Restrictions . ?R)
    (Additions . ?A) (Deletions . ?D)) ?Justification)
    (Pfirst ?Goal ?A) !
    (Check-If-Descriptions-Known ?P (Goal ?Type ?Actor ?Goal))
    (Produce-Warnings ?Type ?Name (Goal ?Type ?Actor ?Goal))
    (Classify-Obligations ?P ?Free-Obligations ?Fixed-Obligations)
    (Plan-For-Goals ?Type ?Actor ?Fixed-Obligations ((Want-Precondition (?Name . ?Arguments))
      (Goal ?Type ?Actor ?Goal)))
    (Fix-Free-Obligations ?Actor ?Name ?Free-Obligations ?New-Fixed-Obligations)
    (Plan-For-Goals ?Type ?Actor ?New-Fixed-Obligations ((Want-Precondition (?Name . ?Arguments))
      (Goal ?Type ?Actor ?Goal)))
    (Get-Beliefs-In-Restrictions ?R ?Restriction-Beliefs)
    (Plan-For-Goals Knowledge ?Actor ?Restriction-Beliefs ((Want-Know-Restrictions (?Name . ?Arguments))
      (Goal ?Type ?Actor ?Goal)))
    (Check-Restrictions ?R)
    (Try-Execute-Body ?Actor ?B (?Name . ?Arguments) ?Goal)
    (Check-Obligations ?A)
    (Execute-Action (Action-Description ?Actor (?Name . ?Arguments) (Preconditions . ?P) (Body . ?B)
      (Additions . ?A) (Deletions . ?D) (Restrictions . ?R)
      (Goal ?Type ?Actor ?Goal)))
    (Delete-Clause (Goal ?Type ?Actor ?Goal ?Justification))
    (Write ?Actor)
    (Write (?Name . ?Arguments)))

```

:: Tries To Execute The Body Of The Action

```
((Try-Execute-Body ?Actor () ?* ?*))
```

```
((Try-Execute-Body ?Actor (Serial-And {}) ?* ?*))
```

```
((Try-Execute-Body ?Actor (Serial-And (?First-Action . ?Rest-Actions)) ?Description ?Parent-Goal)
  (Get-Associated-Action ?Actor ?First-Action ?Action)
  (Eq (Action-Description ?* ?* ?* ?* ?* ?* (Additions ?Goal . ?*) . ?*) ?Action)
  (Add-End-Clause
    ((Goal Physical ?Actor ?Goal ((Wants-Action-In-Body ?Description) (Goal Physical ?Actor ?Parent-Goal))))))
  (Try-Execute-Action ?Action ((Wants-Action-In-Body ?Description) (Goal Physical ?Actor ?Parent-Goal)))
  (Try-Execute-Body ?Actor (Serial-And ?Rest-Actions) ?Description ?Parent-Goal))
```

```
((Try-Execute-Body ?Actor (Code ?Clauses) ?* ?*)
  ?Clauses)
```

:: Gets Action Associated With The Action Description, Sometimes Called The Action Name

```
((Get-Associated-Action ?Actor ?Description (Action-Description ?Type ?Actor ?Description . ?Rest-Of-Action))
  (Believe ?Actor (Actions-For-Obligation ?* ?Type ?Action-Descriptions) ?*)
  (Member (Action-Description ?Type ?Actor ?Description . ?Rest-Of-Action) ?Action-Descriptions) !)
```

:: This Is How A List Of Goals Is Achieved

```
((Plan-For-Goals ?Type ?Anyone ((Believe ?Anyone ?Belief) . ?Rest-Beliefs) ?Justification) ; A List Of Goals
  (Plan-For-Goal ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)
  (Plan-For-Goals ?Type ?Anyone ?Rest-Beliefs ?Justification))
```

```
((Plan-For-Goals ?Type ?Anyone () ?Justification)) ; A Null Goal
```

:: A Goal Of Executing Some Particular Action

```
((Plan-For-Goal Physical ?Anyone (Execute-Action ?Description) ?Justification)
  (Add-End-Clause ((Goal Physical ?Anyone (Execute-Action ?Description) ?Justification)))
  (Get-Associated-Action ?Anyone ?Description ?Action)
  (Try-Execute-Action ?Action ?Justification))
```

:: A General Goal Where ?Type Is Physical Or Knowledge

:: ?Anyone Is Person Or System

:: ?Belief Is Any Of The Legal Beliefs

:: Physical Actions Can Only Change Physical Facts, Thus

:: If ?Type Is Physical Then ?Belief Must Be (Physical-Fact ?Fact)

```
((Plan-For-Goal ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)
  (Add-End-Clause ((Goal ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)))
  (Get-Truth-Value ?Anyone ?Belief ?Truth-Value) !
  (Achieve-Goal ?Type ?Anyone ?Belief ?Truth-Value ?Justification))
```

:: Nothing Need Be Done To Achieve These Goals

```
((Achieve-Goal ?Type ?Anyone ?Belief True ?Justification)
  (Delete-Clause (Goal ?Type ?Anyone (Believe ?Anyone ?Belief) ?Justification)) )
```

```
((Achieve-Goal Knowledge ?Anyone ?Belief False ?Justification)
  (Delete-Clause (Goal Knowledge ?Anyone (Believe ?Anyone ?Belief) ?Justification)) )
```

:: If System Does Not Know And Person Does Not

:: Tell It The System Makes The Appropriate Assumption

```
((Achieve-Goal Knowledge System (Physical-Fact (?Fact-Head ?Fact-State)) Unknown ?Justification)
  (Not Believe System (Actions-For-Obligation ?Fact-Head Knowledge . ?*) ?*) !
  (Obligation ?Fact-Head . ?Possible-States)
  (Member ?Fact-State ?Possible-States)
  (Execute-Action (Action-Description System
    (Assume ?Fact-Head ?Fact-State))
```



```

        (Preconditions)
        (Body)
        (Additions (Believe System (Physical-Fact (?Fact-Head ?Fact-State))))
        (Deletions)
        (Restrictions)
        (Goal Knowledge System (Physical-Fact (?Fact-Head ?Fact-State))))
    (Write (System))
    (Write (Assume ?Fact-Head ?Fact-State))
    (Delete-Clause
      (Goal Knowledge System (Believe System (Physical-Fact (?Fact-Head ?Fact-State))) ?Justification))

    ;; ?Type Is Physical And ?Truth-Value Is False
    ;; Or ?Type Is Knowledge And ?Truth-Value Is Unknown
    ;; In Either Case The Person Must Perform Some Physical Action To Achieve The Goal

((Achieve-Goal ?Type Person (Physical-Fact (?Fact-Head . ?Fact-Rest)) ?Truth-Value ?Justification)
 (Legal-Type-Truth-Value-Combination ?Type ?Truth-Value)
 (Believe System (Actions-For-Obligation ?Fact-Head ?Type ?Action-Descriptions) Premise) ; All Ways System Knows
 How
 (Plan-For-Goal Knowledge Person (Believe Person (Actions-For-Obligation ?Fact-Head ?Type ?Action-Descriptions))
  ((Know-Action ?Fact-Head) ; Make Sure Person Knows How
   (Goal ?Type Person (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))))))
 (Every-Member (Action-Description ?Type Person ?Description (Preconditions . ?P) (Body . ?B) (Restrictions . ?R)
  (Additions (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))) . ?Rest-Additions)
  . ?Rest-Action) ?Action-Descriptions)
 (Try-Execute-Action (Action-Description ?Type Person ?Description (Preconditions . ?P) (Body . ?B) (Restrictions .
 ?R)
  (Additions (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))) . ?Rest-Additions)
  . ?Rest-Action) ?Justification))

((Legal-Type-Truth-Value-Combination Physical False))

((Legal-Type-Truth-Value-Combination Knowledge Unknown))

    ;; If System Does Not Know But The Person Does, It Simply Asks The Person
    ;; The System Does This To Simplify It's Planning Process

((Achieve-Goal Physical Person (Physical-Fact (?Fact-Head . ?Fact-Rest)) Known ?Justification)
 (Add-End-Clause ((Goal Knowledge System (Believe Person (Physical-Fact (?Fact-Head ?State)))
  ((Want-Simplify-Planning ?Fact-Head)
   (Goal Physical Person (Physical-Fact (?Fact-Head . ?Fact-Rest))))))
 (Try-Execute-Action (Action-Description Knowledge System
  (Ask-Person-State ?Fact-Head)
  (Preconditions)
  (Body Code ((Input (What Is The State Of The Physical Fact ?Fact-Head) ?State)))
  (Restrictions)
  (Additions (Believe Person (Physical-Fact (?Fact-Head ?State))))
  (Deletions (Believe Person (Physical-Fact (?Fact-Head ?*))))
  ((Want-Simplify-Planning ?Fact-Head)
   (Goal Physical Person (Physical-Fact (?Fact-Head . ?Fact-Rest))))))
 (Plan-For-Goal Physical Person (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))) ?Justification))

    ;; Person Has Physical Goal Whose Current State Is Unknown Which Means It May Already Be True

((Achieve-Goal Physical Person (Physical-Fact (?Fact-Head . ?Fact-Rest)) Unknown ?Justification)
 (Plan-For-Goal Knowledge Person (Believe Person (Physical-Fact (?Fact-Head . ?*)))
  ((Know-State ?Fact-Head)
   (Goal Physical Person (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))))))
 (Plan-For-Goal Physical Person (Believe Person (Physical-Fact (?Fact-Head . ?Fact-Rest))) ?Justification))

    ;; A General "Catch All" Where The System Tells The Person Something The Person Needs To Know

((Achieve-Goal Knowledge Person ?Belief Unknown ?Justification)
 (Fact-Belief ?Belief)
 (Not-Eq ?Belief (Actions-For-Obligation ?Type Knowledge . ?*))
 (Plan-For-Goal Knowledge System (Believe System ?Belief)
  ((Help-Person-Know-State ?Belief) (Goal Knowledge Person (Believe Person ?Belief))))
 (Get-Current-Fact-State System ?Belief ?New-Belief)
 (Execute-Action (Action-Description System
  (Inform Person ?New-Belief)
  (Preconditions)
  (Body)
  (Additions (Believe Person ?New-Belief))
  (Deletions)
  (Restrictions)
  (Goal Knowledge Person (Believe Person ?Belief))))

```

```

(Write System)
(Write (Inform Person ?New-Belief))
(Delete-Clause (Goal Knowledge Person (Believe Person ?Belief) ?*))

```

```

;; Person Wants To Know The Truth Of Some Inferred Beliefs.
;; Our Solution Is Actually Overkill In The Senses That System Tells Person State Of Every Unknown
;; But Is Reasonable

```

```

((Achieve-Goal Knowledge Person ?Belief Unknown ?Justification)
 (Inferred-Belief ?Belief)
 (Collect-Fact-Beliefs Person ?Belief ?Fact-Beliefs)
 (Plan-For-Goals Knowledge Person ?Fact-Beliefs : Make Sure Person Knows Truth Of Every Underlying Fact Belief
  ((Know-Underlying (Believe Person ?Belief)) (Goal Knowledge Person (Believe Person ?Belief))))
 (Delete-Clause (Goal Knowledge Person (Believe Person ?Belief) ?Justification)))

```

```

;; Shows All The Plans

```

```

((Show-Plan)
 (Future-Action ?Action)
 (Write ?Action)
 (Viewpoint Future ?Action)
 (Show-Plan))

```

```

((Show-Plan))

```

```

((Show-All-Plans)
 (Show-Plan)
 (Fail))

```

```

)

```

E6 Spact

```
(Defun Spact ()
```

```
  ;; Process The Speech Act From The Semantic Form
```

```
  (Pr (Remove-Clauses (Process-Speech-Act Process-Possible-Inform Further-Inform-Processing
    Process-Possible-Request Person-Inform Person-Request)))
```

```
  ;; Process The Speech Act Given The Semantic Form
```

```
((Process-Speech-Act (Believe Person ?Belief))
 (Get-Truth-Value Person ?Belief ?Truth-Value)
 (Process-Possible-Inform ?Belief ?Truth-Value))
```

```
((Process-Speech-Act (Goal ?Type Person (Believe Person ?Belief)))
 (Get-Truth-Value Person ?Belief ?Truth-Value)
 (Process-Possible-Request ?Type ?Belief ?Truth-Value))
```

```
  ;; This Is Not Really A Speech Act That We Can Handle
```

```
((Process-Speech-Act (Justify ?Action-Description))
 (World-Identifier ?Current-World)
 (Show-Justifications ?Action-Description)
 (Jump-World ?Current-World))
```

```
  ;; Process Possible Inform
```

```
  ;; Person Knows And System Does Not So Accept It As A Valid Inform
```

```
((Process-Possible-Inform ?Belief Known)
 (Write (Ok))
 (Person-Inform ?Belief))
```

```
  ;; Person Should Not Know This
```

```
((Process-Possible-Inform ?Belief Unknown)
 (Get-Truth-Value System ?Belief ?Truth-Value)
 (Further-Inform-Processing ?Belief ?Truth-Value))
```

```
  ;; But System Agrees So Be Surprised
```

```
((Further-Inform-Processing ?Belief True)
 (Write (I Agree And Did Not Think You Knew That))
 (Person-Inform ?Belief))
```

```
  ;; But System Disagrees So Complain
```

```
((Further-Inform-Processing ?Belief False)
 (Write (I Disagree))
 (Adjust-System-Beliefs ?Belief)
 (Fail))
```

```
  ;; But System Did Not Know Either So Be Surprised But Accept It As A Valid Inform
```

```
((Further-Inform-Processing ?Belief Unknown)
 (Write (I Did Not Know That And Did Not Think That You Did))
 (Person-Inform ?Belief))
```

```
  ;; Person Knows This Is True So Just Ignore It, It May Be An Indirect Speech Act
```

```
((Process-Possible-Inform ?Belief True))
```

```
  ;; Person Knows This Is False So Complain
```

```

((Process-Possible-Inform ?Belief False)
 (Write (I Disagree And Thought You Did Too)) !
 (Adjust-System-Beliefs ?Belief)
 (Fail))

:: Process Possible Request

:: Person Has Physical Goal That Is Already True So Complain

((Process-Possible-Request Physical ?Belief True)
 (Write (I Think This Is Already True)) !
 (Adjust-System-Beliefs ?Belief)
 (Fail))

:: Person Has Knowledge Goal That Is Already Known So Complain

((Process-Possible-Request Knowledge ?Belief True)
 (Write (I Think You Already Know That This Is True)) !
 (Adjust-System-Beliefs ?Belief)
 (Fail))

:: Person Has Knowledge Goal That Is Already Known So Complain

((Process-Possible-Request Knowledge ?Belief False)
 (Write (I Think You Already Know That Is Not True)) !
 (Adjust-System-Beliefs ?Belief)
 (Fail))

:: Person Has Knowledge Goal That Is Already Known So Complain

((Process-Possible-Request Knowledge ?Belief Known)
 (Write (I Do Not Know But I Think That You Know)) !
 (Adjust-System-Beliefs ?Belief)
 (Fail))

:: The Rest (Physical,False Or Unknown And Knowledge,Unknown Are Accepted As Valid Requests)

((Process-Possible-Request ?Type ?Belief ?Truth-Value)
 (Write (Hm.))
 (Person-Request ?Type ?Belief))

:: Person Inform'S System Speech Act

((Person-Inform (Physical-Fact (?Fact-Head ?Fact-State)))
 (Execute-Action (Action-Description
  Person
  (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
  (Preconditions)
  (Body)
  (Additions (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
  (Deletions (Believe Person (Physical-Fact (?Fact-Head ?*))))
  (Restrictions)
  (Sensors))))))

:: Person Request'S System Speech Act

((Person-Request ?Type ?Belief)
 (Execute-Action (Action-Description
  Person
  (Inform System (Goal ?Type Person (Believe Person ?Belief)))
  (Preconditions)
  (Body)
  (Additions (Believe Person (Goal ?Type Person (Believe Person ?Belief))))
  (Deletions)
  (Restrictions)
  (Sensors))))))

```

E7 Infer

```
(Defun Infer ()
```

```
  ;; Infer Indirect Speech Acts
```

```
  (Pr (Remove-Clauses (Prove-And-Count Prove-And-Count Delete-Clause-In-Case Save-Result Increment-Count
    Find-Action-With-Goal-As-Precondition Check-If-Valid-Precondition Infer-Request
    Find-Action-With-Description-In-Body Additions-True Find-Inferred-Physical-Goal
    Infer-Higher-Goals-Given-Goal Find-Higher-Goals Infer-Higher-Goals-Given-Fact
    Infer-Higher-Actions-Given-Description Verify-Inference)))
```

```
  ;; Prove Clause And Count Number Of Times Clause Can Be Proved
```

```
  ((Prove-And-Count ?Result ?Clause)
   (Delete-Clause-In-Case (Count ?*)
    (Delete-Clause-In-Case (Result ?*)
     (Add-End-Clause ((Count 0))
      (Add-End-Clause ((Result Fail)))
      'Clause
      (Save-Result ?Result)
      (Increment-Count)
      (Fail)))
```

```
  ((Prove-And-Count ?Result ?*)
   (Result ?Result))
```

```
  ;; Deletes Clause If Necessary
```

```
  ((Delete-Clause-In-Case ?Clause)
   (Delete-Clause ?Clause) !)
```

```
  ((Delete-Clause-In-Case ?*) !)
```

```
  ;; Save The Result Of Proof
```

```
  ((Save-Result ?R)
   (Delete-Clause (Result ?*)
    (Add-End-Clause ((Result ?R))) !)
```

```
  ;; Increment Count Of Number Of Proofs
```

```
  ((Increment-Count)
   (Count ?Old-Count)
   (Add 1 ?Old-Count ?New-Count)
   (Delete-Clause (Count ?*))
   (Add-End-Clause ((Count ?New-Count))) !)
```

```
  ;; Find An Action With This Goal As A Precondition
```

```
  ((Find-Action-With-Goal-As-Precondition ?Goal (Action-Description Physical Person (?Name . ?Arguments)
    (Preconditions . ?P) (Body . ?B)
    (Restrictions . ?R) (Additions . ?A)
    (Deletions . ?D)))
   (Believe Person (Actions-For-Obligation ?* Physical ?Action-Descriptions) ?Justification)
   (Member (Action-Description Physical Person (?Name . ?Arguments)
    (Preconditions . ?P) (Body . ?B)
    (Restrictions . ?R) (Additions . ?A)
    (Deletions . ?D)) ?Action-Descriptions)
   (Check-If-Valid-Precondition ?Name ?A ?P ?Goal))
```

```
  ;; Special Care Must Be Taken Because Of Potential Free Obligations
```

```
  ((Check-If-Valid-Precondition ?Action-Name ?Additions ((Believe ?Anyone ?Belief) . ?*) (Believe ?Anyone ?Belief))
   (Not Member (Believe ?Anyone ?Belief) ?Additions))
```

```
  ((Check-If-Valid-Precondition ?Action-Name ?Additions
   ((Believe ?Anyone (Physical-Fact ((?Characteristics ?Obligation-Type) ?Obligation-State))) . ?*)
   (Believe ?Anyone (Physical-Fact (?Fixed-Obligation ?Obligation-State))))
```

```

((Find-Higher-Goals ?Inferred-Goal (Execute-Action ?Highest-Action-Description))
 (Find-Action-With-Goal-As-Precedent ?Inferred-Goal
  (Action-Description ?* Person ?First-Action-Description ?* ?* ?*
   (Additions : ?A) ?* ?* ?*)))
(Not-Additions-True ?A) ; If Affects Already True Can'T Infer This Action
(Infer-Higher-Actions-Given-Description ?First-Action-Description ?Rest-Actions-Description)
(Not-Eq ?Rest-Actions-Description ())
(Final (?(First-Action-Description : ?Rest-Actions-Description) ?Highest-Action-Description) ?)

```

```
((Find-Higher-Goals ?Inferred-Goal ?Inferred-Goal))
```

```
;; Infer Request Given Semantic Form Of Input
```

```
((Infer-Request (Goal ?Type Person (Believe Person ?Belief)) ?Inferred-Goal)
 (Prove-And-Count
  (?Inferred-Goal)
  (Infer-Higher-Goals-Given-Goal ?Type (Believe Person ?Belief) ?Inferred-Goal))
 (Verify-Inference ?Inferred-Goal))
```

```
((Infer-Request (Believe Person ?Belief) ?Inferred-Goal)
 (Prove-And-Count (?Inferred-Goal) (Infer-Higher-Goals-Given-Fact ?Belief ?Inferred-Goal)))
```

```
;; Tries To Infer Higher Goals Given Fact
```

```
((Infer-Higher-Goals-Given-Fact (Physical-Fact (?Fact-Head ?Fact-State))
 (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
 (Obligation ?Fact-Head ?Fact-State ?Inferred-Fact-State)
 (Undesired-State-Of-Obligation ?Fact-Head ?Fact-State) !)
```

```
((Infer-Higher-Goals-Given-Fact (Physical-Fact (?Fact-Head ?Fact-State))
 (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
 (Obligation ?Fact-Head ?Inferred-Fact-State ?Fact-State)
 (Undesired-State-Of-Obligation ?Fact-Head ?Fact-State) !)
```

```
((Infer-Higher-Goals-Given-Fact (Physical-Fact (?Fact-Head ?Fact-State))
 (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
 (Obligation ?Fact-Head ?Inferred-Fact-State ?Fact-State)
 (Not-Past-Action (Action-Description Person (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
 (?*)) !)
```

```
((Infer-Higher-Goals-Given-Fact (Physical-Fact (?Fact-Head ?Fact-State))
 (Believe Person (Physical-Fact (?Fact-Head ?Inferred-Fact-State))))
 (Obligation ?Fact-Head ?Fact-State ?Inferred-Fact-State)
 (Not-Past-Action (Action-Description Person (Inform System (Believe Person (Physical-Fact (?Fact-Head ?Fact-State))))
 (?*)) !)
```

```
;; Tries To Chain Through Body To Top Action That Can Be Inferred
```

```
((Infer-Higher-Actions-Given-Description ?Description ()))
```

```
((Infer-Higher-Actions-Given-Description
 ?Description (?First-Action-Description . ?Rest-Actions-Description))
 (Find-Action-With-Description-In-Body ?Description ?Action) !
 (Not Action-Affects-True ?Action) ; If Affects Already True Can'T Infer This Action
 (Eq (Action-Description ?Type ?Actor ?First-Action-Description . ?*) ?Action)
 (Infer-Higher-Actions-Given-Description ?First-Action-Description ?Rest-Actions-Description))
```

```
;; Verify Inferred Goal
```

```
((Verify-Inference ?Inferred-Goal)
 (Count 1)
 (Semantic-Form-To-English ?Transformed-Goal ?Inferred-Goal)
 (Write (I Inferred That You Want ?Transformed-Goal))
 (Input (If You Agree Say Yes) ?Response) !
 (Eq ?Response Yes))
```

```
)
```

E8 Execute

```
(Defun Execute ())
```

```
;; Execute Predictions Created By Planning
```

```
(Pr (Remove-Clauses (Top Bottom Forward Backward Find-Last-System-Action Execute Predictions
  Execute-Plan Wait-For-Synchronization Perform-All-System-Actions
  Perform-System-Action Silent-Execute-Plan Check-That-Valid-Statement Describe-Assumed-Actions
  Check-That-Assumption-Made Process-Region-Input Respond-To-Input Verify-Person-Action
  Answer-Question Next-Active-Region End-Active-Region Action-Based-On-Assumption)))
```

```
((Execute-Predictions)
 (Novice)
 (Execute-Plan))
```

```
((Execute-Predictions)
 (Expert)
 (Silent-Execute-Plan))
```

```
;; Some Helpful Utilities
```

```
((Top)
 (Backward)
 (Top))
```

```
((Top))
```

```
((Bottom)
 (Forward)
 (Bottom))
```

```
((Bottom))
```

```
((Forward)
 (Future-Action ?Action)
 (Viewpoint Future ?Action))
```

```
((Backward)
 (Past-Action ?Action)
 (Viewpoint Past ?Action))
```

```
((Find-Last-System-Action ?Description)
 (Past-Action (Action-Description System ?Description . ?*)))
```

```
((Find-Last-System-Action ?Description)
 (Past-Action (Action-Description Person . ?Rest-Action))
 (Viewpoint Past (Action-Description Person . ?Rest-Action)) !
 (Find-Last-System-Action ?Description))
```

```
;; This Is The Novice Mode Where The System Guides The Person And Does Not Wait To Be Asked.
;; The System Also Verifies Certain Actions Of The Person
```

```
((Execute-Plan ?Previous-Assumptions)
 (Future-Action (Action-Description Person . ?Rest-Action))
 (Viewpoint Future (Action-Description Person . ?Rest-Action))
 (Verify-Person-Action (Action-Description Person . ?Rest-Action) ?Previous-Assumptions)
 (Execute-Plan ?Previous-Assumptions))
```

```
((Execute-Plan ?*)
 (Future-Action (Action-Description System . ?*))
 (Future-Action (Action-Description System . ?Rest-Action))
 (Perform-All-System-Actions (Action-Description System . ?Rest-Action) ?New-Assumptions)
 (Execute-Plan ?New-Assumptions))
```

```
((Execute-Plan ?*)
 (Not Future-Action . ?*))
```



```

;; Perform A System Action

;; For An Assumption Simply State The Assumption

((Perform-System-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Semantic-Form-To-English ?Transformed-Assume ?Rest-Assume)
 (Write (I Am Assuming That . ?Transformed-Assume))))

;; For An Inform That Was Preceded By An Assumption, Do Nothing

((Perform-System-Action (Action-Description System (Inform Person (Physical-Fact ?Rest-Inform)) . ?Rest-Action))
 (Past-Action (Action-Description System (Assume . ?Rest-Inform) . ?*) !))

;; For A Normal Inform, Simply Execute It

((Perform-System-Action (Action-Description System (Inform Person ?Rest-Inform) . ?Rest-Action))
 (Semantic-Form-To-English ?Transformed-Inform ?Rest-Inform)
 (Write (I Think You Need To Know That . ?Transformed-Inform)))

;; Perform A Series Of Assumptions Or Informs

((Perform-All-System-Actions (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action)
 ((Assume . ?Rest-Assume) . ?Rest-Assumptions))
 (Perform-System-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Viewpoint Future (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Future-Action (Action-Description System . ?Rest-Future-Action))
 (Perform-All-System-Actions (Action-Description System . ?Rest-Future-Action) ?Rest-Assumptions))

((Perform-All-System-Actions (Action-Description System (Inform Person ?Rest-Inform) . ?Rest-Action) ?Assumptions)
 (Perform-System-Action (Action-Description System (Inform Person ?Rest-Inform) . ?Rest-Action))
 (Viewpoint Future (Action-Description System (Inform Person ?Rest-Inform) . ?Rest-Action))
 (If (Future-Action (Action-Description System . ?Rest-New-Action))
 (Perform-All-System-Actions (Action-Description System . ?Rest-New-Action) ?Assumptions))
 (Succeed) (Eq ?Assumptions Nil)))

;; If Need Be, Verify The Person'S Actions

((Verify-Person-Action (Action-Description Person ?Name . ?Rest-Action) ?Assumptions)
 ; Check If Based On Assumption
 (World-Identifier ?Current-World)
 (Action-Based-On-Assumption ?Answer)
 (Jump-World ?Current-World)
 (Eq ?Answer Yes) !
 (Input (Did You Successfully . ?Name) ?Response)
 (Eq ?Response Yes))

((Verify-Person-Action (Action-Description Person ?Name . ?Rest-Action) ?Assumptions) ; Check If Synchronization
 Point
 (Future-Action (Action-Description System . ?*)) ; This Is Last Person Action Before First System Action
 (Not Eq ?Name (Inform . ?*))
 (Not Eq ?Name (Check . ?*))
 (Input (Tell Me When You Have Done Action ?Name) ?Response) !)

((Verify-Person-Action (Action-Description Person (Check ?Obligation ?State) . ?Rest-Action) ?Assumptions) ; Verify
 Result If
 (Input (Is The ?Obligation In The State ?State) ?Response) ! ; Action Is Knowledge Action
 (Eq ?Response Yes))

((Verify-Person-Action (Action-Description Person ?Name . ?Rest-Action) ?Assumptions))

;; Action Is Based On Assumption If Assumption Is Two Steps Back From The Action.
;; This Is A Good Approximation Since Assume Is Always Done For An Action.
;; Correct Method Is Actually To Look At The Action Preconditions And Restrictions.

((Action-Based-On-Assumption Yes)
 (Backward)
 (Backward)
 (Past-Action (Action-Description System (Assume . ?Arguments) . ?*) !))

```

```
((Action-Based-On-Assumption No))
```

```
:: This Is Expert Mode Where The System Simply Waits Until It Is Asked The Question.
:: This Is Rather Strange Since It Knows What Will Be Asked
```

```
((Silent-Execute-Plan)
(Future-Action (Action-Description System . ?Rest-Action))
(Perform-All-System-Actions (Action-Description System . ?Rest-Action))
(World-Identifier ?Current-Region)
(Process-Region-Input ?Current-Region ()))
```

```
((Silent-Execute-Plan)
(Future-Action (Action-Description Person . ?*))
(World-Identifier ?Current-Region)
(Process-Region-Input ?Current-Region ()))
```

```
((Silent-Execute-Plan))
```

```
:: Checks That Person Does Not Already Know The Answer
```

```
((Check-That-Valid-Statement (Goal Knowledge Person (Believe Person ?Belief)))
(Get-Truth-Value Person ?Belief Known)
(Write (I Thought You Knew That)) !
(Fail))
```

```
((Check-That-Valid-Statement (Goal Knowledge Person (Believe Person ?Belief)))
(Get-Truth-Value Person ?Belief True)
(Semantic-Form-To-English ?Transformed-Belief ?Belief)
(Write (I Thought You Already Knew . ?Transformed-Belief)) !
(Fail))
```

```
((Check-That-Valid-Statement (Believe Person ?Belief))
(Get-Truth-Value System ?Belief False)
(Adjust-System-Beliefs ?Belief) !
(Fail))
```

```
((Check-That-Valid-Statement ?*))
```

```
:: Process All Questions For A Region.
:: A Region Is A Contiguous Area Of The Tree With The Same Actor
```

```
((Process-Region-Input ?Current-Region ?Currently-Answered)
(Input (Questions) ?Question)
(English-To-Semantic-Form ?Question ?Semantic-Form)
(If (Check-That-Valid-Statement ?Semantic-Form)
(Respond-To-Input ?Semantic-Form ?Current-Region ?New-Region ?Currently-Answered ?New-Currently-Answered)
(Process-Region-Input ?New-Region ?New-Currently-Answered)
(Process-Region-Input ?Current-Region ?Currently-Answered))))
```

```
:: Try To Respond To The Input
```

```
((Respond-To-Input (Justify ?Action-Description) ?Current-Region ?Current-Region ?Currently-Answered ?Currently-Answered)
(Show-Justifications ?Action-Description))
```

```
((Respond-To-Input (Goal Knowledge Person (Believe Person ?Belief)) ; Answer Question If Possible
?Current-Region ?Current-Region ?Currently-Answered (?Belief . ?Currently-Answered))
(Answer-Question ?Belief)
(Jump-World ?Current-Region))
```

```
((Respond-To-Input (Goal Knowledge Person (Believe Person ?Belief))
?Current-Region ?New-Region ?Currently-Answered ?New-Currently-Answered)
; Question Can Not Be Answered
; So Go On To Next Region
(End-Active-Region ?Currently-Answered)
(Next-Active-Region ?New-Region ?Persons-Actions)
(Describe-Assumed-Actions ?Persons-Actions)
(Respond-To-Input (Goal Knowledge Person (Believe Person ?Belief))
?New-Region ?New-Region () ?New-Currently-Answered))
```

```
:: Describe The Actions That System Must Assume The Person Has Done
```

:: Because The Question Occurs In The Next Region

((Describe-Assumed-Actions ()))

((Describe-Assumed-Actions (?First-Description . ?Rest-Descriptions))
 (Write (I Assumed That You Have Done Action . ?First-Description))
 (Describe-Assumed-Actions ?Rest-Descriptions))

:: Answer The Question

((Answer-Question ?Belief)
 (Future-Action (Action-Description System (Inform Person ?Belief) . ?Rest-Action))
 (Not Past-Action (Action-Description System (Assume . ?Belief) . ?*))
 (Perform-System-Action (Action-Description System (Inform Person ?Belief) . ?Rest-Action)))

((Answer-Question ?Belief)
 (Future-Action (Action-Description System . ?Rest-Action))
 (Viewpoint Future (Action-Description System . ?Rest-Action))
 (Answer-Question ?Belief))

:: Find The Next Region

((Next-Active-Region ?World-Identifier (?Description . ?Rest-Person-Actions))
 (Future-Action (Action-Description Person ?Description . ?Rest-Action))
 (Viewpoint Future (Action-Description Person ?Description . ?Rest-Action))
 (Next-Active-Region ?World-Identifier ?Rest-Person-Actions))

((Next-Active-Region ?World-Identifier ())
 (Future-Action (Action-Description System . ?Rest-Action))
 (World-Identifier ?World-Identifier))

:: Go To The End Of The Current Region

((End-Active-Region ?Currently-Answered)
 (Future-Action (Action-Description System (Inform . ?Rest-Inform) . ?Rest-Action))
 (Viewpoint Future (Action-Description System (Inform . ?Rest-Inform) . ?Rest-Action))
 (End-Active-Region ?Currently-Answered))

((End-Active-Region ?Currently-Answered)
 (Future-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Check-That-Assumption-Made ?Currently-Answered ?Assumption-Description)
 (Viewpoint Future (Action-Description System ?Assumption-Description . ?*))
 (End-Active-Region ?Currently-Answered))

((End-Active-Region ?*)
 (Future-Action (Action-Description Person . ?Rest-Action)))

:: Check The Assumptions

((Check-That-Assumption-Made ?Currently-Answered (Assume . ?Rest-Assume))
 (Future-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Member (Physical-Fact ?Rest-Assume) ?Currently-Answered))

((Check-That-Assumption-Made ?Currently-Answered (Assume . ?Rest-Assume))
 (Future-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action))
 (Perform-System-Action (Action-Description System (Assume . ?Rest-Assume) . ?Rest-Action)))

)

E9 Debug

```

(Defun Debug ()

  ;; Tries To Find The Reason For Discrepancies In Beliefs Between The System And The Person

  (Pr (Remove-Clauses (Adjust-System-Beliefs Check-For-Missing-Actions Back-Up-To-World
    Check-If-Negating-Action-Forgotten)))

  ;; Back Up To The First World Where The Belief Is True.
  ;; The Future Action In This World Is The One That Negated This Belief

  ((Back-Up-To-World ?Belief ())
   (Believe Person ?Belief ?Justification) !)

  ((Back-Up-To-World ?Belief (?Description . ?Rest-Descriptions))
   (Fast-Action (Action-Description ?Actor ?Description . ?Rest-Action))
   (Viewpoint Past (Action-Description ?Actor ?Description . ?Rest-Action))
   (Back-Up-To-World ?Belief ?Rest-Descriptions))

  ;; Ask Person If Negating Action Is Forgotten, If So Person Must Redo A Number Of Actions

  ((Check-If-Negating-Action-Forgotten ?Belief)
   (Back-Up-To-World ?Belief ?Descriptions)
   (Reverse ?Descriptions (?First-Description . ?Rest-Descriptions))
   (Input (Did You Do . ?First-Description) ?Response)
   (Eq ?Response No) !
   (Write (Then Do ?First-Description And Redo ?Rest-Descriptions)))

  ;; Belief Must Be An Inferred Belief.
  ;; System Believes That Some Underlying Fact Is False But Person Does Not.
  ;; System Checks Each Of The Fact Beliefs To Make Sure That Each Action Was Properly
  ;; Executed By The Person

  ((Adjust-System-Beliefs ?Belief)
   (Collect-Fact-Beliefs Person ?Belief ?Fact-Beliefs)
   (Check-For-Missing-Actions ?Fact-Beliefs))

  ((Check-For-Missing-Actions ((Believe ?Anyone (Physical-Fact ?Fact)) . ?Rest-Fact-Beliefs)
   (Check-If-Negating-Action-Forgotten (Physical-Fact ?Fact))
   (Check-For-Missing-Actions ?Rest-Fact-Beliefs))

  ((Check-For-Missing-Actions ()))

)

```

E10 Justify

```
(Defun Justify ()
```

```
  ;; Show Justifications For Action That The System Has Performed.
  ;; A Fact Is Justified By An Action Or Is A Premise
  ;; An Action Is Justified By A Goal
  ;; A Goal Is Justified By The Pair (Goal Rule, (Another Goal Or Action))
```

```
  (Pr (Remove-Clauses (Find-Goal-Justification-For-Action
                       Print-Goal-Justification-Chain Show-Justifications
                       Semantic-Form-To-English-If-Possible)))
```

```
  ;; Finds The Goal Which Justified The Action.
  ;; Must Search The Past Till The Goal Is Found Since Action Deleted It In The Present
```

```
((Find-Goal-Justification-For-Action ?Description ?Goal-Justification)
 (Past-Action (Action-Description ?Actor ?Description ?Preconditions ?Body
                                   ?Additions ?Deletions ?Restrictions ?Goal-Justification)) !
 (Viewpoint Past (Action-Description ?Actor ?Description ?Preconditions ?Body
                                   ?Additions ?Deletions ?Restrictions ?Goal-Justification)))
```

```
((Find-Goal-Justification-For-Action ?Description ?Goal-Justification)
 (Past-Action ?Action-Description)
 (Viewpoint Past ?Action-Description) !
 (Find-Goal-Justification-For-Action ?Description ?Goal-Justification))
```

```
  ;; Convert Semantic Form To English If Possible
```

```
((Semantic-Form-To-English-If-Possible ?Transformed-Goal ?Goal)
 (Semantic-Form-To-English ?Transformed-Goal ?Goal))
```

```
((Semantic-Form-To-English-If-Possible ?Goal ?Goal))
```

```
  ;; Print Chain Of Justifications For Goals.
  ;; The Chain Ends With An Action Which Is Always The Person Informing The System
```

```
((Print-Goal-Justification-Chain (Goal ?Type ?Actor (Believe ?Actor ?Belief)))
 (Goal ?Type ?Actor (Believe ?Actor ?Belief) (?Rule ?Parent-Goal))
 (Write (Goal ?Type ?Actor ?Belief Because ?Rule))
 (Input (Further Justifications) ?Response) !
 (Eq ?Response Yes)
 (Print-Goal-Justification-Chain ?Parent-Goal))
```

```
((Print-Goal-Justification-Chain (Inform . ?*))
 (Write (Because You Told Me)))
```

```
((Print-Goal-Justification-Chain (Believe Person (Physical-Fact ?Fact)))
 (Semantic-Form-To-English ?Transformed-Fact ?Fact)
 (Write (Because You Told Me . ?Transformed-Fact)))
 (Believe Person (Physical-Fact ?Fact) ?Justification)
 (Write (Believe Person (Physical-Fact ?Fact) Because ?Justification)))
```

```
  ;; Shows Justifications For An Action
```

```
((Show-Justifications ?Action-Description)
 (Find-Goal-Justification-For-Action ?Action-Description ?Goal-Justification)
 (Print-Goal-Justification-Chain ?Goal-Justification)
 )
```

E11 Main

```

(Defun Main ()

  ;; The Main Loop

  (Pr (Remove-Clauses (Novice-Mode Expert-Mode Get-English-Input Main Process-Input Process-Possible-Direct-Request
    Process-Possible-Indirect-Request)))

  ;; Set Mode

  ((Expert-Mode)
   (Delete-Clause (Novice))
   (Add-End-Clause ((Expert))))

  ((Novice-Mode)
   (Delete-Clause (Expert))
   (Add-End-Clause ((Novice))))

  ;; Get Input

  ((Get-English-Input ?Input)
   (Input (Well) ?Input))

  ((Get-English-Input ?Input)
   (Get-English-Input ?Input))

  ;; The Main Loop

  ((Main)
   (For-All ((Get-English-Input ?English-Input)
    ((Process-Input ?English-Input))))

  ;; Process Input

  ((Process-Input ?English-Input)
   (English-To-Semantic-Form ?English-Input ?Semantic-Form)
   (Process-Speech-Act ?Semantic-Form)
   (Process-Possible-Direct-Request ?Semantic-Form)
   (Process-Possible-Indirect-Request ?Semantic-Form))

  ((Process-Input ?*))

  ;; Process Possible Direct Request

  ((Process-Possible-Direct-Request (Goal ?Type Person (Believe Person ?Belief))
   (Prove-Every (Plan-For-Goal ?Type Person (Believe Person ?Belief)
    (Person-Said (Inform System (Goal ?Type Person (Believe Person ?Belief))))))
   (Execute-Predictions))

  ((Process-Possible-Direct-Request ?*))

  ;; Process Possible Indirect Request

  ((Process-Possible-Indirect-Request ?Semantic-Form)
   (Infer-Request ?Semantic-Form ?Inferred-Goal)
   (Prove-Every (Plan-For-Goal Physical Person ?Inferred-Goal (Inferred-Goal ?Semantic-Form)))
   (Execute-Predictions))

  ((Process-Possible-Indirect-Request ?*))

  )

(Defun
  Init ()

```

(Clear-All Clauses)
(Builtin)
(Facts)
(Beliefs)
(Knowrep)
(Hierarchy)
(Warn)
(Plan)
(Infer)
(Execute)
(Transform)
(Spact)
(Debug)
(Justify)
(Main))

(Defun
  Maint ()
  (Pr (Main)))

E12 Hierarchy

```
(Defun Hierarchy ()  
  ;; Isa Hierarchy  
  
  ((Pr (Remove-Clauses (Isa Is Get-Obligation-Name))))  
  
  ((Isa Adjustable-Wrench Wrench))  
  
  ((Isa Socket-Wrench Wrench))  
  
  ((Isa Pocket-Wrench Wrench))  
  
  ((Is ?X ?Y)  
   (Isa ?X ?Y))  
  
  ((Is ?X ?Y)  
   (Isa ?X ?Z)  
   (Is ?Z ?Y))  
  
  ((Get-Obligation-Name Adjustable-Wrench Location At-Adjustable-Wrench))  
  
  ((Get-Obligation-Name Socket-Wrench Location At-Socket-Wrench))  
  
  ((Get-Obligation-Name Pocket-Wrench Location At-Pocket-Wrench))  
  
  ((Get-Obligation-Name Generator State Generator))  
  
  )  
)
```


E13 Transform

```
(Defun Transform ()
```

```
  ;; Transform Between English And Semantic Form
```

```
  (Pr (Remove-Clauses (Transform-English-To-Semantic-Form Obligation-And-State Transform-Semantic-Form-To-English
    Action-Description-To-English English-To-Semantic-Form
    Semantic-Form-To-English)))
```

```
  ;; Input Transformations
```

```
  ((English-To-Semantic-Form ?English ?Semantic-Form)
   (Transform-English-To-Semantic-Form ?English ?Semantic-Form)
   (Write ?Semantic-Form))
```

```
  ((Transform-English-To-Semantic-Form (Is Everything Done)
    (Goal Knowledge Person (Believe Person (Everything-Done)))) !)
```

```
  ((Transform-English-To-Semantic-Form (Which Wrench Should I Use To Tighten The Generator Bolts)
    (Goal Knowledge Person
     (Believe Person (Prohibition Tighten-Generator-Bolts (((Wrench) ?*) . ?*)))))) ')
```

```
  ((Transform-English-To-Semantic-Form (Why Did You Tell Me . ?Obligation-And-State)
    (Justify (Inform Person (Physical-Fact ?Transformed-Obligation-And-State))))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form
    (Where Is The . ?Obligation-And-State)
    (Goal Knowledge Person (Believe Person (Physical-Fact ?Transformed-Obligation-And-State))))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form
    (Where Are The . ?Obligation-And-State)
    (Goal Knowledge Person (Believe Person (Physical-Fact ?Transformed-Obligation-And-State))))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form (I Want The . ?Obligation-And-State)
    (Goal Physical Person (Believe Person (Physical-Fact ?Transformed-Obligation-And-State))))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form
    (How Do I Get The . ?Obligation-And-State)
    (Goal Knowledge Person
     (Believe Person
      (Actions-For-Obligation ?Obligation Physical
       ((Action-Description Physical Person ?* ?* ?* ?*
        (Additions
         (Believe Person
          (Physical-Fact ?Transformed-Obligation-And-State))
          ?* ?*)))))))
    (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State)
    (Eq (?Obligation-Name ?*) ?Transformed-Obligation-And-State)
    (Get-Obligation-Name ?Obligation ?* ?Obligation-Name) !)
```

```
  ((Transform-English-To-Semantic-Form
    (Is The ?Obligation-And-State)
    (Goal Knowledge Person (Believe Person (Physical-Fact ?Transformed-Obligation-And-State))))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form
    (The . ?Obligation-And-State) (Believe Person ?Transformed-Obligation-And-State))
   (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)
```

```
  ((Transform-English-To-Semantic-Form ?Thing ?Thing))
```

```
  ((Obligation-And-State (Socket Wrench Is In The Corner) (At-Socket-Wrench Corner)) !)
```

```
  ((Obligation-And-State (Socket Wrench Is At The Generator Base) (At-Socket-Wrench Generator-Base)) !)
```

```

((Obligation-And-State (Socket Wrench Is At The Pump Base) (At-Socket-Wrench Pump Base)) !)

((Obligation-And-State (Screwdriver At The Generator Base) (At-Screwdriver Generator-Base)) !)

((Obligation-And-State (Screwdriver Is In The Corner) (At-Screwdriver Corner)) !)

((Obligation-And-State (Socket Wrench) (At-Socket-Wrench . ?*)) !)

((Obligation-And-State (Screwdriver) (At-Screwdriver . ?*)) !)

((Obligation-And-State (Generator Bolts) (At-Generator-Bolts . ?*)) !)

((Obligation-And-State (Generator Installed) (Generator Installed)) !)

((Obligation-And-State (Generator Is Not Installed) (Physical-Fact (Generator Not-Installed))) !)

((Obligation-And-State (Generator Shakes) (Generator-Shakes)) !)

;; Output Transformations

((Semantic-Form-To-English ?English ?Semantic-Form)
 (Transform-Semantic-Form-To-English ?English ?Semantic-Form)
 (Write ?Semantic-Form))

((Transform-Semantic-Form-To-English (Since The Action Is Hard To Undo You Should Be Careful
    To Do Everything Properly Before You . ?Transformed-Action-Description)
 (Reversible ?Action-Description No))
 (Action-Description-To-English ?Transformed-Action-Description (?Action-Description)) !)

((Transform-Semantic-Form-To-English
 (To Tighten The Generator Bolts You Should Use A Socket Wrench)
 (Prohibition Tighten-Generator-Bolts (((Wrench) (Socket-Wrench)))))) !)

((Transform-Semantic-Form-To-English (A ?Object Is . ?Description)
 (Description ?Object ?Description)) !)

((Transform-Semantic-Form-To-English
 (To Execute The Action Of ?Transformed-Action-Description) (Execute-Action ?Action-Description))
 (Action-Description-To-English ?Transformed-Action-Description ?Action-Description) !)

((Transform-Semantic-Form-To-English (To Achieve State With . ?Obligation-And-State)
 (Believe Person (Physical-Fact ?Transformed-Obligation-And-State)))
 (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)

((Transform-Semantic-Form-To-English
 (The ?Obligation-And-State) (Physical-Fact ?Transformed-Obligation-And-State))
 (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)

((Transform-Semantic-Form-To-English (The . ?Obligation-And-State) ?Transformed-Obligation-And-State)
 (Obligation-And-State ?Obligation-And-State ?Transformed-Obligation-And-State) !)

((Transform-Semantic-Form-To-English (To Install Generator You Must Lift It Then Tighten Bolts Then Attach Power
Cable)
 (Actions-For-Obligation
  Generator Physical
  ((Action-Description
   Physical Person
   (Install-Generator)
   (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
   (Body Serial-And (((Tighten-Generator-Bolts) (Attach-Power-Cable))))
   (Restrictions (Believe Person (Physical-Fact (Generator Not-Installed)) ?*))
   (Additions (Believe Person (Physical-Fact (Generator Installed))))
   (Deletions (Believe Person (Physical-Fact (Generator Not-Installed))))))))) !)

((Transform-Semantic-Form-To-English (To Turn The Power Off Use The Switch In The Corner)
 (Actions-For-Obligation

```

```

Power Physical
((Action-Description
  Physical Person
  (Turn-Off-Power)
  (Preconditions (Believe Person (Physical-Fact (At-Person Corner))))
  (Body)
  (Restrictions (Believe Person (Physical-Fact (Power On)) ?*))
  (Additions (Believe Person (Physical-Fact (Power Off))))
  (Deletions (Believe Person (Physical-Fact (Power On)))))) !)

((Transform-Semantic-Form-To-English (To Lift A Heavy Generator Use A Fork Lift)
  (Actions-For-Obligation
    At-Generator Physical
    ((Action-Description
      Physical Person
      (Fork-Lift-Generator)
      (Preconditions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))
        (Believe Person (Physical-Fact (At-Fork-Lift Generator-Base))))
      (Body)
      (Restrictions (Believe Person (Physical-Fact (Generator-Type Heavy)) ?*))
      (Additions (Believe Person (Physical-Fact (At-Generator Generator-Base-Top))))
      (Deletions (Believe Person (Physical-Fact (At-Generator Generator-Base-Bottom)))))) !)
    ))

((Transform-Semantic-Form-To-English ?Semantic-Form ?Semantic-Form))

((Action-Description-To-English (Tighten The Generator Bolts) (Tighten-Generator-Bolts)))

((Action-Description-To-English (Install Generator) (Install-Generator)))

((Action-Description-To-English (Lift Generator) (Lift-Generator)))

```

END

1 | 7 | 0 | 4 | 8 | 6

FIN