Mprolog as an expert system development tool

Grant Edward Wright
University of Nevada, Las Vegas
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
MPROLOG as an expert system development tool

Wright, Grant Edward, M.S.
University of Nevada, Las Vegas, 1990
MPROLOG as an Expert System

Development Tool

by

Grant Wright

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Department of Computer Science
University of Nevada, Las Vegas
May 1990
The thesis of Grant Wright for the degree of Master of Science in Computer Science is approved.

Chairperson, John T. Minor III, Ph.D.

Examining Committee Member, Yonina S. Cooper, Ph.D.

Examining Committee Member, Evangelos A. Yfantis, Ph.D.

Graduate Faculty Representative, Craig Walton, Ph.D.

Graduate Dean Ronald W. Smith, Ph.D.

University of Nevada, Las Vegas
May 1990
Abstract

The difficult process of designing expert systems has caused the development of many useful expert system design tools. A new tool, called MPROLOG, has been developed into an expert system tool by giving the designer the power of a programming language, PROLOG, along with a method for specifying uncertainties. Descriptions of KEE and ART, two popular expert system design tools, and MPROLOG are presented along with a description of perhaps the most important phase of expert system design: knowledge acquisition. An analysis of the implementation of an MPROLOG expert system, "the F-111 Wing Commander", throughout the knowledge acquisition and design phases is also documented. Finally, an evaluation of MPROLOG as an expert system design tool is presented.
Acknowledgements

All research on this thesis has been supported by the U.S. Army research Office under Grant DAAL03-87-G-0004. Also the support and guidance provided by Dr. John Minor and the rest of the Computer Science faculty at the University of Nevada, Las Vegas, have been very instrumental in my research.
1. Introduction

1.1 What expert systems are

Expert systems are designed to perform the tasks currently performed by human experts. The specialized knowledge a human expert has is placed in the expert system program. If constructed properly, the expert system will take the same information that an expert would request then output the same kind of decision that an expert would. These programs are useful because there usually is a shortage of qualified human experts in a given field. Also, the computer can be fully informed of all changing facts at all times and will never get fatigued or temperamental. An expert system can apply the expertise of several human experts, if so programmed, and can free human experts to do more creative things. Many expert systems have already been developed performing tasks like medical diagnosis, scientific analysis, electronic design and more.

Expert systems are usually divided into a knowledge base and an inference program part. The knowledge base is kept separate because it must be allowed to grow and change, while the inference program part is not changed. The knowledge base is usually made up of decision rules and facts, which the program uses when a result is output. Since the expert will make decisions on factors with a certain degree of truth, an uncertainty factor is often associated
with knowledge base information. For instance a medical diagnosis program may have a rule stating "the patient has disease1 with confidence level of 89% if x-rays show symptom A with a confidence level of 85% and symptoms C and D are present". The role of the program part would simply be asking the user for input, processing the proper rules and outputting an acceptable result.

The construction of expert systems is called knowledge engineering. Acquiring the knowledge is the most important step because the expert system will only be as good as the knowledge base it uses. The programmer, or knowledge engineer, must have some familiarization with the domain of the expert system. Then the knowledge engineer must tap the expertise of the domain expert, a person that has expertise in the given area, who may not be able to convey his knowledge readily. The knowledge engineer must organize the information into a series of facts and decision rules and design a program to process the rules that will then output a solution the expert thinks is feasible. Problems arise in rule design because of uncertainty, since an expert will make decisions on factors with a certain confidence level. When the confidence levels of facts in the knowledge base have been determined with accuracy, the likelihood of the expert system doing the intended job is much greater.

1.2 Focus of this paper

Choosing the tool of implementation is the next problem in expert system design. Most expert systems were designed using ad hoc methods. They were designed from the ground up and used system dependent/domain dependent
strategies. There are many useful expert system design tools, among which are KEE (knowledge engineering environment) and ART (automated reasoning tool). The main problem with these tools is cost; these systems are not free even to universities. A university researcher has developed a free expert system tool called MPROLOG. This tool essentially provides a designer with the programming power of PROLOG (a logic programming language), as well as providing the representation power for a rule-base with confidence levels. Descriptions of these expert system design tools are presented in chapter 2.

MPROLOG is used to implement the simulated expert system discussed in this thesis. A general description of the knowledge acquisition phase in expert system design is presented in chapter 3. A description of the expert system, "the F-111 wing commander", and its implementation, starting with the knowledge acquisition and design phase, are discussed in chapter 4. The problems encountered developing the expert system and problems due to MPROLOG will also be presented in chapter 4. Finally, an evaluation of MPROLOG as an expert system development tool will make up chapter 5.
2. Expert System Design Tools

2.1 Introduction

There are a number of tools available for expert system design. Two of the most popular commercial tools are discussed in sections 2.2 and 2.3. Following these in section 2.3 is a discussion of a new tool called MPROLOG. Here is a set of terms common in expert system design:

rule: a simple IF <cond> THEN <result>, where the <result> gets evaluated when the <cond> is true.

Horn Clause: logic rule of the form p :- q1,q2,...,qn. where p is true if all goals q1,...,qn are satisfied (found to be true).

backward-chaining: goal directed reasoning. Back-chained rules respond to goals by trying to satisfy them.

forward-chaining: data directed reasoning. Actions are taken when rules are satisfied with facts.

monotonic reasoning: the number of true things in the system increase over time. Once something is found true, it will not be shown false later.

non-monotonic: the number of true things varies over time.

Backtracking: when a system is given a set of goals to satisfy for some rule, if any of the goals fail then a backtracking system will review what has happenned so far and make a new attempt to satisfy the goal. If the system
runs out of options trying to satisfy the goal then the rule with the given conjunction of goals fails.

2.2 The Automated Reasoning Tool

Automated Reasoning Tool (ART) is an expert system development mechanism created at Inference Corporation. The current version is ART 3.0 which was developed in mid 1986 and fully supports the following implementations: ART/LISP which runs on dedicated LISP workstations, ART/UNIX which has versatile graphics and windowing capabilities and is supported on Sun-3 workstations, and ART/VMS which runs on Digital Equipment Corporation's VAX minicomputers. Many commercially developed expert systems use ART. Some of these include NAVEX, a navigation expert system designed at NASA, Authorizer's Assistant, a credit card charge authorization system developed at American Express Corporation, and Master Scheduling Unit, a production scheduler for new automated plants which was developed at Westinghouse Electric Corporation.

In ART, a fact is a fundamental piece of knowledge consisting of a proposition and an extent. A proposition is a statement of existence or value while an extent describes the circumstances under which a fact has meaning. A fact could be something like "the car is red". The statement "the car is red" would be the proposition while "at this particular time" would be the intended extent. A schema is a multi-fact description of an object. ART automatically provides for characteristics of related schemata to be inherited. Observe the following schema example:
Inferences are made through inherited properties in facts and schemata. In the above example, a dog is a predator with fixed claws. It is also a mammal, an inherited characteristic, so it is warm-blooded and fur-covered. It is also an Animal, another inherited characteristic, and has the attributes of an animal. Inheritance allows the programmer to build complex concepts by relating them to more general concepts. ART can infer these relationships by pattern-matching schemata characteristics through the inference engine.

ART rules are used to infer procedural knowledge in expert systems. ART can use both forward chaining rules, and backward chaining rules. Rules stored in an ART database can be applied whenever the described situation arises, while a rule in a conventional system is only applied when the statement is executed. Associated with ART rules is a salience declaration, which is used by the ART scheduler to determine the order in which rule activations are processed. The scheduler always activates rules with the highest salience value for execution.

The viewpoint mechanism models hypothetical alternative plans and can explore processes that change over time. Viewpoints can be manipulated by hypothetical rules, constraint rules and belief rules. ART builds a net-like structure of viewpoints by exploring all paths to a possible goal. Unproductive
viewpoint branches can be removed by "poisoning", which is used to eliminate dead-end viewpoints.

Rule types:

Hypothetical rules: when the condition is satisfied, one or more hypothetical viewpoints are spawned for ART to process in parallel until they can be resolved into one confirmed reality.

Constraint rules: when the condition is true a viewpoint is poisoned (ART discards the viewpoint).

Belief rules: the condition of this is an objective that must be reached. When this objective is reached, the viewpoint becomes "believed". This is the way desirable viewpoints are recognized.

When an expert system is designed in ART, the knowledge-base is constructed as facts and schemata. The knowledge engineer then creates the rule-base by translating the rules from the domain expert into ART rules. ART is data-driven, so rules get applied anytime facts match the conditions. The patterns of the rule satisfied by a fact are called instantiated. If the rule still contains uninstantiated patterns, ART may invoke backward-chaining to satisfy the remaining patterns. If all patterns are matched then the rule is sent to the agenda, where the rules saliency will determine when it will be executed. The cycle continues until the agenda is empty. The way the process grows is that rules often create more facts to be matched against the entire set of rules. Facts that are still true during a new cycle will not make a rule "fire" again, thus the same rule does not continue firing infinitely.
2.3 The Knowledge Engineering Environment

KEE, the Knowledge Engineering Environment, was developed by Intellicorp in 1983. The latest release, KEE 3.1, was made in 1987 and features a C integration toolkit which allows a C programmer to access the KEE environment and also allows C functions to be run in the KEE environment. KEE also supports both monotonic and non-monotonic reasoning. Many of the 500 sites with KEE have developed expert systems for decision support, diagnosis & repair, planning, modeling & simulation, real time, and other applications. KEE runs on DEC VAXstation II, Symbolics 3600 series, Sun-3 workstations and Xerox 1100 series machines among others.

KEE operates using frame-based representation in reasoning. Frames are great inference tools since they provide mechanisms for inheriting characteristics (basically, frames are the same as schemata in ART). The frames are called units in KEE, and attributes are represented as slots. KEE units can contain consistency checking mechanisms, dynamic subroutines and can inherit characteristics from other units. Units can be put into hierarchies and classed into knowledge bases. Fully modeled knowledge bases are called KEEworlds.

example:

unit: thing from knowledge base ex1
own slot: mass
own slot: location

unit: living thing from knowledge base ex1
inheritance slot: thing
own slot: reproduction
own slot: growth
As in ART, KEE can make inferences through inherited properties. It is obvious what the inherited relationships are. The dog is a predator with fixed claws and is also a mammal and is also an animal and is also a living thing and is also a thing. These are all linked through the inheritance slot.

Production rules in KEE are represented as units. Rule premises and conclusions are slot values in those units. Since rules are units, they can be classed. Rules can be forward-chaining, backward-chaining, or both. Backtracking also occurs when appropriate. The basic rule types in KEE are deduction rules, which infer facts from other facts and state constraints to remove undesirable paths (like poisoning viewpoints in ART), and Action Rules, which add/delete facts and can start new search paths (new viewpoints in ART).

Rules that are activated are placed on an agenda which controls the order in which they are executed. A language called Tell and Ask controls how rules reference the frame structure. The language has logical operators to evaluate
the frame values. Like ART, KEE is data-driven, so information in units has rules applied to it when its patterns match something. The rules then get sent to an agenda mechanism and get executed. Overall the methods of KEE and ART are the same, most differences are notational and structural (ART facts and schemata versus KEE units).

2.4 MPROLOG as an expert system design tool

MPROLOG is an expert system development tool designed by Dr. John Minor and implemented by Martin Flatebo in 1988 to complete a Masters degree in computer science at UNLV. MPROLOG is written in Common LISP on the Symbolics 3600 series machines. The purpose of MPROLOG is to advance the capabilities of a powerful programming language, PROLOG, by supplementing the language with multi-valued logic capabilities. Standard PROLOG uses simple two-valued logic leaving no room for uncertainties. MPROLOG allows programmers to store information with uncertainties based on minimal bounded fuzzy logic. The value of PROLOG as an expert system development tool is greatly enhanced if uncertainties are attached since this information can be put directly into the knowledge base. It allows the programmer a mechanism for implementing the knowledge of the expert in uncertain cases.

The fundamental piece of knowledge in MPROLOG is the fact. Facts are basically the same as predicates in first order logic. For instance the fact likes(john, mary) could mean "john likes mary". A group of facts with the same predicate name could be thought of as a database relation. The following example demonstrates this:
Group of Facts
plane( 111, fighter, damaged ).
plane( 121, fighter, available ).
plane( 211, bomber, damaged ).
plane( 221, bomber, available ).

Plane database relation
\[ \begin{array}{ccc}
\text{plane#} & \text{type} & \text{status} \\
111 & \text{fighter} & \text{damaged} \\
121 & \text{fighter} & \text{available} \\
211 & \text{bomber} & \text{damaged} \\
221 & \text{bomber} & \text{available} \\
\end{array} \]

MPROLOG is not frame-based like ART and KEE. The inheritable properties from other relations will only be extracted with rules. Rules are simply Horn clauses which are reasoned over with backward-chaining. When a rule fails, backtracking occurs in an attempt to resatisfy the rule. If the rule can not be satisfied it simply fails.

MPROLOG facts look like this:
\[ p(pc)(X1,X2,\ldots,Xn). \]
where \( p \) is the fact name (predicate name)
\( pc \) is the optional uncertainty factor, or truth value \((0 \leq pc \leq 1)\)
\( X1,\ldots,Xn \) are the variables (Attributes of the fact)
MPROLOG rules look like this:

\[
\begin{align*}
p(pc)(X_1, X_2, ..., X_m) &::= q_1(pc_1)(Q_{11}, Q_{12}, ..., Q_{1i}), \\
&\quad q_2(pc_2)(Q_{21}, Q_{22}, ..., Q_{2j}), \\
&\quad \ldots \\
&\quad \ldots \\
&\quad q_n(pc_n)(Q_{n1}, Q_{n2}, ..., Q_{nk}).
\end{align*}
\]

where \( p \) is the head of the rule,
\( pc_i \)'s are the optional uncertainty factors, or truth values (0 \( \leq \) \( pc \) \( \leq \) 1)
\( q_1, q_2, ..., q_n \) are predicates/values that make up the tail of the rule.
\( X_i \)'s and \( Q_{ij} \)'s are the variables for the respective predicates involved.

A rule \( p \) is satisfied (the head is true) if each of \( q_1, ..., q_n \) are true with the truth value of at least \( pc_i \). Any \( q_i \) matching facts are true, and any \( q_i \) that matches the head of a rule will cause MPROLOG to attempt satisfying that rule in the same way. MPROLOG works by resolving rules against rules and facts.

The first rule is invoked at the MPROLOG prompt "?-". At this point, the question invokes the system. For instance, a system with the above rule could be invoked with "?- \( p(A_1, A_2, ..., A_n) \).".
3. Knowledge Acquisition

3.1 Introduction

The process of knowledge acquisition is basically the most important phase of expert system design. The human expert, or domain expert, must be capable of relating his expertise to the expert system programmer, or knowledge engineer. The knowledge engineer needs to be a quick learner because he must understand enough of the domain, the scope of the expert system, so he can question the domain expert effectively. He should also be an effective communicator so he can remain "in sync" with the domain expert, he should conceptualize well, he should be very organized and keep records, and should have insight into diverse areas of knowledge.

Tasks of a knowledge engineer
- analyzing information flow
- determining program structure
- working with experts to obtain information
- performing design functions

3.2 Tasks and Problems

The knowledge engineer must perform the listed tasks well. Analyzing information flow is important because this is how the knowledge engineer
reveals the domain expert's methodologies. Setting up frameworks of these methodologies is how the program structure will be determined. All of the knowledge base will be developed by working with the domain expert, and the overall design of the expert system are the responsibilities of the knowledge engineer.

The domain expert must have patience. He must realize the knowledge engineer may not grasp much of the domain so he should be able to communicate effectively at a lower level. The most important thing is he must explain important decision factors and heuristics accurately. Ability to relate methods used to solve problems is most beneficial and explaining how judgements are made with missing information make the knowledge to rule transition much simpler.

3.3 Extracting Knowledge from the Domain Expert

How the knowledge engineer extracts information from the domain expert is also of importance. Sometimes only interviews are needed because the expert can easily identify his problem solving techniques and relate needed heuristics. This is the case when knowledge of the domain expert is declarative in nature. The knowledge engineer may have trouble extracting information from the domain expert because the processes used in solving the problems may be second nature to the expert. This is a reflection of procedural knowledge. To extract procedural knowledge, the knowledge engineer should use many simulations and interviews and employ process tracing techniques, such as having the expert "think aloud" during simulation
problem solving. Extracting episodic knowledge is best done in this manner also. Episodic knowledge is autobiographical and experiential information that is chunked by episodes in long term memory. It is the knowledge of things that have become so routine, that a person has difficulty stating rules on doing it. To extract semantic knowledge, the expert may use repertory grid analysis, in which the expert constructs a model of how he organizes his world. Task analysis, where the knowledge engineer breaks down the tasks performed into the most primitive components and process tracing are other techniques that are available for extracting this type of knowledge. Success of the expert system depends heavily on the semantic knowledge because this is where information about facts, concepts and their relationships to each other are stored.

Types of knowledge

procedural: skills an individual "knows how" to perform, often reactionary to stimuli (has difficult time stating rules on how to do this)

declarative: "Knowing that". Surface level info experts can verbalize. Expressions of what an expert is aware of knowing

episodic: autobiographical, experiential info that the expert has grouped or chunked by episodes. Very difficult to extract.

semantic: organized knowledge about the following:
(1) words and verbal symbols
(2) word/symbol meanings and usage rules
(3) word/symbol referents and interrelationships
(4) algorithms for manipulating symbols, concepts, relations.

The problems of the knowledge acquisition phase can place heavy burdens upon the knowledge engineer. First, the knowledge engineer must familiarize himself with the domain. This involves mostly terminology and specific
concepts common to the people for whom the expert system is being developed. Then the way the domain expert organizes his world must be conceptualized. The largest problem is the ability of the expert to express knowledge, or correspondingly, the ability of the knowledge engineer to extract information from the domain expert. The knowledge engineer can analyze the types of knowledge and methods to acquire it and yet miss critical bits and pieces. These bits and pieces make the final problem of verification difficult. The validation of the knowledge base can take much time, as will verification of the system. The fine tuning that is done during the verification phase will be a good indication of how well the knowledge engineer extracted the domain expert's expertise.

3.4 Factors in decision rule design

The design of decision rules becomes an issue because the domain expert usually provides multiple rules to be applied at certain instances. The rules will likely share certain attributes and will probably have some differing attributes. The following categories of decision rules make up the common approaches to selecting alternative decision rules.

**Dominance rule:** for alternative rules A and B, if A is better than B on at least one attribute and no worse on all other attributes then A is selected as the rule of choice.

**Conjunctive decision rule:** a set of important values for attributes is established by the expert. Any of the alternative rules that do not have one or more of these attributes are discarded.
Disjunctive decision rules: a set of important values for attributes is established by the expert. The alternative selected is one that exceeds on at least one attribute while other alternatives are beneath it on this criterion value.

Lexicographic decision rule: attributes are rank-ordered by the expert. The alternative chosen has the highest possible rank-ordering.

Elimination by aspects rule: attributes are ranked-ordered. A set of criterion values for each attribute is also established. Alternatives below any of the criterion value of the highest ranked attribute are eliminated continually until only one alternative remains.

Maximum attribute attractiveness rule: criterion values for the attributes are again set up. The expert compares these values to the corresponding values in each alternative. The alternative equalling or exceeding the most criterion values are chosen.

3.5 Difficulties

Obviously, many problems face the knowledge engineer when entering the realm of knowledge acquisition. There are not many knowledge engineering methodologies to begin with. The techniques that are available in knowledge acquisition are unfamiliar to most knowledge engineers. For those familiar with the techniques, selecting the appropriate ones may be difficult. Simply selecting the domain expert can be a problem, especially if access to him is limited. These problems only intensify in situations where more than one domain expert is needed. When these problems are overcome, the knowledge engineer finally starts the knowledge acquisition phase.
4. Designing an MPROLOG Expert System: The F-111 Wing Commander

4.1 Description of the expert system domain

4.1.1 Expectations from an expert in the domain

The domain of this simulated MPROLOG expert system is an F-111 wing commander circa October 1975. The wing commander is expected to assign planes, weapons and weapon fusings to missions which field personnel want to see implemented. The conditions of the mission affect the number of planes needed, the types of weapons used on the mission, and the fusing of those weapons. The commander issues a mission suggestion if the needed planes and weapons are available. The commander does not assign the individual pilots, this is done at a lower level.

Most of this assigning of planes and weapon loads is done by looking up missions given certain factors like low altitude cover and terrain (which affects target visibility and hit/miss probability), structure materials (affecting load types), defensive positioning and anti-aircraft weaponry (slightly affecting weapon loads but affecting number of planes needed).
These factors are looked up in mission implementation manuals, the suggestions are noted, and the commander estimates the number of planes needed and the type of loads needed for a mission consisting of the given factors. An intelligence officer keeps the commander briefed on current conditions on all missions.

4.1.2 Expectations for the Expert System Program

The expert system is expected to perform the duties of the wing commander. It will interact with the field personnel by querying about the type of mission, problems which can possibly affect the mission, and the desired completion level for the mission. If the mission can be implemented, the expert system should make a suggestion by listing the number of planes, types of weapons, and types of weapon fusings needed to carry out the request. If the mission cannot be implemented, the system should cite possible factors for this, such as insufficient supplies or too many difficult factors.

The expert system is expected to maintain databases for missions, planes and weapons. Thus field personnel can check the status of all such items. The expert system also allows changes and deletions to a certain degree in the mission database. Changes to the plane and weapon databases are unlimited, but there is a password system protecting access to these databases. A standards file is used when starting up the expert system from scratch. The field personnel can also edit a saved session of the expert system. Changes can be made to the system to reflect more current conditions, but no special capability for the opinions of an intelligence officer was provided.
4.2 Description of the domain expert

The domain expert, Colonel E. Kowalczyk, USAF retired, was a pilot of F-111 aircraft during the Vietnam War. He has extensive knowledge of missions implemented during this era, and knows the expected effectiveness for these missions. His knowledge is largely declarative, and he had no problems relating heuristic values or describing critical factors. He had a great deal of patience in explaining much of the military jargon, semantic knowledge, and was helpful in eliminating unimportant factors and establishing restrictions on other factors.

4.3 Design of the expert system

The expert system consists of databases for planes, weapons, missions, and mission components. The plane database keeps track of plane status and availability. The weapon database consists of just two attributes, weapon name and amount available. The mission database consists of many attributes: mission number, type, desired completion level, start time, location, and number of planes. Some mission component databases also exist keeping track of data associated with mission type, and planned mission parts consisting of the number of planes carrying a certain load with a certain weapon fusing.

4.3.1 Design of the framework for the program
The expert system is menu driven. The menu breakdown was developed by Dr. Minor, who developed the idea for the F-111 wing commander expert system.

The top level of the expert system has 5 choices: plan mission, change mission, check status, change database and quit the system. The plan mission section is where all of the expert knowledge will be used. All other sections are used to configure, change, and check the databases.

The plan mission section is used to develop a mission frame. A mission frame is as follows:

```
mission number : 1
location: locname
start-time: 1200
mission type: interdiction
number of planes assigned: 33
desired completion percentage: 60-75%
```

Also associated is a mission type frame and a list of planes assigned with corresponding weapon loads and fusings.

The plan mission section starts by querying for location and start-time of the mission. The mission number is supplied by the system. Then the user is asked to select from the following mission types: interdiction against a target, area preparation, close-air support, and 24 hour alert. The interdiction selection also brings up a menu on target types: personnel concentration, unarmoured vehicles, armoured vehicles, building complex, roads/railroads, and bridges. Selection of any interdiction target types, or the other mission types excluding 24 Hour alert result in a series of menus designed to get a proper description of important factors for this chosen mission (see figure 1 for list of these...
factors). All of these factors are represented as menus except for 24 hour alert, which simply asks for the number of planes. The final menu records the desired completion percentage for the mission, except for 24 hour alert missions. The desired completion percentage for a 24 hour alert is assumed to be 100%.

The expert system takes all these mission factors and resolves them against rules to figure out the number of planes needed and load types and fusings needed for the mission to meet the desired completion level. If there are enough planes and weapons to meet mission specifications, the system suggests planes and weapons to use to the user who can assign or scrap this suggestion. Assigning causes the system to subtract out the number of planes used and amount of each weapon used from the appropriate databases. If the number of planes available is insufficient for the mission, or if the amount of any weapon is insufficient then the system tells the user the mission can not be implemented. Factors are cited as to why the mission can not be implemented look like this:

mission requires X planes but only y available
not enough of weapon X to implement mission

Some important factors limiting the mission completion may also be cited:

Heavy foliage makes completion level difficult.
Urban cover makes completion level difficult.
Dense Fog makes completion level difficult.
Mountainous terrain makes completion level difficult.
Attempting to drop completion level can aid you.
4.3.2 Knowledge acquisition phase

The knowledge acquisition stage is interleaved with the design of the program. Initially, Dr. Minor worked on the menu framework design with the domain expert. The mission types were established, and the factors involved in each mission were listed. A method of supplying values for these factors was needed, so giving the user menus was chosen because it would limit difficulties in MPROLOG input/output. Menus also allowed the programmer a simple way of defining ranges of values to be input. The information flow for the expert system program had been established, but more interviewing was needed to get menu ranges. The expert supplied much semantic information that was used in developing the menus, such as dividing force sizes into values like squads, platoons, battalions, and dividing anti-aircraft guns and SAM-site menus into section, battery, battalion.

The most important menu established was the desired completion percentage. This menu required many changes. The expert recalled most missions with above 60% completion were considered successful with an acceptable degree of satisfaction. Missions with over 75% completion were considered excellent successes, but missions with over 90% completion were unlikely while missions under 50% completions were implementable with fewer planes. This led to a percentage range breakdown of: 20-44%, 45-59%, 60-74%, 75-89%, and over 90%. The 60-74% range was considered the basis value for the facts in the rule-base. Since above 90% was so unlikely, the expert said the number of planes needed to implement a mission at that level would be 3 times the number needed at the 60-74% level. The 75-89% level was considered 1.5 times
as difficult while 45-59% and 20-44% were considered 0.8 and 0.4 times as difficult respectively.

Other factors were decided declaratively. These included things like anti-aircraft guns and SAM-sites. The expert estimated the number of planes these devices could eliminate at each of their respective menu levels. The expert estimated multiplier values for these menus: terrain, protective cover and low visibility. These multipliers were created to increase the number of planes to a level that would make the desired completion level attainable. The number of planes required to implement every mission type at every level was also recorded, and all such knowledge was placed in the rule-base.

The load types and fusings associated with each mission type was also recorded. Certain mission types could be affected by defensive positions and protective cover menus. These menus require different weapon loads on these mission types.

There were few decision rule requirements for the system. Since the number of planes for each mission type and each menu level were in the database, alternative rules were no problem. Weapon loads were determined in a similar fashion and again alternative rules would not occur. This was due to the menu implementation which by supplying specific menu values steered the weapon load assignments. If the weapon load type and fusing are considered the attributes of the rule, then one can see that the conjunctive decision rule is employed. The deciding criterion to be met will be one representing an attribute passed out of a menu.
4.3.3 Implementing the rule base

Implementing the rule base was done as follows. Each mission had specific plane and weapon assignments for each menu value. The basis completion percentage value was 60-74%. Thus if the menu for mission type type_x had menu values 1, 2, and 3, and these menu values required 5, 10, and 20 planes to allow completion at the basis level then the rules would look like this:

\[
\text{assn\{.60\} (type}_x,1,5). \\
\text{assn\{.60\} (type}_x,2,10). \\
\text{assn\{.60\} (type}_x,3,20).
\]

To find the values of these at different percentages, the following rules would be used:

\[
\text{assn\{.20\} (type}_x,M,NP) :- \text{assn\{.60\} (type}_x,M,N), \text{NP is } N*0.4. \\
\text{assn\{.45\} (type}_x,M,NP) :- \text{assn\{.60\} (type}_x,M,N), \text{NP is } N*0.8. \\
\text{assn\{.75\} (type}_x,M,NP) :- \text{assn\{.60\} (type}_x,M,N), \text{NP is } N*1.5. \\
\text{assn\{.90\} (type}_x,M,NP) :- \text{assn\{.60\} (type}_x,M,N), \text{NP is } N*3.0.
\]

This is how all initial plane assignments are handled. The actual number of planes can be higher because of other multipliers as well as addition of planes from factors like anti-aircraft guns and SAM-sites.

Load types and fusings are figured by asserting appropriate percentages of weapons for each mission type. The weapon types and fusings vary on specific menu criterion that characterize a mission. Load rules look like this:
load_missiontype( MissionNum, Factors, Percentages ) :-
    assert( loadpc( MissionNum, Load1, Fuse1, Percent1) ),
    assert( loadpc( MissionNum, Load2, Fuse2, Percent2) ),
    .
    .
    assert( loadpc( MissionNum, Loadn, Fusen, Percentn) ).

The Percentages variable is needed for missions that have more than one set of load constraints. Area preparation missions, for example, are affected by defensive positioning and protective cover.

4.3.4 Code decisions

Implementing the weapon loads as assertions to a new loadpc database was necessary for many reasons. The most important reason was the absence of list access in MPROLOG. The easiest way of implementing the loads would be constructing a list of load and fuse tuples. The system would avoid costly assertions and deletions to unnecessary databases. A similar database called load_amt was created later to find the actual amounts of each weapon for the mission. Then a database with tuples for mission number, number of planes, load on the planes, fusing for the load, and load amount was also created to store the actual mission data. Even this could have been implemented more efficiently as lists.

Certain allowances had to be made for underflow. Missions against small forces with low desired completion levels tended to need less than 2 planes.
This caused zero planes carrying nothing to be asserted into the database because most missions needed more than 2 weapon loads. This was solved using the Maximum attribute attractiveness rule. The expert supplied weapon types for each mission that were considered most essential. These weapons only were asserted in cases of underflow.

4.4 Examples

All examples have been tested using the Wing Commander expert system. The menu ordering is that found when running the Wing Commander. All mission suggestions and failure factors were provided by the Wing Commander expert system. All cases start by choosing "plan mission" from the menu in figure 2(m).

4.4.1 Example 1

Close air support mission: a company is to be provided with close air support. The company is travelling over rough terrain under a jungle cover. Enemy positions will be underground (in tunnels). No armored vehicles, anti-aircraft guns, SAM-sites are expected to be encountered. Low altitude visibility is clear and the mission should be carried out to 75% satisfaction level. Set location as "loc 1" and time at 0830 hours.
Menu (figure number) Selection
---------------------- --------------
Select Mission Type(2r) Close-air Support
Select Principle Terrain Type(2u) Mountainous / Rough
Select protective cover type(2o) Heavy Forest / Jungle
Select type of defensive position(2i) Tunnels
Est. number of armored vehicles(2d) none
Est. number of Anti-aircraft guns(2a) none
Est. number of SAM-sites(2q) none
Low altitude cover(2k) clear
Desired completion percentage(2h) 75 - 89%

The Wing Commander system output the following mission suggestion:

5 planes will carry 60 units of napalm with an impact fusing
5 planes will carry 100 units of 500lb hi drag bombs with an impact fusing
1 planes will carry 16 units of cluster bombs with a proximity fusing
4 planes will carry 16 units of 2000lb bombs with a delay fusing
All 15 planes will carry a total of 30000 rounds of 20mm

4.4.2 Example 2
Bridge mission: A large concrete bridge is to be destroyed. The bridge crosses
a gorge of approximately 250 feet. Expect a battery of anti-aircraft guns and a
section of SAM-sites. Visibility is hazy due to heavy rain in the area. A 50%
completion level will be satisfactory. Set location as "loc 2" and time at 1130
hours.

Menu (figure number) Selection
---------------------- --------------
Select Mission Type(2r) Interdiction
Select Principle target type (2s) Bridge
Est. number of Anti-aircraft guns(2a) 3 - 6 (section)
Est. number of SAM-sites(2q) 1 - 2 (battery)
Low altitude cover(2k) hazy / heavy rain / fog patches
Desired completion percentage(2h) 45 - 59%

The Wing Commander system output the following mission suggestion:

2 planes will carry 32 units of 750lb bombs with an delay fusing
2 planes will carry 8 units of 2000lb bombs with a delay fusing
4 planes will carry 96 units of air-to-ground missiles with an impact fusing
All 8 planes will carry a total of 16000 rounds of 20mm
4.4.2 Example 3

Area preparation: an area of more than one square mile has been requested to be cleared. The area is in a mountainous area in thick jungle. Enemy positions include a set of bunkers and a platoon of armored vehicles. No anti-aircraft devices are expected and visibility will likely be clear. Completion level of 70% is satisfactory. Set location as "loc 3" and time at 0310 hours.

<table>
<thead>
<tr>
<th>Menu (figure number)</th>
<th>Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Mission Type(2r)</td>
<td>area preparation</td>
</tr>
<tr>
<td>Select area in square miles (2b)</td>
<td>1 - 1 1/2</td>
</tr>
<tr>
<td>Select Principle terrain type (2u)</td>
<td>Mountainous</td>
</tr>
<tr>
<td>Select protective cover type(2o)</td>
<td>Heavy Forest / Jungle</td>
</tr>
<tr>
<td>Select type of defensive position(2i)</td>
<td>Reinforced concrete bunkers</td>
</tr>
<tr>
<td>Est. number of armored vehicles(2d)</td>
<td>1 - 4 (platoon)</td>
</tr>
<tr>
<td>Est. number of Anti-aircraft guns(2a)</td>
<td>none</td>
</tr>
<tr>
<td>Est. number of SAM-sites(2q)</td>
<td>none</td>
</tr>
<tr>
<td>Low altitude cover(2k)</td>
<td>clear</td>
</tr>
<tr>
<td>Desired completion percentage(2h)</td>
<td>60 - 74%</td>
</tr>
</tbody>
</table>

This mission will fail. The Wing Commander would state this as follows:

Not enough planes implement mission. 57 needed but only 50 available.
Mountainous terrain makes completion level difficult.
Heavy foliage make completion level difficult.
Attempting to drop desired completion level may aid you.

If the mission is attempted again at 45-59% completion, the result is:

12 planes will carry 192 units of cluster bombs with a proximity fusing
2 planes will carry 40 units of 500lb hi drag bombs with an impact fusing
2 planes will carry 32 units of 750lb bombs with an impact fusing
2 planes will carry 8 units of 2000lb with an impact fusing
15 planes will carry 60 units of 2000lb bombs with a delay fusing
15 planes will carry 240 units of 750lb bombs with a delay fusing
All 48 planes will carry a total of 96000 rounds of 20mm
4.5 Problems with the wing commander expert system

4.5.1 Limitations due to program framework and MPROLOG

The framework of the program is very restrictive. In the era of the wing commander, planes could carry mixed loads. Only certain loads could be mixed on any one plane. This could have been implemented if the list notation of PROLOG existed in MPROLOG.

The system automatically quits resolving against rules when there are insufficient amounts of any weapon load. Suggesting the mission anyway and simply telling the user that it can not be implemented would have been better.

The numbers for most of the mission suggestions seem to "round". A problem occurs with floating point numbers, which end up getting represented as strings. Multiplication of these numbers, which lose accuracy being represented as strings, compound numerical error. Factors affecting completion percentage had to be processed by furnishing multipliers to add in more planes. MPROLOG had no way of providing a upper bound limit on the completion percentage.

4.5.2 Verification

Verifying the expert systems operation could not fully be carried out. The only source to verify the system was the domain expert since access to the military information needed would be next to impossible (because of the time
required to get it and unlikelihood that this old information would be available). The verification consisted, more or less, of showing the domain expert a series of missions suggested by the system for given inputs. These simulations were carried out on an array of different mission types, and in each case the domain expert evaluated the results of the program and suggested corrections. Adjustments to the rule base were made until the expert verified that subsequent simulations run through the program produced satisfactory results.
<table>
<thead>
<tr>
<th>mission type</th>
<th>Affecting factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 Hour Alert</td>
<td>number of planes to put on alert.</td>
</tr>
<tr>
<td>close-air support</td>
<td>force-size to support, terrain type, protective cover,</td>
</tr>
<tr>
<td></td>
<td>defensive positions, armored vehicles,</td>
</tr>
<tr>
<td></td>
<td>anti-aircraft guns, surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>area preparation</td>
<td>size of area to prepare, terrain type, protective</td>
</tr>
<tr>
<td>cover</td>
<td>defensive positions, armored vehicles,</td>
</tr>
<tr>
<td></td>
<td>anti-aircraft guns, surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>interdiction type</td>
<td>force size of targeted personnel, terrain type, protective</td>
</tr>
<tr>
<td></td>
<td>cover, anti-aircraft guns, surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>personnel concentration</td>
<td>size of vehicle convoy, terrain type, protective cover, anti-aircraft guns,</td>
</tr>
<tr>
<td></td>
<td>surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>unarmored vehicles</td>
<td>size of vehicle convoy, terrain type, protective cover, anti-aircraft guns,</td>
</tr>
<tr>
<td></td>
<td>surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>armored vehicles</td>
<td>size of vehicle convoy, terrain type, protective cover, anti-aircraft guns,</td>
</tr>
<tr>
<td></td>
<td>surface-to-air missile sites, low visibility</td>
</tr>
<tr>
<td>building complex</td>
<td>area of complex, building materials, anti-aircraft guns, surface-to-air missile</td>
</tr>
<tr>
<td></td>
<td>sites, low visibility</td>
</tr>
<tr>
<td>road/railroad cuts</td>
<td>number of cuts, road-type, anti-aircraft guns, surface-to-air missile sites, low</td>
</tr>
<tr>
<td></td>
<td>visibility</td>
</tr>
<tr>
<td>bridge</td>
<td>length of bridge, building materials, anti-aircraft guns, surface-to-air missile</td>
</tr>
<tr>
<td></td>
<td>sites, low visibility</td>
</tr>
</tbody>
</table>

Figure 1
Estimate number of anti-aircraft guns

none
1 - 2 (section)
3 - 6 (battery)
7 - 18 (battalion)
over 18

(a)

Select area in square miles

0 - 1/4
1/4 - 1/2
1/2 - 1
1 - 1 1/2
1 1/2 - 2

(b)

Select CHANGE you wish to make.
NOTE: mission TYPE can be changed !!!
Delete this mission and re-plan !!!
Delete this mission
Location
Start Time
Desired Completion %
Make no changes

(c)

Estimate Number of Armored Vehicles

none
1 - 4 (platoon)
5 - 12 (company)
13-36 (battalion)
more than 36

(d)

Figure 2
Estimate length of bridge in feet
under 50
50 - 100
100 - 200
200 - 300
over 300

Select Bridge Material
Wood
Concrete
Steel

Select Building Material Type
Wood, straw, or tents
Sandbag reinforced hut
Brick
Reinforced concrete

Estimate desired mission completion level
90 - 100%
75 - 89%
60 - 74%
45 - 59%
20 - 44%

Select Principle Type of Defensive Position
None
Trenches/Earth works
Tunnels
Reinforced concrete bunkers/caves

Figure 2 (cont)
Select Approximate Force Size
Squad (1 - 10)
Platoon (11 - 30)
Company (31 - 100)
Battalion (101 - 350)
Regiment (351 - 1000)
LARGER (over 1000)

(j)

Low altitude cover
dense fog / smoke
haze / heavy rain / fog patches
light rain / drizzle
clear

(k)

Fighter Mission Dispatching System
Plan mission
Change mission
Check status
Change database
Quit

(m)

Road/RR cuts
standard cut approx 25-50 ft
simple cut
double cut
intersection/fork
major junction

(n)

Figure 2 (cont)
Select protective cover type
open
light trees / scattered buildings
Heavy forest / Jungle
City / Urban area

Road Type / Rail Underlining
roadtype
dirt
macadam / rock
concrete

Estimate number of SAM sites
none
1 - 2 (section)
3 - 6 (battery)
7 - 18 (battalion)
over 18

Select Mission Type
Interdiction
Area Preparation
Close-air Support
On 24-hour Alert

Figure 2 (cont)
Select Principle Target Type
Personnel Camp
Unarmored vehicles (convoy)
Armored vehicles
Building complex
Roads/railroads
Bridge

(s)

Figure 2 (cont)

Estimate Number of unarmoured Vehicles
more than 20
16 to 20
11 to 15
5 to 10
less than 5

(t)

Select Principle terrain type
flat
rolling hills
mountainous / rough

(u)

Figure 2 (cont)
5. Evaluation of MPROLOG

5.1 MPROLOG's advantages

MPROLOG has a distinct advantage over other expert system design tools. PROLOG is familiar to everyone in the AI community, thus most knowledge engineers would also be familiar with it. MPROLOG is basically PROLOG with optional uncertainty factors. The presence of backtracking allows MPROLOG to perform non-monotonic reasoning, a feature present in both KEE and ART. Database facts are basically the same in all three systems. The rule types in KEE and ART can be modeled in MPROLOG. Both KEE and ART have more messy LISP-like implementations. A person designing an expert system in KEE or ART must have a good understanding of LISP, must learn the syntax of the tool, and must learn to work with a new system dependent environment. MPROLOG only requires the designer to understand PROLOG. MPROLOG does not require training of system personnel to keep it running, it can simply be loaded onto a system and run like any PROLOG implementation.

MPROLOG features a way of updating information in the database with the built-in predicates support and detract. These clauses work by "adding" some confidence percentage to a given rule when support is used, and "subtracting" some confidence percentage when detract is used. In diagnostic expert systems applications, this would be incredibly useful.
5.2 MPROLOG problems

Some of MPROLOG's advantages become its problems. The fact that MPROLOG is based on PROLOG gives it the power of a programming language. Also, the problems associated with PROLOG are inherited. Basic counting is a task in PROLOG. Asserting and deleting database items is not order preserving. Infinite looping and recursion can occur much more easily in PROLOG. Some predicates allow the programmer to write rules that are unsafe.

List notation was left out of MPROLOG, and implementing list operations using the lisp predicate is incredibly difficult. Using lisp quote notation seems impossible, and the lack of the "I" is a major drawback. Floating point handling must also be improved and expressions should be evaluated as arguments to functions and predicates before being passed. Another helpful fix would involve the uncertainty factor. If the system provided ways to allow variables in the uncertainty factor field, MPROLOG would be more powerful. This would lessen the changes necessary when using support and detract on rule-base facts. If the variable in the factor field can be instantiated then the system will be able to use the actual value for any other purposes.

Procedural attachments available in ART and KEE (using KEE methods) are not available in MPROLOG. Through these procedural attachments, both ART and KEE support object oriented programming. The inheritance properties in ART and KEE provide easy classification methods not present in MPROLOG. Since MPROLOG has the power of a programming language, some these properties and attachments can be created. Overall, the problems of MPROLOG will hinder
the expert system programmer, but all of these problems are mendable. If mended, MPROLOG will be very useful.
References


Appendix:

Wing Commander Source Code
/*
Structure of Wing Commander expert system

<table>
<thead>
<tr>
<th>level</th>
<th>description</th>
<th>predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plan Mission</td>
<td>process1(1,..)</td>
</tr>
<tr>
<td>1.1</td>
<td>Interdictions</td>
<td>process11</td>
</tr>
<tr>
<td>1.1.1</td>
<td>personnel concentration</td>
<td>process11(pc,..)</td>
</tr>
<tr>
<td>1.1.2</td>
<td>unarmored vehicles</td>
<td>process11(uv,..)</td>
</tr>
<tr>
<td>1.1.3</td>
<td>armored vehicles</td>
<td>process11(av,..)</td>
</tr>
<tr>
<td>1.1.4</td>
<td>building complex</td>
<td>process11(bc,..)</td>
</tr>
<tr>
<td>1.1.5</td>
<td>roads / railroads</td>
<td>process11(rr,..)</td>
</tr>
<tr>
<td>1.1.6</td>
<td>bridges</td>
<td>process11(bridge,..)</td>
</tr>
<tr>
<td>1.2</td>
<td>Area Preparation</td>
<td>process1(areaPrep,..)</td>
</tr>
<tr>
<td>1.3</td>
<td>Close-air Support</td>
<td>process1(closeAirSupport,..)</td>
</tr>
<tr>
<td>1.4</td>
<td>On 24 Hour Alert</td>
<td>process1(on24HrAlert,..)</td>
</tr>
<tr>
<td>2</td>
<td>Change Mission</td>
<td>process2(2,..)</td>
</tr>
<tr>
<td>2.1</td>
<td>Delete</td>
<td>process2(Mnum,del)</td>
</tr>
<tr>
<td>2.2</td>
<td>Location</td>
<td>process2(Mnum,loc)</td>
</tr>
<tr>
<td>2.3</td>
<td>Time</td>
<td>process2(Mnum,time)</td>
</tr>
<tr>
<td>2.4</td>
<td>Completion %</td>
<td>process2(Mnum,cmp)</td>
</tr>
<tr>
<td>3</td>
<td>Check Status</td>
<td>process3(3,..)</td>
</tr>
<tr>
<td>3.1</td>
<td>Status of Planes</td>
<td>process3(1)</td>
</tr>
<tr>
<td>3.2</td>
<td>Status of Weapon Loads</td>
<td>process3(2)</td>
</tr>
<tr>
<td>3.3</td>
<td>Status of Missions</td>
<td>process3(3)</td>
</tr>
<tr>
<td>4</td>
<td>Change Database</td>
<td>process4(4,..)</td>
</tr>
<tr>
<td>4.0</td>
<td>Change Password for Access</td>
<td>process4(0)</td>
</tr>
<tr>
<td>4.1</td>
<td>Change Plane Amounts</td>
<td>process4(1)</td>
</tr>
<tr>
<td>4.1.1</td>
<td>number of unassigned</td>
<td>process41(ua)</td>
</tr>
<tr>
<td>4.1.2</td>
<td>number on maintenance</td>
<td>process41(sm)</td>
</tr>
<tr>
<td>4.1.3</td>
<td>number with battle damage</td>
<td>process41(bd)</td>
</tr>
<tr>
<td>4.1.4</td>
<td>number malfunctioning</td>
<td>process41(mf)</td>
</tr>
<tr>
<td>4.2</td>
<td>Change weapon amounts</td>
<td>process42(2)</td>
</tr>
<tr>
<td>4.2.1</td>
<td>amount of 20mm</td>
<td>process42(mm20)</td>
</tr>
<tr>
<td>4.2.2</td>
<td>amount of Cluster Bombs</td>
<td>process42(cb)</td>
</tr>
<tr>
<td>4.2.3</td>
<td>amount of 500lb high drag</td>
<td>process42(hd500)</td>
</tr>
<tr>
<td>4.2.4</td>
<td>amount of 500lb low drag</td>
<td>process42(ld500)</td>
</tr>
<tr>
<td>4.2.5</td>
<td>amount of 750lb</td>
<td>process42(m750)</td>
</tr>
<tr>
<td>4.2.6</td>
<td>amount of 2000lb</td>
<td>process42(m2000)</td>
</tr>
<tr>
<td>4.2.7</td>
<td>amount of napalm</td>
<td>process42(napalm)</td>
</tr>
<tr>
<td>4.2.8</td>
<td>amount of air-to-ground</td>
<td>process42(atg)</td>
</tr>
<tr>
<td>5</td>
<td>Quit</td>
<td>process5(5,..)</td>
</tr>
</tbody>
</table>

round(X,Y) :-
  W is X + 0.5,
  lisp(Y,truncate,W).

trunc(X,Y) :- lisp(Y,truncate,X).

bk_amt(mm20,0). /* Keeps track of load usage during weapon assignment */
bk_amt(cb,0). /* In case same weapon used with different fusing */
bk_amt(hd500,0). /* Must be reset every time a mission is planned */
bk_amt(ld500,0).
bk_amt(m750,0).

*/
bk_amt(m2000,0).
bk_amt(napalm,0).
bk_amt(atg,0).

name_of(mm20,"20 mm") :- !.
name_of(cb,"cluster bombs") :- !.
name_of(hd500,"500lb high drag bombs") :- !.
name_of(ld500,"500lb low drag bombs") :- !.
name_of(m750,"750lb bombs") :- !.
name_of(m2000,"2000lb bombs") :- !.
name_of(napalm,"napalm") :- !.
name_of(atg,"air to ground missiles") :- !.

expert :- /* Start the database by typing expert at the '?' prompt */
clear_db,
lisp(Ans,newdb),
open_proper(Ans),!,
do(0).

clear_db :-
mission(M,_,Sel,_,_),
retract(mission(M,_,Sel,_,_)),
kill_loadamts(M), kill_sugg(M), kill_loadpc(M),
removeTypeAttr(M,Sel),
clear_db.

clear_db :- cpass, ccounts, cplanes, cweaps.

cpass :-
password(_),
replace(password(_),password(secret)),
cpass.
ccounts :-
count(_), miscount(_),
retract(count(_)),
retract(miscount(_)).
ccounts.
cplanes :-
uaplanes(_),
retract(uaplanes(_)), retract(asplanes(_)), retract(bdplanes(_)),
retract(mfplanes(_)), retract(smplanes(_)).
cplanes.
cweaps :-
weapon(mm20,_),
retract(weapon(mm20,_)), retract(weapon(cb,_)), retract(weapon(hd500,_)),
retract(weapon(ld500,_)), retract(weapon(m750,_)), retract(weapon(m2000,_)),
retract(weapon(napalm,_)), retract(weapon(atg,_)).
cweaps.

open_proper(n) :-
ask('Supply file name of your database (in quotes with .mprolog): ',Name),
[Name],nl,
write(' File ',Name,' is in the work space...'),nl.
open_proper( y ) :-
    ["standards.mprolog"]).

do(Activity) :-
    Activity = quit,
    lisp(Ans, exitMenu),
    saveDB(Ans).

saveDB(Ans) :- /* Save session in a file */
    Ans = 'y',
    write(" Supply prefix file name (without .mprolog) to save database in:"),
    read(SaveMe), nl,
    tell(SaveMe),
    countListing,
    missionListing,
    planeListing,
    weaponListing,
    list1, list2, list3,
    list4, list5, list6, list7, list8,
    list9, list10, list11, list12,
    list13, list14, list15,
    told.

saveDB(Ans) :- told.

countListing :- listing( count ), listing( miscount ).

countListing.

planeListing :-
    listing(uaplanes), listing(smplanes), listing(bdplanes),
    listing(mfplanes), listing(asplanes).

planeListing.

weaponListing :- listing(weapon).

weaponListing.

missionListing :- listing(mission).

missionListing.

list1 :- listing(new_pl_amt).

list1.

list2 :- listing(bk_amt).

list2.

list3 :- listing(password).

list3.

list4 :- listing(personnel).

list4.

list5 :- listing(uVehicles).

list5.

list6 :- listing(aVehicles).

list6.

list7 :- listing(buildings).

list7.
(*/users/masl/gwright/lonnie/gex.mprolog*)

```
list7.
list8 :- listing(cuts).
list8.
list9 :- listing(bridge).
list9.
list10 :- listing(areaPrep).
list10.
list11 :- listing(closeAirSupport).
list11.
list12 :- listing(on24HrAlert).
list12.
list13 :- listing(loadpc).
list13.
list14 :- listing(load_amt).
list14.
list15 :- listing(sugg_miss).
list15.

do(Activity) :- !, process(Activity, Next), do(Next).

process(0, Next) :- /* Process selections from menu */
    lisp(Selection, menu0),
    process(Selection, Next).

process(1, 0) :- nl, /* Plan Mission level 1 */
    write('Plan mission'), nl,
    missionNum(Mnum),
    resetbk,
    ask('Enter mission location: ', Mloc),
    ask('Enter mission starting time: ', Mstart),
    lisp(Selection, selectMission),
    process1(Selection, Mnum, Inter),
    get_cp( Sel, Comp ),
    assignPlanes(Mnum, Selection, Inter, Comp, NPs), !,
    round( NPs, Planes ),
    !,
    implement_mission( Mnum, Mloc, Mstart, Planes, Selection, Inter, Comp ).

get_cp(on24HrAlert, 0).
get_cp( Sel, Comp ) :-
    lisp(Comp,compperc).

resetbk :-
    replace(bk_amt(mm20,),bk_amt(mm20,0)),
    replace(bk_amt(cb,),bk_amt(cb,0)),
    replace(bk_amt(hd500,),bk_amt(hd500,0)),
    replace(bk_amt(ld500,),bk_amt(ld500,0)),
    replace(bk_amt(m750,),bk_amt(m750,0)),
    replace(bk_amt(m2000,),bk_amt(m2000,0)),
    replace(bk_amt(napalm,),bk_amt(napalm,0)),
    replace(bk_amt(atg,),bk_amt(atg,0)).
```
assignPlanes(M, on24HrAlert, none, Comp, Nplanes) :-
on24HrAlert( M, Nplanes ),
load_24hr( M ).

assignPlanes(M, closeAirSupport, none, Comp, TotPlanes) :-
closeAirSupport( M, FS, Terr, PC, DP, AV, AA, SS, LC ),
assnpc( Comp, closeAirSupport, FS, Nplanes ),
assign_std_factors( AA, SS, AddPlanes ),
assign_avs( AV, AddMorePlanes ),
compute_terr_factors( Terr, PC, LC, TFactor ),!;

X is Nplanes * TFactor,
Y is AddPlanes + AddMorePlanes,
TotPlanes is X + Y,

cond_load( M, closeAir, DP ).

assignPlanes(M, areaPrep, none, Comp, TotPlanes) :-
areaPrep( M, AR, Terr, PC, DP, AV, AA, SS, LC ),
assnpc( Comp, areaPrep, AR, Nplanes ),
assign_std_factors( AA, SS, AddPlanes ),
assign_avs( AV, AddMorePlanes ),
compute_terr_factors( Terr, PC, LC, TFactor ),!;

X is Nplanes * TFactor,
Y is AddPlanes + AddMorePlanes,
TotPlanes is X + Y,

load_ap( M, 0.75 ),
cov_and_def( M, PC, DP ).

assignPlanes(M, interdiction, pc, Comp, TotPlanes) :-
personnel( M, FS, Terr, Pcov, DP, AA, SS, LC ),
assnpc( Comp, pc, FS, Nplanes ),
assign_std_factors( AA, SS, AddPlanes ),
compute_terr_factors( Terr, Pcov, LC, TFactor ),!;

X is Nplanes * TFactor,
TotPlanes is X + AddPlanes,
cond_load( M, pc, DP ).
/*
 *  load_pers( M, 0.5 ),
 cov_and_def( M, PC, DP ). */
cond_load( M, pc, none ) :-
  load_pers( M, 1.0 ).
cond_load( M, pc, DP ) :-
  load_pers( M, 0.75 ),
  defpos_loads( M, 0.25, DP ).
cov_and_def( M, PC, none ) :-
  cover_loads( M, 0.25, PC ).
cov_and_def( M, PC, DP ) :-
  cover_loads( M, 0.125, PC ),
  defpos_loads( M, 0.125, DP ).

assignPlanes(M, interdiction, uv, Comp, TotPlanes) :-
  uVehicles( M, UV, Terr, PC, AA, SS, LC ),
  assnpc( Comp, uv, UV, Nplanes ),
  assign_std_factors( AA, SS, AddPlanes ),
  compute_terr_factors( Terr, PC, LC, TFactor ), !,
X is Nplanes * TFactor,
TotPlanes is X + AddPlanes,
  load_uv( M, 1.0 ).
/*
 *  load_uv( M, 0.5 ),
 cover_loads( M, 0.5, PC ). */
assignPlanes(M, interdiction, av, Comp, TotPlanes) :-
  aVehicles( M, AV, Terr, PC, AA, SS, LC ),
  assnpc( Comp, av, AV, Nplanes ),
  assign_avs( AV, AddPlanes ),
  compute_terr_factors( Terr, PC, LC, TFactor ), !,
X is Nplanes * TFactor,
TotPlanes is X + AddPlanes,
  load_av( M, 1.0 ).
/*
 *  load_av( M, 0.5 ),
 cover_loads( M, 0.5, PC ). */
assignPlanes(M, interdiction, bc, Comp, TotPlanes) :-
  buildings( M, Area, Mats, AA, SS, LC ),
  assnpc( Comp, bc, Area, Nplanes ),
  assign_std_factors( AA, SS, AddPlanes ), !,
TotPlanes is Nplanes + AddPlanes,
  load_buildmats( M, 1.0, Mats ).
assignPlanes(M, interdiction, rr, Comp, TotPlanes) :-
cuts( M, NC, RT, AA, SS, LC ),
assnpc( Comp, rr, NC, Nplanes ),
assign_std_factors( AA, SS, AddPlanes ), !,

    TotPlanes is Nplanes + AddPlanes,
load_cuts( M, 1.0, RT).

assignPlanes(M, interdiction, bridge, Comp, TotPlanes) :-
    bridge( M, Len, Mat, AA, SS, LC ),
assnpc( Comp, bridge, Len, Nplanes ),
assign_std_factors( AA, SS, AddPlanes ), !,

    TotPlanes is Nplanes + AddPlanes,
load_bridgemats( M, 1.0, Mat ).

ask(Question, Response) :- write(Question),
                    read(Response), nl, write("response: ", Response, ">"), nl.

process1(interdiction, Mnum, Selection) :- lisp(Selection, targetType),
process11(Selection, Mnum).

process11(pc, Mnum) :-
    lisp(ES, forceSize),
lisp(Terrain, terrain),
lisp(Pcov, pcover),
lisp(DefPos, defpos),
airdef(AA, SS),
lisp(LC, locover),
replace(personnel(Mnum, _, _, _, _),
            personnel(Mnum, ES, Terrain, Pcov, DefPos, AA, SS, LC)).

replace(C1, C2) :- C1, retract(C1), assert(C2).
replace(C1, C2) :- assert(C2).

process11(uv, Mnum) :-
uvehicles(UV),
lisp(Terrain, terrain),
lisp(Pcov, pcover),
airdef(AA, SS),
lisp(LC, locover),
replace(uVehicles(Mnum, _, _, _, _, _), uVehicles(Mnum, UV, Terrain, Pcov, AA, SS, LC)).

process11(av, Mnum) :-
    lisp(AV, uvehicles),
lisp(Terrain, terrain),
lisp(Pcov, pcover),
airdef(AA, SS),
lisp(LC, locover),
replace(uVehicles(Mnum, _, _, _, _), uVehicles(Mnum, UV, Terrain, Pcov, AA, SS, LC)).
process1(bc,Mnum) :-
    lisp(SF,buildingarea),
    lisp(Material, buildingMats),
    lisp(AA,SS),
    lisp(LC,locover),
    replace(buildings(Mnum, SF, Material), buildings(Mnum, AA, SS, LC)).

process1(rr,Mnum) :-
    lisp(FC,ncuts),
    lisp(RT,roadtype),
    lisp(AA,SS),
    lisp(LC,locover),
    replace(cuts(Mnum, FC, RT, AA, SS, LC)).

process1(bridge,Mnum) :-
    lisp(Len,bridgelen),
    lisp(Material, bridgeMats),
    lisp(AA,SS),
    lisp(LC,locover),
    replace(bridge(Mnum, Len, Material), bridge(Mnum, AA, SS, LC)).

process1(areaPrep, Mnum, none) :-
    lisp(Area, area),
    lisp(Terrain, terrain),
    lisp(Pcov, pcover),
    lisp(DefPos, defPos),
    lisp(AV, armoredVehicles),
    lisp(AA,SS),
    lisp(LC,locover),
    replace(areaPrep(Mnum, Area, Terrain, Pcov, DefPos, AV, AA, SS, LC)).

armoredVehicles(AV) :- lisp(AV,av2).

uvehicles(UV) :- lisp(UV,uvehicles).

airdef(X,Y) :- aaGuns(X), samSites(Y).

aaGuns(AAguns) :- lisp(AAguns,aadef).

samSites(SAMsites) :- lisp(SAMsites,samsites).

process1(closeAirSupport, Mnum, none) :-
    lisp(ForceSize, forceSize),
    lisp(Terrain, terrain),
    lisp(Pcov, pcover),
    lisp(DefPos, defPos),
    lisp(AV, armoredVehicles),
    lisp(AA,SS),
    lisp(LC,locover),
    replace(closeAirSupport(Mnum, ForceSize, Terrain, Pcov, DefPos, AV, AA, SS, LC)).

process1(on24HrAlert, Mnum, none) :-

14:57 Apr 19 1990
write('24 hour alert'), nl, uaplanes(Count), nl, write('Number of unassigned planes = '), write(Count), nl, write('How many planes are to be put on 24 hour alert? '), read(NumPlanes), nl, replace(on24HrAlert(Mnum,_,on24HrAlert(Mnum,NumPlanes)).

suggest( M ) :- /* Output a "suggestion" for the given mission M */
   sugg_miss( M, Load, Fuse, Amt, Planes ),
   name_of(Load,Lname),
   print_line( Load, Planes, Amt, Lname, Fuse ),
   fail.

suggest( M ).

print_line( mm20, Planes, Amt, Lname, Fuse ) :-
   write("-----------------------------------------------"),nl,
   write("All ",Planes," will also carry ",Amt," rounds of 20mm "),nl,nl,nl.

print_line( Load, Planes, Amt, Lname, Fuse ) :-
   Load |= mm20,
   nl.

/*
 * implement_mission( Mnum, Nplanes, MissionType, InterdictionType )
 */

At this point, an attempt to implement the mission is put forth.
Failure in implement mission occurs if the number of planes available
is insufficient. Failure in calculations results if an augmented
plane amount is still insufficient. Failure in more_calculations
results when there are insufficient weapon loads of any kind.

implement_mission( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :-
   uaplanes( X ),
   X >= NP,
   !,
   calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ).

implement_mission( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :-
   uaplanes(X),
   write("Not enough planes in database for mission ",Mnum),nl,
   write("Mission requires ",NP," planes, only ",X," available."),nl,nl,
   excuse( Mnum, Sel, Inter, Comp, NP ),
   finally_implement( 2, Mnum, Sel, Inter, NP ).

calculations( Mnum, Mloc, Mstart, NP, on24HrAlert, none, Comp ) :-
   uaplanes( X ),
   X >= NP,
   special_24( Mnum, NP ),
   more_calculations( Mnum, Mloc, Mstart, NP, on24HrAlert, none, Comp ).

calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :-
   Sel |= on24HrAlert,
calc_load_amts( Mnum, NP ),
new_pl_amt( Mnum, NNP ),
uaplanes( X ),
X >= NNP,
retract( new_pl_amt( Mnum, NNP ) ),
!,
more_calculations( Mnum, Mloc, Mstart, NNP, Sel, Inter, Comp ).

calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :-
not( new_pl_amt(_,J ),
too_few( Mnum, Sel, Inter, NP),
more_calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ).

calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :- /* Not enough of weapon */
uaplanes(X),
new_pl_amt( Mnum, NNP ),
retract( new_pl_amt( Mnum, NNP ) ),
write("Not enough planes in database for mission ",Mnum),nl,
write(" Mission requires ",NNP," planes, only ",X," available."),nl,nl,
excuse( Mnum, Sel, Inter, Comp, NP ),
finally_implement( 2, Mnum, Sel, Inter, NP ).

special_24( Mnum, NP ) :- /* Kludge to avoid approximations on 24Hr Alert mission */
L1 is .45*NP,
L3 is .10*NP,
trunc(L1,R1),
trunc(L1,R2),
round(L3,R3),
figure(NP,R1,R2,R3,X,Y),
loadpc( Mnum, LD1, F1, PC1 ),
retract( loadpc( Mnum, LD1, F1, PC1 ) ),
loadpc( Mnum, LD2, F2, PC2 ),
retract( loadpc( Mnum, LD2, F2, PC2 ) ),
loadpc( Mnum, LD3, F3, PC3 ),
retract( loadpc( Mnum, LD3, F3, PC3 ) ),
std_load_amts( LD1, Amt1 ),
std_load_amts( LD2, Amt2 ),
std_load_amts( LD3, Amt3 ),
AN1 is Amt1*X,
AN2 is Amt2*Y,
AN3 is Amt3*R3,
load_and_suggest( Mnum, AN1, LD1, F1, X ),
load_and_suggest( Mnum, AN2, LD2, F2, Y ),
load_and_suggest( Mnum, AN3, LD3, F3, R3 ).

figure( NP, X, Y, Z, NX, NY ) :-
T is X + Y + Z,
NP = T,
NX is X,
NY is Y.

figure( NP, X, Y, Z, NX, NY ) :-
X = Y, XXX is X + 1,
figure( NP, XXX, Y, Z, NX, NY ).
figure( NP, X, Y, Z, NX, NY ) :- YYY is Y + 1, figure( NP, X, YYY, Z, NX, NY ).

set_npa( A, B, C ) :- B > C, A is B.
set_npa( A, B, C ) :- A is C.

too_few( Mnum, Sel, Inter, NP ) :- /* Too few is evoked when underflow occurs */
    kill_loadamts( Mnum ), kill_sugg( Mnum ),
    princ_load( Mnum, Sel, Inter, Lode, Fuse ),
    std_load_amts( Lode, Amt ),
    weapon( Lode, AmtAvail ),
    AmtNeeded is NP * Amt,
    AmtAvail > AmtNeeded,
    assert( load_amt( Mnum, Lode, Fuse, AmtNeeded ) ),
    assert( sugg_miss( Mnum, Lode, Fuse, AmtNeeded, NP ) ).

too_few( Mnum, Sel, Inter, NP ) :-
    princ_load( Mnum, Sel, Inter, Lode, Fuse ),
    assert( load_amt( Mnum, Lode, Fuse, 0 ) ).

too_few( A, B, C, D ).

more_calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :-
    'calc_20( Mnum, NP ),
    not( load_amt( Mnum, _, _, 0 ) ), nl, nl, nl,
    write(" Here is a suggestion for implementing mission ", Mnum), nl,
    write("------------------------------------------------------------------------------------------"), nl, nl,
    suggest( Mnum ),
    lisp( Ans, assnmission ),
    finally_implement( Ans, Mnum, Sel, Inter, NP ),
    assert_if_necessary( Ans, Mnum, Mloc, Sel, Mstart, NP, Comp ).

more_calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp ) :- /* Not enough of weapon */
    report_failures( Mnum ),
    excuse( Mnum, Sel, Inter, Comp, NP ),
    !,
    finally_implement( 2, Mnum, Sel, Inter, NP ).

load_20( Mnum, NP ) :-
    std_load_amts( mm20, Amt ),
    weapon( mm20, Old ),
    SP is Amt*NP,
    NewAmt is Old - SP,
    replace( weapon(mm20,Old), weapon(mm20,NewAmt) ).

assert_if_necessary( 1, Mnum, Mloc, Sel, Mstart, NP, Comp ) :-
    assert( mission( Mnum, Mloc, Sel, Mstart, NP, Comp ) ),
    uaplanes( X ), asplanes( Z ), miscount( R ),
    Y is X - NP, W is Z + NP, S is R + 1,
    replace( uaplanes(X), uaplanes(Y) ),
replace( asplanes(Z), asplanes(W) ),
replace( miscount(R), miscount(S) ).

assert_if_necessary( 2, Mnum, _, _, _, _) :- kill_sugg( Mnum ).

/* excuse cover possible reasons for a missions failure */

excuse( Mnum, closeAirSupport, none, Comp, N ) :- !,
closeAirSupport( Mnum, _, Terrain, Pcover, _, _, _, LC),
terrExcuse( Terrain ),
pcovExcuse( Pcover ),
compExcuse( Comp ).

excuse( Mnum, areaPrep, none, Comp, N ) :- !,
areaPrep( Mnum, _, Terrain, Pcover, _, _, _, LC),
terrExcuse( Terrain ),
pcovExcuse( Pcover ),
compExcuse( Comp ).

excuse( Mnum, interdiction, pc, Comp, N ) :- !,
personnel( Mnum, _, Terrain, Pcover, _, _, _, LC),
terrExcuse( Terrain ),
pcovExcuse( Pcover ),
compExcuse( Comp ).

excuse( Mnum, interdiction, uv, Comp, N ) :- !,
uVehicles( Mnum, _, Terrain, Pcover, _, _, LC),
terrExcuse( Terrain ),
pcovExcuse( Pcover ),
compExcuse( Comp ).

excuse( Mnum, interdiction, av, Comp, N ) :- !,
aVehicles( Mnum, _, Terrain, Pcover, _, _LC),
terrExcuse( Terrain ),
p covExcuse( Pcover ),
compExcuse( Comp ).

excuse( Mnum, interdiction, bc, Comp, N ) :- !,
buildings( Mnum, _, _, _, _, LC ),
locovExcuse( LC ),
compExcuse( Comp ).

excuse( Mnum, interdiction, rr, Comp, N ) :- !,
cuts( Mnum, _, _, _, _, LC ),
locovExcuse( LC ),
compExcuse( Comp ).

excuse( Mnum, interdiction, bridge, Comp, N ) :- !,
bridge( Mnum, _, _, _, _, LC ),
locovExcuse( LC ),
compExcuse( Comp ).

excuse( Mnum, X, Y, Z, W ) :- !.
compExcuse( 0 ).
compExcuse( 1 ).
compExcuse( 2 ).
compExcuse( 3 ) :- nl, write("Attempting to drop completion percentage may aid you.").
compExcuse( 4 ) :- nl, write("Attempting to drop completion percentage can aid you.").
compExcuse( 5 ) :- nl, write("Attempting to drop completion percentage will aid you.").

terrExcuse(mountains) :- nl, write("Mountainous terrain makes completion level difficult.").
terrExcuse(open).
terrExcuse(hilly).

covExcuse(open).
covExcuse(sb).
covExcuse(jungle) :- nl, write("Heavy foliage makes completion level difficult.").
covExcuse(urban) :- nl, write("Urban cover makes completion level difficult.").

covExcuse(1).
covExcuse(2).
covExcuse(3) :- nl, write("Haze/Heavy Rain/Fog Patches make completion level difficult.").
covExcuse(4) :- nl, write("Dense Fog/Smoke make completion level very difficult.").

calc_20( Mnum, NP ) :- /* Find amount of 20mm needed to implement mission */
  std_load_amts( mm20, Amt ),
  AmtNeeded is NP * Amt,
  weapon( mm20, AmtAvail ),
  AmtNeeded <= AmtAvail,
  assert( sugg_miss(Mnum, mm20, impact, AmtNeeded, NP ) ).

calc_20( Mnum, NP ) :-
  assert( load_amt(Mnum, mm20, impact, 0) ).

calc_load_amts( Mnum, NP ) :- /* Find amount of any weapon needed to implement mission */
  loadpc( Mnum, Load, Fuse, P ),
  retract( loadpc( Mnum, Load, Fuse, P ) ),
  !, X is P*NP,
  round(X,Y),
  add_new_tot( Mnum, Y ),
  std_load_amts( Load, AmtStd ),
  AmtNeeded is Y*AmtStd, /* Y is # of planes */
  load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y )!,
  calc_load_amts( Mnum, NP ).

calc_load_amts( Mnum, NP ) .

add_new_tot( M, 0 ).

add_new_tot( M, X ) :-
  new_pl_amt( M, Y ),
  Z is X + Y,
  replace( new_pl_amt(M,Y), new_pl_amt(M,Z) ).
add_new_tot( M, X ) :—  assert( new_pl_amt( M, X ) ).

load_and_suggest( Mnum, AmtNeeded, Load, Fuse, 0 ) . /* do not assert 0 planes !!! */ load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y ) :—
    weapon( Load, AmtAvail ),
    bk_amt( Load, Used ), /* Load Total used on same weapon */
    Total is Used + AmtNeeded, /* (if weapon uses with different fusings) */
    Total =< AmtAvail,
    assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ).

assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ) :—
    load_amt( Mnum, Load, Fuse, Other_Amt ),
    sugg_miss( Mnum, Load, Fuse, Old_Amt, OY ),
    NY is OY + Y, /* Changing #’s in case weapon already asserted */
    Total_Amt is Other_Amt + AmtNeeded,
    replace( load_amt( Mnum, Load, Fuse, Other_Amt ),
             load_amt( Mnum, Load, Fuse, Total_Amt ) ),
    replace( sugg_miss( Mnum, Load, Fuse, Other_Amt, OY ),
             sugg_miss( Mnum, Load, Fuse, Total_Amt, NY ) ).

assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ) :—
    assert( load_amt( Mnum, Load, Fuse, AmtNeeded ) ),
    assert( sugg_miss( Mnum, Load, Fuse, AmtNeeded, Y ) ).

load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y ) :—
    assert( load_amt( Mnum, Load, Fuse, 0 ) ).

report_failures( Mnum ) :—
    load_amt( Mnum, Load, _, Amt ),
    Amt > 0,
    retract( load_amt( Mnum, Load, _, Amt ) ),
    report_failures( Mnum ).

report_failures( Mnum ) :—
    load_amt( Mnum, Load, _, Amt ),
    Amt = 0,
    name_of( Load, WeaponName ),
    write("Hot enough ",WeaponName," to implement mission ",Mnum),
    retract( load_amt( Mnum, Load, _, Amt ) ),
    report_failures( Mnum ).

report_failures( Mnum ).

finally_implement( 1, M, S, I, NP ) :—
    load_20( Mnum, NP ),
    actually_load( M ).

actually_load( Mnum ) :—
    load_amt( Mnum, Load, Fuse, Amt ),
    Amt > 0,
    weapon( Load, X ),
    Y is X — Amt,
replace( weapon(Load,X), weapon(Load,Y) ),
retract( load_amt( Mnum, Load, Fuse, Amt ) ),
actually_load( Mnum ).

actually_load( Mnum ).

finally_implement( 2, M, S, I, NP ) :-
    deletemtype( M, S, I ),
    count( X ),
    Z is X - 1,
    replace( count( X ), count( Z ) ),
    kill_loadpc( M ),
    kill_sugg( M ),
    kill_loadamts( M ).

kill_loadpc( M ) :- loadpc( M, ... ), retract( loadpc( M, ... ) ), kill_loadpc( M ).
kill_loadpc( M ).

kill_loadamts( M ) :-
    load_amt( M, ... ),
    retract( load_amt( M, ... ) ),
    kill_loadamts( M ).

kill_loadamts( M ).

kill_sugg( M ) :-
    sugg_miss( M, ... ),
    retract( sugg_miss( M, ... ) ),
    kill_sugg( M ).

kill_sugg( M ).

deletemtype( M, interdiction, pc ) :- retract( personnel( M, ... ) ).

deletemtype( M, interdiction, uv ) :- retract( uVehicles( M, ... ) ).

deletemtype( M, interdiction, av ) :- retract( aVehicles( M, ... ) ).

deletemtype( M, interdiction, bc ) :- retract( buildings( M, ... ) ).

deletemtype( M, interdiction, rr ) :- retract( cuts( M, ... ) ).

deletemtype( M, interdiction, bridge ) :- retract( bridge( M, ... ) ).

deletemtype( M, closeAirSupport, none ) :- retract( closeAirSupport( M, ... ) ).

deletemtype( M, areaPrep, none ) :- retract( areaPrep( M, ... ) ).

deletemtype( M, on24HrAlert, none ) :- retract( on24HrAlert( M, ... ) ).

std_load_amts( cb, 16 ). /* in fours */
std_load_amts( mm20, 2000 ).
std_load_amts( hd500, 20 ).
std_load_amts( ld500, 24 ).
std_load_amts( m750, 16 ).
std_load_amts( m2000, 4 ).
std_load_amts( napalm, 12 ).
std_load_amts( atg, 24 ). /* in sixes */

princ_load( Sel, Inter, Lode, Fuse ) /*
princ_load( M, interdiction, pc, hd500, impact ).

14:57 Apr 19 1990
princ_load( M, interdiction, uv, atg, impact ).
princ_load( M, interdiction, av, atg, impact ).
princ_load( M, interdiction, bc, Lode, Fuse ) :-
    buildings( M, _, Mats, _, Fuse ),
    pla_appropo( be, Mats, Lode, Fuse ).
princ_load( M, interdiction, rr, Lode, Fuse ) :-
    cuts( M, _, Mats, _, Fuse ),
    pla_appropo( rr, Mats, Lode, Fuse ).
princ_load( M, interdiction, bridge , Lode, Fuse ) :-
    bridge( M, _, Mats, _, Fuse ),
    pla_appropo( bridge, Mats, Lode, Fuse ).
princ_load( M, closeAirSupport, none, hd500, impact ).
princ_load( M, areaPrep, none, m750, delayed ).
princ_load( M, on24HrAlert, none, hd500, impact ).
pla_appropo( be, wood, napalm, impact ).
pla_appropo( be, rhut, m750, proximity ).
pla_appropo( be, brick, m2000, proximity ).
pla_appropo( be, rc, m2000, delayed ).
pla_appropo( rr, 1, m2000, delayed ).
pla_appropo( rr, 2, m750, impact ).
pla_appropo( rr, 3, m750, impact ).
pla_appropo( bridge, wood, m750, impact ).
pla_appropo( bridge, concrete, m2000, delay ).
pla_appropo( bridge, steel, m2000, impact ).
missionNum(Mnum) :-
    count(X),
    Mnum is X + 1,
    not(mission(Mnum, _, _, _)),
    replace(count(X), count(Mnum)).
missionNum(1).
process(5, quit) :- write('Quit').
level(complete, 5, "90 - 100").
level(complete, 4, "75 - 89").
level(complete, 3, "60 - 74").
level(complete, 2, "45 - 59").
level(complete, 1, "20 - 44").
level(complete, 0, "100").

location(Loc) :-
    nl, write('Where are the planes to be based? '),
    read(Loc), nl.

process(2, 0) :- /* Make changes to missions */
    write('Change mission'), nl, nl,
    miscount(Nmiss), Nmiss > 0,
    write('There are ',Nmiss),write(' mission(s) in the database. '),nl, nl,
    listmissions,
    ask('Which mission would you like to change (enter number): ',MN),
    showMission(MN),
    lisp(Which,attribute), nl,
    process2(MN,Which).

process(2, 0) :- miscount(Nmiss), Nmiss = 0, nl, write("There are no planned missions!!"),nl.

process(2, 0) :- nl, write('mission not changed...'), nl.

continue(M) :-
    mission(M, _, _, _, _, _),
    write('New Mission Status: Mission '),nl, nl,
    showMission(M).

continue(M).

process2(M,none).

process2(M,del) :- /* Delete this mission */
    return_weapons(M),
    remove_book_keeping(M),
    miscount(X), Y is X - 1,
    replace( miscount(X), miscount(Y) ),
    retract(mission(M,_,_,_,_,_)).

return_weapons(M) :-
    pull_and_replace_loads(M),!,
    return_weapons(M).

return_weapons(M) :-
    mission( M, _, _, NP, _ ),
    uaplanes( X ),
    Y is X + NP,
    replace( uaplanes( X ), uaplanes( Y ) ),
    asplanes( R ),
    S is R - NP,
replace( asplanes( R ), asplanes( S ) ).

pull_and_replace_loads(M):-
  sugg_miss( M, Lode, Fusing, Amt, _ ),!
  weapon( Lode, Old ),
  New is Old + Amt,
  retract( weapon( Lode, Old ), weapon( Lode, New ) ),
  retract( sugg_miss( M, Lode, Fusing, Amt, _ ) ).

pull_and_replace_loads(M):- fail.

remove_book_keeping(M) :-
  mission(M,_,Type,_,_),
  removeTypeAttr(M,Type).

process2(M,loc) :- /* Change location name of this mission */
  ask("Enter new mission location",Nloc),nl,
  mission(M,_,W,X,Y,Z),
  replace(mission(M,_,W,X,Y,Z),mission(M,Nloc,_,_,_)),
  continue(M).

process2(M,time) :- /* Change time of this mission */
  ask("Enter new mission start time",Time),nl,
  mission(M,_,W,X,_,Y,Z),
  replace(mission(M,_,W,X,_,Y,Z),mission(M,W,X,_,Y,Z)),
  continue(M).

process2(M,cmp) :- /* Change desired completion % for this mission */
  mission(M,_,W,X,Y,Z,_,_),
  not( on24HrAlert( M, _ ) ),
  lisp(CP,compperc),
  return_weapons(M),!
  miscount(J), K is J - 1,
  replace( miscount(J), miscount(K) ),
  retract(mission(M,_,W,X,Y,Z,_,_)),
  resetbk,
  type_inter(M,X,Inter),
  assignPlanes(M,X,Inter,CP,NP),!
  round( NP, NPlanes ),!
  implement_mission( M, W, Y, NPlanes, X, Inter, CP ),
  continue(M).

process2(M,cmp) :-
  on24HrAlert(M),
  write(‘Completion percentage for 24 Hour Alert can only be 100%. ‘).

process2(M,cmp) :- write("Error in changing mission",M),nl.

write("Error in changing mission",M),nl.

4:57 Apr 19 1990
type_inter(M, interdiction, av) :- aVehicles(M,_,_,_,_,_).

removeTypeAttr(M,interdiction) :- personnel(M,_,_,_),interKill(M,pc).
removeTypeAttr(M,interdiction) :- uVehicles(M,_,_,_,_,_),interKill(M,uv).
removeTypeAttr(M,interdiction) :- aVehicles(M,_,_,_,_,_),interKill(M,av).
removeTypeAttr(M,interdiction) :- buildings(M,_,_),interKill(M,bc).
removeTypeAttr(M,interdiction) :- cuts(M,_,_),interKill(M,rr).
removeTypeAttr(M,interdiction) :- bridge(M,_,_),interKill(M,bridge).

write('error finding mission type : ',X),nl.

interKill(M,pc) :- retract(personnel(M,_,_,_,_,_)).
interKill(M,uv) :- retract(uVehicles(M,_,_,_,_,_,_)).
interKill(M,av) :- retract(aVehicles(M,_,_,_,_,_,_)).
interKill(M,bc) :- retract(buildings(M,_,_,_,_,_,_,_)).
interKill(M,rr) :- retract(cuts(M,_,_,_,_,_,_)).
interKill(M,bridge) :- retract(bridge(M,_,_,_,_,_,_,_)).
removeTypeAttr(M,AreaPrep) :- retract(areaPrep(M,_,_,_,_,_,_,_,_)).
removeTypeAttr(M,closeAirSupport) :- retract(closeAirSupport(M,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,.,. 110
write('500 HD:'), write(Amt500HD), nl.

weap3 :-
    weapon(ld500, Amt500LD),
    write('500 LD:'), write(Amt500LD), nl.

weap4 :-
    weapon(m750, Amt750),
    write('750:'), write(Amt750), nl.

weap5 :-
    weapon(m2000, Amt2000),
    write('2000:'), write(Amt2000), nl.

weap6 :-
    weapon(napalm, AmtNapalm),
    write('Napalm (200 gal):'), write(AmtNapalm), nl.

weap7 :-
    weapon(atg,Amt),
    write('Air to ground:'), write(Amt), nl.

process3(3) :-
    miscount(Mcount),
    Mcount > 0,
    write('There are '), write(Mcount),
    write(' planned missions:'), nl, listmissions,
    write('Which mission number? '),read(M),
    showMission(M),
    process3(3,0).

process3(3) :-
    miscount(Mcount),
    Mcount = 0,
    write('No planned missions at this time...'), nl.

process3(3) :- write('Supply an existant mission next time').

showMission(M) :-
    mission(M, Loc, Type, ST, NP, Comp),
    write('Mission type: '), write(Type), nl,
    write('Mission location: '), write(Loc), nl,
    write('Mission start time: '), write(ST), nl,
    write('Number of Planes Assigned: '), write(NP), nl, nl, nl,
    suggest(M),
    level(complete,Comp,X),
    write('Completion level '), write(X), nl, nl.

showMission(M) :- write('No record of mission'), M, fail.

listmissions :-

mission(X,,Y,,Y),
print_out(X,Y),
fail.

listmissions :- nl.

print_out(X,Y) :- !,
write(X,' ',Y),nl.

process3(4). /* quit status check */

process(4,0) :- /* make changes to databases */
write('Change database'), nl,
write('Enter password: '),
read(Password),
password(Password),
lisp(Selection,menu4),
process4(Selection).

process(4,0) :- write('Invalid password').

password(secret).

changePassword :-
ask('Enter old password',OP),nl,
password(OP),
ask('Enter NEW password',NP),nl,
replace(password(OP),password(NP)).

changePassword :-
write('Password not changed... error on entry').

process4(0) :-
changePassword.

process4(1) :- /* Change plane amounts */
lisp(Pick,menu41),
process41(Pick).

process41(ua) :-
uaplanes(X),
write('There were ',X),write(' planes unassigned.'),nl,
ask('Input the number of planes that will now be unassigned: ',N),
replace(uaplanes(_,),uaplanes(N)).

process41(sm) :-
smplanes(X),
write('There were ',X),write(' planes on scheduled maintenance.'),nl,
ask('Input the number of planes that will now be on scheduled maintenance: ',N),
replace(smplanes(_,),smplanes(N)).

process41(bd) :-
bdplanes(X),
write('There were ',X),write(' planes with battle damage.'),nl,
ask('Input the number of planes that will now have battle damage: ',N),
replace(bdplanes(_,bdplanes(N)).

process41(nf) :-
mplanes(X),
write('There were ',X),write(' planes malfunctioning.'),nl,
ask('Input the number of planes that will now be malfunctioning: ',N),
replace(mplanes(_,mplanes(N)).

process41(none) :-
lisp(Selection,menu4),
process4(Selection).

process4(2) :-
lisp(Sel,loadtype),
process42(Sel).

addchange(Type,add) :-
weapon(Type,X),
ask('Input number of units to add: ',Amt),nl,
Y is X + Amt,
replace(weapon(Type,X),weapon(Type,Y)).

addchange(Type,change) :-
ask('Input number of units: ',Amt),nl,
replace(weapon(Type,X),weapon(Type,Amt)).

addchange(Type,neither) :- write('No change made.'),nl.

process42(mm20) :- /* change weapon amounts */
weapon(mm20,X),
write('There are ',X),write(' rounds of 20mm.'),
lisp(Choice,addorchange),
addchange(mm20,Choice).

process42(cb) :-
weapon(cb,X),
write('There are ',X),write(' Cluster-Bombs.'),
lisp(Choice,addorchange),
addchange(cb,Choice).

process42(hd500) :-
weapon(hd500,X),
write('There are ',X),write(' 500 HD'),
lisp(Choice,addorchange),
addchange(hd500,Choice).

process42(ld500) :-
weapon(ld500,X),
write('There are ',X),write(' 500 LD'),
lisp(Choice,addorchange),
addchange(ld500,Choice).
Format of file:

Plane Assignment Rules

\[
\text{assn}(\text{Mission_type}, \text{MenuChoice}, \text{PlanesNeeded}).
\]
\[
\text{assn}(\text{not .60})(\text{Mission_type}, \text{MenuChoice}, \text{PlanesNeeded}) :-
\]
\[
\text{assn}(\text{.60})(\text{Mission_type}, \text{MenuChoice}, \text{Needed}),
\quad \text{PlanesNeeded is Needed} \times \text{Factor}.
\]

where

- Factor is 3.0 for 90+%  
- Factor is 1.5 for 75+%  
- Factor is 0.8 for 45+%  
- Factor is 0.4 for 20+%  

\[
\text{assn}(\text{.20})(\text{on24HrAlert}, \_ , \_).
\]
\[
\text{assn}(\text{.45})(\text{on24HrAlert}, \_ , \_).
\]
\[
\text{assn}(\text{.60})(\text{on24HrAlert}, \_ , \_).
\]
\[
\text{assn}(\text{.75})(\text{on24HrAlert}, \_ , \_).
\]
\[
\text{assn}(\text{.90})(\text{on24HrAlert}, \_ , \_).
\]

\[
\text{assn}(\text{.20})(\text{areaprep,M,NP}) :- \text{assn}(\text{.60})(\text{areaprep,M,N}), \text{NP is N} \times 0.4.
\]
\[
\text{assn}(\text{.45})(\text{areaprep,M,NP}) :- \text{assn}(\text{.60})(\text{areaprep,M,N}), \text{NP is N} \times 0.8.
\]
\[
\text{assn}(\text{.60})(\text{areaprep, 1, 4.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{areaprep, 2,10.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{areaprep, 3,18.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{areaprep, 4,30.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{areaprep, 5,50.0}) :- !.
\]
\[
\text{assn}(\text{.75})(\text{areaprep,M,NP}) :- \text{assn}(\text{.60})(\text{areaprep,M,N}), \text{NP is N} \times 1.5.
\]
\[
\text{assn}(\text{.90})(\text{areaprep,M,NP}) :- \text{assn}(\text{.60})(\text{areaprep,M,N}), \text{NP is N} \times 3.0.
\]

\[
\text{assn}(\text{.20})(\text{pc,M,NP}) :- \text{assn}(\text{.60})(\text{pc,M,N}), \text{NP is N} \times 0.4.
\]
\[
\text{assn}(\text{.45})(\text{pc,M,NP}) :- \text{assn}(\text{.60})(\text{pc,M,N}), \text{NP is N} \times 0.8.
\]
\[
\text{assn}(\text{.60})(\text{pc,squad,1.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{pc,platoon,1.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{pc,company,3.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{pc,battalion,8.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{pc,regiment,25.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{pc,other,70.0}) :- !.
\]
\[
\text{assn}(\text{.75})(\text{pc,M,NP}) :- \text{assn}(\text{.60})(\text{pc,M,N}), \text{NP is N} \times 1.5.
\]
\[
\text{assn}(\text{.90})(\text{pc,M,NP}) :- \text{assn}(\text{.60})(\text{pc,M,N}), \text{NP is N} \times 3.0.
\]

\[
\text{assn}(\text{.20})(\text{closeAirSupport,M,NP}) :- \text{assn}(\text{.60})(\text{closeAirSupport,M,N}), \text{NP is N} \times 0.4.
\]
\[
\text{assn}(\text{.45})(\text{closeAirSupport,M,NP}) :- \text{assn}(\text{.60})(\text{closeAirSupport,M,N}), \text{NP is N} \times 0.8.
\]
\[
\text{assn}(\text{.60})(\text{closeAirSupport,squad,1.0}) :- !.
\]
\[
\text{assn}(\text{.60})(\text{closeAirSupport,platoon,1.0}) :- !.
\]
(\texttt{users/masl/gwright/lonnie/newrules.mprolog})

\begin{verbatim}
assn{.60}(closeAirSupport,company,3.0):- !.
assn{.60}(closeAirSupport,battalion,8.0):- !.
assn{.60}(closeAirSupport,regiment,25.0):- !.
assn{.60}(closeAirSupport,other,70.0):- !.

assn{.75}(closeAirSupport,M,\textit{NP}):- assn{.60}(closeAirSupport,M,N), \textit{NP} is \textit{N} * 1.5.
assn{.90}(closeAirSupport,M,\textit{NP}):- assn{.60}(closeAirSupport,M,N), \textit{NP} is \textit{N} * 3.0.

assn{.20}(uv,M,\textit{NP}) :- assn{.60}(uv,M,N), \textit{NP} is \textit{N} * 0.4.
assn{.45}(uv,M,\textit{NP}) :- assn{.60}(uv,M,N), \textit{NP} is \textit{N} * 0.8.

assn{.60}(uv,5,24.0):- !.
assn{.60}(uv,4,8.0):- !.
assn{.60}(uv,3,5.0):- !.
assn{.60}(uv,2,4.0):- !.
assn{.60}(uv,1,2.0):- !.

assn{.75}(uv,M,\textit{NP}):- assn{.60}(uv,M,N), \textit{NP} is \textit{N} * 1.5.
assn{.90}(uv,M,\textit{NP}):- assn{.60}(uv,M,N), \textit{NP} is \textit{N} * 3.0.

assn{.20}(av,M,\textit{NP}) :- assn{.60}(av,M,N), \textit{NP} is \textit{N} * 0.4.
assn{.45}(av,M,\textit{NP}) :- assn{.60}(av,M,N), \textit{NP} is \textit{N} * 0.8.

assn{.60}(av,0,0.0):- !.
assn{.60}(av,1,3.0):- !.
assn{.60}(av,2,6.0):- !.
assn{.60}(av,3,12.0):- !.
assn{.60}(av,4,20.0):- !.

assn{.75}(av,M,\textit{NP}):- assn{.60}(av,M,N), \textit{NP} is \textit{N} * 1.5.
assn{.90}(av,M,\textit{NP}):- assn{.60}(av,M,N), \textit{NP} is \textit{N} * 3.0.

assn{.20}(bc,M,\textit{NP}) :- assn{.60}(bc,M,N), \textit{NP} is \textit{N} * 0.4.
assn{.45}(bc,M,\textit{NP}) :- assn{.60}(bc,M,N), \textit{NP} is \textit{N} * 0.8.

assn{.60}(bc,1,2.0):- !.
assn{.60}(bc,2,4.0):- !.
assn{.60}(bc,3,8.0):- !.
assn{.60}(bc,4,16.0):- !.
assn{.60}(bc,5,24.0):- !.
assn{.60}(bc,6,30.0):- !.

assn{.75}(bc,M,\textit{NP}):- assn{.60}(bc,M,N), \textit{NP} is \textit{N} * 1.5.
assn{.90}(bc,M,\textit{NP}):- assn{.60}(bc,M,N), \textit{NP} is \textit{N} * 3.0.

assn{.20}(rr,M,\textit{NP}) :- assn{.60}(rr,M,N), \textit{NP} is \textit{N} * 0.4.
assn{.45}(rr,M,\textit{NP}) :- assn{.60}(rr,M,N), \textit{NP} is \textit{N} * 0.8.

assn{.60}(rr,1,2.0):- !.
assn{.60}(rr,2,2.0):- !.
\end{verbatim}
assigning additional aircraft for various factors:

/* Anti-aircraft guns */
extra_assn(anti_air,1,0).
extra_assn(anti_air,2,1).
extra_assn(anti_air,3,2).
extra_assn(anti_air,4,4).
extra_assn(anti_air,5,7).

/* Surface to Air missile sites */
extra_assn(samsites,1,0).
extra_assn(samsites,2,1.5).
extra_assn(samsites,3,3).
extra_assn(samsites,4,5).
extra_assn(samsites,5,10).

/* Low altitude cover */
extra_assn(cover,open,1).

/* Terrain type */
extra_assn(terrain,open,1).
extra_assn(terrain,hilly,1.3).
extra_assn(terrain,mountains,2).

/* Protective cover type */
extra_assn(cover,open,1).
extra_assn(cover, sb, 1.25).
extra_assn(cover, jungle, 1.5).
extra_assn(cover, urban, 1.8).

extra_assn(avehics, 0, 0). /* number of armoured vehicles */
extra_assn(avehics, 1, 3).
extra_assn(avehics, 2, 6).
extra_assn(avehics, 3, 14).
extra_assn(avehics, 4, 20).

assign_avs(AV, Nplanes) :- !, extra_assn(avehics, AV, Nplanes).

assign_std_factors(AA, SS, Nplanes) :- /* assign values for these typical mission factors */
    !,
    extra_assn(anti_air, AA, A),
    extra_assn(samsites, SS, S),
    Nplanes is A + S.

compute_terr_factors(Terr, Cov, LC, Factor) :- /* Multiplying factors to number of planes */
    !,
    extra_assn(terrain, Terr, T),
    extra_assn(cover, Cov, C),
    extra_assn(locover, LC, L),
    X is T * C,
    Factor is X * L.

/* Load assignments:
loads will be made into a database, since lists are so difficult to implement. The tuples added to the database will follow this general rule:

load_missiontype( Mnum, %possible factors ) :-
    assert( loadpc( Mnum, Load, Fusing, %x ) ),
    ...
    assert( loadpc( Mnum, Load, Fusing, %x ) ).

where Mnum is a mission number, possible factors include materials, roadtypes, defensive positions, cover etc...

/* cover_loads( Mnum, Perc, CoverLevel ) */

cover_loads( M, P, open ) :-
    X is P/2,
    assert( loadpc( M, napalm, impact, X ) ),
    assert( loadpc( M, cb, impact, X ) ).

cover_loads( M, P, sb ) :-
    Y is P/3,
assert( loadpc( M, napalm, impact, Y ) ),
assert( loadpc( M, cb, impact, Y ) ),
assert( loadpc( M, hd500, impact, Y ) ).

cover_loads( M, P, jungle ) :-
Y is P/3,
assert( loadpc( M, hd500, impact, Y ) ),
assert( loadpc( M, m750, impact, Y ) ),
assert( loadpc( M, m2000, impact, Y ) ).

cover_loads( M, P, urban ) :-
X is P/2,
assert( loadpc( M, m2000, impact, X ) ),
assert( loadpc( M, atg, impact, X ) ).

/* defpos_loads( Mnum, Perc, DLevel ) */
defpos_loads( M, P, none ).
defpos_loads( M, P, trenches ) :-
X is P/2,
assert( loadpc( M, hd500, delayed, X ) ),
assert( loadpc( M, m750, delayed, X ) ).
defpos_loads( M, P, tunnels ) :-
assert( loadpc( M, m2000, delayed, P ) ).
defpos_loads( M, P, rcb ) :-
X is P/2,
assert( loadpc( M, m2000, delayed, X ) ),
assert( loadpc( M, m750, delayed, X ) ).

load_24hr( M ) :-
assert( loadpc( M, napalm, impact, 0.40 ) ),
assert( loadpc( M, hd500, impact, 0.40 ) ),
assert( loadpc( M, atg, proximity, 0.20 ) ).

load_closeAir( M, P ) :-
A is .45*P, B is .10*P,
assert( loadpc( M, napalm, impact, A ) ),
assert( loadpc( M, hd500, impact, A ) ),
assert( loadpc( M, cb, proximity, B ) ).

load_ap( M, P ) :-
Y is P/3,
assert( loadpc( M, cb, proximity, Y ) ),
assert( loadpc( M, m750, delayed, Y ) ),
assert( loadpc( M, m2000, delayed, Y ) ).

/* interdiction loads (std) */
load_pers( M, P ) :-
Z is P/4,
assert( loadpc( M, hd500, impact, Z ) ),
assert( loadpc( M, m750, impact, Z ) ),
assert( loadpc( M, m2000, impact, Z ) ),
assert( loadpc( M, cb, proximity, Z ) ).

load_uv( M, P ) :-
    X is P/2,
    assert( loadpc( M, atg, impact, X ) ),
    assert( loadpc( M, napalm, impact, X ) ).

load_av( M, P ) :-
    X is P/2,
    assert( loadpc( M, atg, impact, X ) ),
    assert( loadpc( M, ld500, impact, X ) ).

load_cuts( M, P, 1 ) :-
    X is P/2,
    assert( loadpc( M, m750, delayed, X ) ),
    assert( loadpc( M, m2000, delayed, X ) ).

load_cuts( M, P, 2 ) :-
    X is P/2,
    assert( loadpc( M, m750, impact, X ) ),
    assert( loadpc( M, atg, impact, X ) ).

load_cuts( M, P, 3 ) :-
    X is P/2,
    assert( loadpc( M, m750, impact, X ) ),
    assert( loadpc( M, atg, impact, X ) ).

load_buildmats( M, P, wood ) :-
    assert( loadpc( M, napalm, impact, P ) ).

load_buildmats( M, P, rhut ) :-
    X is P/2,
    assert( loadpc( M, hd500, proximity, X ) ),
    assert( loadpc( M, m750, proximity, X ) ).

load_buildmats( M, P, brick ) :-
    X is P/2,
    assert( loadpc( M, m750, proximity, X ) ),
    assert( loadpc( M, m2000, proximity, X ) ).

load_buildmats( M, P, re ) :-
    X is P/2,
    assert( loadpc( M, m750, delayed, X ) ),
    assert( loadpc( M, m2000, delayed, X ) ).

load_bridgemats( M, P, wood ) :-
    Y is P/3,
    assert( loadpc( M, hd500, impact, Y ) ),
    assert( loadpc( M, m750, impact, Y ) ),
    assert( loadpc( M, atg, impact, Y ) ).

load_bridgemats( M, P, concrete ) :-
    X is P/2, Z is P/4,
    assert( loadpc( M, m750, delay, Z ) ),
assert( loadpc( M, m2000, delay, Z ) ),
assert( loadpc( M, atg, impact, X ) ).

load_bridgemats( M, P, steel ) :-
  X is P/2,  Z is P/4,
  assert( loadpc( M, m750, delay, Z ) ),
  assert( loadpc( M, m2000, delay, Z ) ),
  assert( loadpc( M, atg, impact, X ) ).