

Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering
Research

Computer Science and Engineering

Report Number: WUCS-88-22

1988-08-01

BEMAS: User's Manual, 2nd Edition

Rosanne M. Fulcomer, J. Andrew Fingerhut, and William E. Ball

This paper is a user's manual for BEMAS, a BELIEF MAINTENANCE System. BEMAS is a menu driven system which provides an easy to use interface between a user and a knowledge base. Given a set of data, and a set of rules, BEMAS will help the user to identify an object by analyzing the properties of that object. Data can be added and deleted at any time, either directly or by deleting beliefs on which the data is dependent. BEMAS maintains the relations and dependencies between data using a dynamic dependency net. BEMAS also has the capability... [Read complete abstract on page 2.](#)

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation

Fulcomer, Rosanne M.; Fingerhut, J. Andrew; and Ball, William E., "BEMAS: User's Manual, 2nd Edition" Report Number: WUCS-88-22 (1988). *All Computer Science and Engineering Research*. https://openscholarship.wustl.edu/cse_research/779

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

BEMAS: User's Manual, 2nd Edition

Rosanne M. Fulcomer, J. Andrew Fingerhut, and William E. Ball

Complete Abstract:

This paper is a user's manual for BEMAS, a BELIEF MAINTENANCE System. BEMAS is a menu driven system which provides an easy to use interface between a user and a knowledge base. Given a set of data, and a set of rules, BEMAS will help the user to identify an object by analyzing the properties of that object. Data can be added and deleted at any time, either directly or by deleting beliefs on which the data is dependent. BEMAS maintains the relations and dependencies between data using a dynamic dependency net. BEMAS also has the capability to make inferences using incomplete information while still maintaining knowledge base integrity.

**BEMAS:
USER'S MANUAL, 2ND EDITION**

**Rosanne M. Fulcomer, J. Andrew Fingerhut
and William E. Ball**

WUCS-88-22

**Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899**

This work was supported by McDonnell Aircraft Company under contract Z71021.

BEMAS: User's Manual, 2nd Edition

Rosanne M. Fulcomer
J. Andrew Fingerhut
William E. Ball

Department of Computer Science
Washington University
St. Louis, MO 63130

This paper is a user's manual for BEMAS, a **BE**lief **MA**intenance **S**ystem. BEMAS is a menu driven system which provides an easy to use interface between a user and a knowledge base. Given a set of data, and a set of rules, BEMAS will help the user to identify an object by analyzing the properties of that object. Data can be added and deleted at any time, either directly or by deleting beliefs on which the data is dependent. BEMAS maintains the relations and dependencies between data using a dynamic dependency net. BEMAS also has the capability to make inferences using incomplete information while still maintaining knowledge base integrity.

Acknowledgements: This work was supported by McDonnell Aircraft Company under contract Z71021.

TABLE OF CONTENTS

1. Introduction	1
2. Files In BEMAS	2
2.1. Setting Up The Initial Input Files	2
2.1.1. The BELIEF File	2
2.1.2. The RULE File	3
2.2. Contents Of The Dependency Net File	5
2.2.1. Counter values	5
2.2.2. The Knowledge Base	5
2.2.3. Dynamic Dependency Net	7
2.2.3.1. Beliefs	7
2.2.3.2. Justifications	8
3. Working In The Macintosh/Common Lisp Environment	9
3.1. How To Load BEMAS	10
3.2. The Startup Options Menu	10
3.3. The BEMAS Main Menu	11
4. Options in BEMAS	13
4.1. Adding A Premise	13
4.1.1. The Belief Component Menu	13
4.2. Deleting A Premise	16
4.3. Querying	16
4.4. Display Belief	19
4.4.1. The Graphical Display	20
4.5. Showing The Justifications In BEMAS Database	21
4.6. Showing The Rules In BEMAS Database	22
4.7. Saving the Dependency Net	22
4.8. Restarting BEMAS	22
4.9. Leaving BEMAS	23
5. Error Handling	24



BEMAS: User's Manual, 2nd Edition

Rosanne M. Fulcomer
J. Andrew Fingerhut
William E. Ball

WUCS-88-22

1. Introduction

BEMAS, which stands for **BELief MAintenance System**, is an Artificial Intelligence prototype using Allegro Common Lisp on a Macintosh II.

The objective of the Belief Maintenance System is to deal with incomplete or uncertain information. It should also be able to automatically draw inferences and conclusions from a set of data using rules in its knowledge base [1]. It can also be queried for dependencies among beliefs in its database. Currently, BEMAS will answer yes, no, or unknown to the query for the truth of a belief.

Given a set of data, and a set of rules, BEMAS will help the user to identify an object by analyzing the properties of that object. Data can be added and deleted at any time, either directly or by deleting beliefs on which the data is dependent. Rules must be given at the time of initialization, thus they cannot be added nor deleted interactively.

The first edition of this manual corresponds to the coding of BEMAS in C-Prolog. There were no interactive graphics in the previous system. References to this system can be found in [2].

The notational convention will be as follows, throughout the paper. Examples of what BEMAS displays will be in **bold font**, and examples of what the user types into the computer will be in *italic font*, if the entry is to be done outside a BEMAS window.

2. Files In BEMAS

This section will discuss what files are needed while working with BEMAS. Also in this section are some of the concepts behind the development of BEMAS.

At least one of the following files are needed for BEMAS to be used:

1. RULE file
2. Dependency Net file

The RULE file contains all the rules that represent the dependencies and/or constraints that govern the beliefs in the system. Their format will be discussed in detail in the Section 2.1.2. This file may be used with another file called the BELIEF file. The BELIEF file contains all initial premise beliefs to be input into the system. The format of these beliefs will be discussed in Section 2.1.1. The RULE file may be used alone also, in which case all external beliefs must be entered using the BEMAS menus (Section 3.3). When the user desires to change any rules, the rules must be changed in the RULE file. Then this changed file must be used again as the input RULE file.

The Dependency Net file is considered to be both an input and an output file. A dependency net is what BEMAS uses to store beliefs and reasons for the existence of those beliefs. Also in the Dependency Net file, will be the original rules used to create the dependency net structure (Section 2.2), since these rules are always needed as input to BEMAS. When BEMAS uses a RULE file and the user asks BEMAS to perform computations using a certain set of beliefs, they have the option to save those computations to a Dependency Net file. Once this file is saved as output, it can be used again, so that given the same rules and beliefs, the computations that were previously performed will still exist. The user does not have to instruct BEMAS to perform any of the same computations. The format of this file will be discussed in Section 2.2.

2.1. Setting Up The Initial Input Files

The very first time BEMAS is used with some set of rules, the RULE file must be used as the input file. We will call this file and the BELIEF file, which may or may not be used with the RULE file, the initial input files. The following section will discuss the contents and usage of these initial input files. Instructions on formatting and loading the files will be provided.

2.1.1. The BELIEF File

A belief is the most basic structure used by BEMAS. There are four types of beliefs, but only the premise belief is entered into the BELIEF file. The other types of beliefs will be discussed in Section 2.2.3.1. The premise belief is the most important belief. This type of belief, along with the rules, is what allows BEMAS to draw certain conclusions about the particular world representation it is given to work with. These beliefs are considered to be known, or factual, at the time they are entered into the system, and for as long as they remain in the system. Thus, when a BELIEF file is used, the user is simply entering those facts that are known at the time BEMAS is started. A premise in the input data file is considered a particular kind of belief whose label depends on the fact that it was explicitly entered into the system from an external source. The premise belief should be in the following format:

BEMAS: A Belief Maintenance System Prototype

(property (object) truth_value)

The truth_value slot can contain either a value of "true" or "false". When it contains "true", the user believes that the object has the given property. When the truth_value contains "false", the user believes that the object does not have the given property. For instance:

(can_fly (cardinal) true)

indicates that the user believes that the cardinal can fly. Notice that the object is the cardinal, the property is can_fly (using BEMAS notation), and the truth_value is true. It is important that the object is surrounded by one set of parentheses and that the entire belief is surrounded by another set of parentheses. On notation, words to describe objects and properties may be joined with a dash or an underscore, as seen the above example using can_fly.

2.1.2. The RULE File

A rule in BEMAS is an IF-THEN statement as follows:

IF A1, A2, ... An THEN C.

where A1, A2, ... An are called "antecedents" of the rule and C is called the "consequent" of the rule.

In general, given the situation in which all the antecedents of the rule are in the database, the rule will be satisfied and the consequent will be asserted into the database. Specifically when a consequent is asserted will be discussed in sections 4.1 and 4.3. In the input file, the rules should be in the following format:

((A1 A2 ... An) C)

which is interpreted as "A1 & A2 & ... & An ==> C" or "if A1 and A2 and ... and An, then C." Again the antecedents of the rule are A1, A2, ..., An and the consequent is C. Consequents and antecedents should be in the following format:

(property (?object) truth_value)

This format is similar to the format of premise, except that the first letter of the object is a question mark, representing a variable, instead of an object without a preceding question mark which represents a constant. Multiple antecedents in this format are separated by a space with a set of antecedents surrounded by parentheses. An example of this follows.

(((prop1 (?obj) t1) (prop2 (?obj) t2)) (prop3 (?obj) t3))

which says, "if truth value, t1, holds for some object, ?obj, having property, prop1, and truth value, t2, holds for the same object, ?obj, having property, prop2, then the truth value, t3, holds

BEMAS: A Belief Maintenance System Prototype

for that object, ?obj, to have the property, prop3." Notice that the variables pertaining to the same object within a rule should be represented by the same variable.

BEMAS expects the rules it is given to be consistent with each other. Thus it is the user's responsibility to check for this initial consistency. One meaning of consistency is that two rules should not come to opposite conclusions with the same antecedents, as shown in the following example, using the animal recognition:

R1: (((has_feathers (?Y) true) (flies (?Y) true)) (is_bird (?Y) true))
R2: (((has_feathers (?Y) true) (flies (?Y) true)) (is_bird (?Y) false))

If these rules or similar ones are present in the knowledge base, then the knowledge base would be considered inconsistent. There are other ways that a knowledge base may become inconsistent, e.g., circular rules, where the truth of a belief inadvertently depends on the falseness of itself or some variation of this occurrence, and incomplete rules where more rules must be added to make the database complete so that all conclusions may be reached. The following example demonstrates that situations of circularity and incompleteness may go unrecognized until the rules are thoroughly tested. An example of this is:

*R3: (((lays_eggs (?X) true) (flies (?X) true))
 (is_bird (?X) true))*
*R4: (((is_bird (?X) true) (flies (?X) false)
 (swims (?X) true) (is_black_and_white (?X) true))
 (penguin (?X) true))*

It is known that a penguin is a type of bird though it does not fly. It must be explicitly asserted that a pet, Ludwig, is a bird, i.e. (is_bird (ludwig) true), or that Ludwig is in fact a penguin, i.e. (penguin (ludwig) true) to show that Ludwig is a penguin. Otherwise, given the rules above we could never show that Ludwig is a penguin, since the penguin rule depends on (flies (ludwig) false), and (flies (ludwig) true) is needed for (is_bird (ludwig) true) which is also needed in the penguin rule. One can see the type of circularity occurring. This situation can be resolved by recognizing that the rules are incomplete, thus asserting another rule to support is_bird that can be satisfied when the bird is a penguin, such as:

R5: (((has_feathers (?X) true)) (is_bird (?X) true))

or the antecedent (is_bird (?X) true) could be deleted from the penguin rule and possibly be replaced by (has_feathers (?X) true). A potential problem with adding the new is_bird rule above is that it consists of a single antecedent. With multiple antecedents, if at least one has a well-founded support, then any nonexistent antecedents have a chance of being assumptions and if the rule is satisfied, the consequent can be entered into the database. If there is a single antecedent in a rule and that antecedent is not present in the database, then BEMAS will not ask for that antecedent to be assumed, since there is no well-founded support for the consequent. Thus, the rule has less of a chance of being satisfied than a rule with multiple antecedents. This will be discussed further in sections 4.1 and 4.3. Our answer to the problem would be to make both corrections, i.e. add the new is_bird rule and add the characteristic of (has_feathers (?X) true) to the penguin rule.

Once a rule is satisfied, a justification is created for the consequent as it is instantiated. This justification is entered into the database and remains there until it becomes invalid by the

BEMAS: A Belief Maintenance System Prototype

removal of one or more of the antecedents on which the consequent depended. Justifications will be mentioned throughout the manual, but will be discussed in detail in section 2.2.3.2.

2.2. Contents Of The Dependency Net File

The user may save the computations that BEMAS performs to an output file called the Dependency Net. This file can then be used as input once it is created. The user cannot create the Dependency Net file directly, but must have already used BEMAS with at least a RULE file, before the Dependency Net file can be created. The user may save to a Dependency Net file at any time during interaction with BEMAS. Also, BEMAS will ask if it should save its computations to a Dependency Net file when the user quits. The next section will discuss the contents of the Dependency Net file in detail.

BEMAS creates a Dependency Net file from the information it has been given by the user and from the computation it has performed. This file is divided into the following four sections:

1. Counter Values:
which maintain the current number of rules, beliefs, and justifications.
2. Knowledge Base:
which includes the rules given to BEMAS from the initial input file.
3. Dynamic Dependency Net
which holds all the beliefs and justifications in the current representation.

2.2.1. Counter values

These counters contain the last unique identification number given to a member of each of the sets of rules, justifications, and beliefs. An example of this first section of output:

```
6 ; Belief counter
3 ; Rule counter
2 ; Justification counter
```

The example tells the user that the next belief that is entered will receive an identification number of 7 and the next justification that is computed will receive an identification number of 3. But since rules cannot be entered interactively with BEMAS, the rule counter gives an indication of how many rules are being used by BEMAS. For this example, BEMAS is using 3 rules. We cannot use the belief or justification counter to give us this indication, since beliefs and justifications can be deleted from the system, and this has no effect on the respective counter.

2.2.2. The Knowledge Base

BEMAS creates a knowledge base from rules it is given when the RULE file is initially loaded. This knowledge base is what BEMAS uses to make inferences. The rules are taken from the input file and transformed into the data structures BEMAS uses which make up the knowledge base. Once BEMAS creates the knowledge base, the rules that comprise it cannot be

BEMAS: A Belief Maintenance System Prototype

changed interactively, nor can they be changed directly within the output file. If the user desires to change the rules, it must be done in the initial input file and BEMAS must be given these rules in their initial form, from the RULE file, so that BEMAS can create a new knowledge base. Once again, it is the user's responsibility to make sure that the rules initially given to BEMAS are consistent.

The knowledge base BEMAS creates is a collection of records, one for each rule, each one being distinguished by its unique identification number.

For internal use, a rule record has the following fields:

1. Unique Identification Number
2. List of Uninstantiated Antecedents
3. Uninstantiated Consequent
4. List of Justifications currently using the rule

where the list of justifications contains the unique identification numbers of the justifications used, as discussed in Section 2.2.3.2. When a rule uses variables, it can have more than one justification due to multiple instantiations of the variable to an object.

An example of a rule as BEMAS would display it interactively to the user is as follows:

```
(Rule 1
ant = ( HAS_HAIR (?X) TRUE )
cons = ( IS_MAMMAL (?X) TRUE )
justs = (<Just 1>))
```

This example rule says, that for all ?X, if ?X has hair (ant=), then ?X is a mammal (cons=). BEMAS will also list all justifications that use the rule by their unique identification number after (justs=).

What BEMAS prints to the Dependency Net output file is a less detailed description (as far as the user is concerned) of a rule than what is described interactively. The following is an example of the output of the knowledge base to the Dependency Net file:

```
(R2 ((GIVES_MILK (?X) TRUE) (FLIES (?X) FALSE)) (IS_MAMMAL (?X) TRUE) NIL)
(R1 ((HAS_HAIR (?X) TRUE)) (IS_MAMMAL (?X) TRUE) (J1) )
(R3 ((LAYS_EGGS (?X) TRUE) (FLIES (?X) TRUE)) (IS_BIRD (?X) TRUE) (J5) )
```

In this example, the three rules are given with their respective fields. For the first rule, R2 is the unique i.d., GIVES_MILK and FLIES are the properties of its antecedents, IS_MAMMAL is the property of its consequent, and NIL tells BEMAS that there are no justifications that are currently using this rule. In the second rule, R1 is the unique i.d., HAS_HAIR is the property of the only antecedent, IS_MAMMAL is the property of the consequent, and J1 is the identification number of a justification that is currently using the rule.

BEMAS: A Belief Maintenance System Prototype

2.2.3. Dynamic Dependency Net

A structure called a dynamic dependency net holds the representation of the current state of the world. What BEMAS prints to the output file are all beliefs and their levels, which will be briefly explained next, along with the justification for the beliefs which are neither premises nor assumptions, but have been deduced by BEMAS.

2.2.3.1. Beliefs

There are considered to be four levels of beliefs, which form a natural hierarchy. Each level of belief is stored as an array of records, one for each belief at that level. All beliefs are distinguished by a unique identification number, which is a positive integer. The fields in the belief will be discussed later. For now, the levels of beliefs are as follows:

premise:

This level contains those beliefs which have been explicitly entered by the user, either initially, as discussed before, or interactively within BEMAS. Beliefs that are members of this level are called premises. These beliefs need no justifications.

derived:

This level contains those beliefs that are deduced using only premises and other beliefs from the derived level, along with the rules in the knowledge base. These beliefs are called derived beliefs. They will only be asserted during forward chaining (section 4.1) and need justifications to exist.

assumed:

This level contains those beliefs that are assumed interactively within BEMAS in order to prove some other beliefs. These beliefs must be authorized by the user. The beliefs at the assumed level are called assumptions. When an assumption is permitted by the user, the assumption will be asserted into the database. After this assertion, there is no forward chaining. Only the belief that needed the assumption for a proof will be asserted. This will be discussed in section 4.3.

inferred:

This level contains those beliefs that are deduced using at least one premise along with derived beliefs, and one or more assumptions. These beliefs are called inferred beliefs. They will only be asserted during querying and backward chaining (section 4.3). Inferred beliefs need justifications to exist. The assertion of an inferred belief causes no forward chaining.

For use in BEMAS, a single belief is represented using a record structure with the following fields:

1. Unique Identification Number
2. Level at which belief resides - (Premise, Derived, Assumed, Inferred)
3. Description or Property
4. Instantiated Object - (object)
5. Truth Value
6. Applicable Rules where belief could be a consequent
7. Justifications with the belief as a consequent
8. Applicable Rules where belief could be used as an antecedent
9. Justifications that use the belief as an antecedent

BEMAS: A Belief Maintenance System Prototype

where the list of applicable rules contains the unique identification number of each rule, which is explained in section 2.2.2. The list of justifications will contain the unique identification number of each justification used for that belief, which will be explained in next section.

BEMAS will print out the contents of each field of the belief record to the output file. Examples of what such beliefs look like are:

```
(P4 HAS_FEATHERS (ROBIN) TRUE NIL NIL (R4 R3) (J3 J2) )
(P15 IS_MAMMAL (ELEPHANT) TRUE (R1 R2) NIL (R10 R11) NIL)
(D9 TIGER (JOE) FALSE (R16) (J4) NIL NIL)
```

The first belief in the example is a premise (P4) with an i.d. number of 4. This belief has the property HAS_FEATHERS, the object ROBIN and the truth value, TRUE. There are no rules for which the belief can be a consequent, and hence no justifications for which the belief is a consequent. R4 and R3 are the unique i.d.'s of the rules for which the belief can be an antecedent, and J3 and J2 are the unique i.d.'s of the justifications for which the belief is an antecedent.

The second belief in the example is also a premise, this one with an i.d. number of 15. It has the property IS_MAMMAL, the object ELEPHANT, and the truth value, TRUE. This belief can be used as a consequent of the rules R1 and R2. It can be used as an antecedent of the rules R10 and R11. There are no justifications for which it is used as either an antecedent or a consequent.

The third example belief is a derived belief (D9) with an i.d. of 9. This belief has the property TIGER, the object JOE, and the truth value FALSE. It can be a consequent of the rule R16, and is a consequent of the justification J4. It cannot be an antecedent of any rule in the system, and hence cannot be an antecedent of any justification.

BEMAS does not display so much information to the user directly. An example of a belief as BEMAS displays it to the user is as follows:

```
<I16 = IS_BIRD((PENGUIN),TRUE)>
```

This belief tells the user that "a penguin is a bird" is represented as an inferred belief in the system with unique identification 16. The rest of the fields of the belief are not directly visible to the user.

2.2.3.2. Justifications

A justification is created when a rule is satisfied for a particular belief. That belief exists as long as the justification is valid. Each justification is a record from a collection of all justifications, where each is distinguished by its unique identification number.

For use in BEMAS, the justification record has the following fields:

1. Unique Identification Number
2. List of instantiated antecedents
3. Justified consequent
4. Rule being justified

BEMAS: A Belief Maintenance System Prototype

BEMAS prints a condensed version of all the fields of a justification to the Dependency Net output file as follows:

```
(J5 (P5 D8) D10 R22)
(J2 (P4 P7) D8 R4)
```

In the above example, the first justification (J5) has a premise (P5) and a derived belief (D8) as its antecedents, and a derived belief (D10) as its consequent. R22 is the rule on which J5 is based. The second justification (J2) has premises P4 and P7 as its antecedents and a derived belief (D8) as its consequent. R4 is the rule on which J2 is based.

BEMAS actually displays more information about the justification to the user than what the user sees in the output file. An example of a justification as BEMAS displays it to the user is:

```
(Just 6
ant = ( <Bel 17 = FLIES((ALBATROSS),TRUE)>
        <Bel 18 = LAYS_EGGS((ALBATROSS),TRUE)> )
cons = <Bel 19 = IS_BIRD((ALBATROSS),TRUE)>
rule = <Rule 5> )
```

This display of the justification gives the antecedent (ant=), which is "an albatross flies and lays eggs", and the consequent (cons=), which is "since the antecedent holds, then an albatross is a bird." The last part is the rule (rule=) which is the basis for the justification.

3. Working In The Macintosh/Common Lisp Environment

We are assuming that the user of BEMAS has working knowledge of the Macintosh II, and will therefore only explain those details that pertain to the operation of BEMAS directly. The user just needs some familiarity with Lisp, but is not required to do any programming in the language.

When the user loads BEMAS, Common Lisp will be loaded automatically. When Lisp has been loaded the Lisp Listener window will be displayed at the bottom of the screen. Inside the Listener window will be the message:

```
Welcome to Allegro CL Version 1.2.1
?
```

where the ? on the second line is the Lisp prompt.

For best possible viewing of the menus displayed by BEMAS, it is best to shrink the width of the Listener window to about a quarter of its original size and move it to the bottom of the screen. This is done as it would be with any other Macintosh window.

BEMAS: A Belief Maintenance System Prototype

When BEMAS is exited, it will automatically quit the Lisp process.

3.1. How To Load BEMAS

To load BEMAS, first display the Common Lisp directory. There will be one BEMAS folder inside this directory where system files and user files will be stored. There will also be a file in the Lisp directory that is called BEMAS Startup. Double clicking on this file icon will load BEMAS, which will load Common Lisp. An example of the what the Lisp folder will look like is in Figure 1.

3.2. The Startup Options Menu

The first menu that appears will be one entitled **Welcome to BEMAS** (see Figure 2). This is called the "Startup Options" menu, because it will ask you to choose one of three ways to start BEMAS. Click one of the three lines. When you do so, the circle at the beginning of the line will darken to indicate which method you have selected. You can change your mind by clicking a different line. When you have made your final decision, click the button Start at the bottom. If you decide not to start BEMAS at all, click Cancel and you will be left in Lisp.

The first button is for loading a dependency net that BEMAS has already computed. This way should only be selected if you have run BEMAS before and saved a dependency net.

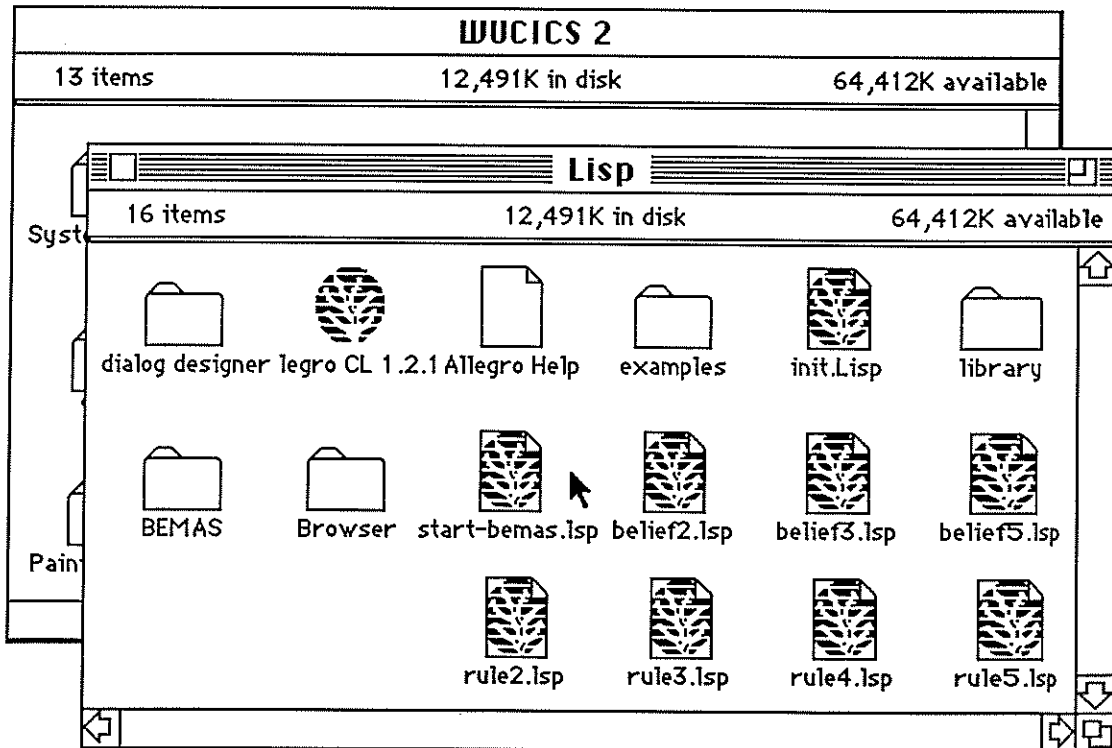


Figure 1: Loading BEMAS

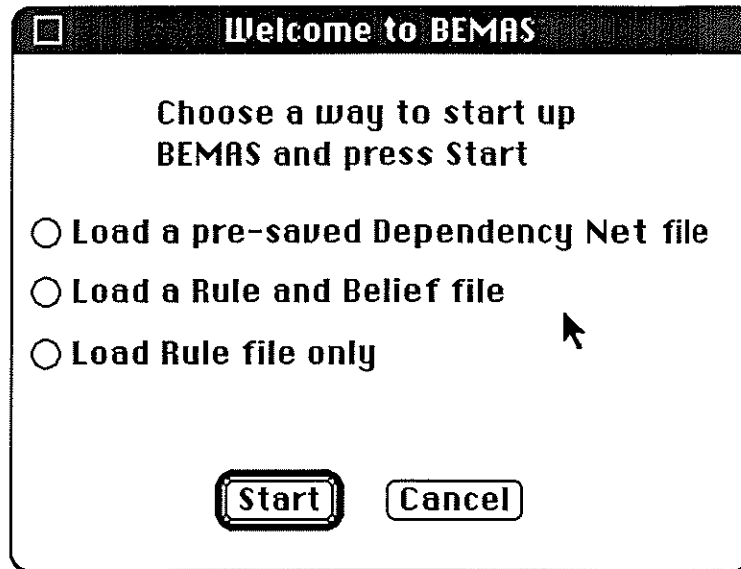


Figure 2: The Startup Options Menu

If the user wishes to load only a rule file and create each belief manually, one at a time, then the third button should be clicked. What will then appear (after also clicking Start) will be the standard Macintosh "load file" menu, from which the user can choose the rule file that was created as in Section 2.1.2. BEMAS will load in the rule file and give control back to the user. This will be discussed in Section 3.3.

If the user has also created a belief file in the manner of Section 2.1.1, then to load both the initial rule file and belief file, click the second button. The standard Macintosh menu will appear once for the user to choose the rule file, and when this is done, the menu will automatically appear a second a time for the user to pick the belief file. BEMAS will transform these premise belief and initial rules into the structures it needs. Also, BEMAS will use the premises that it has to satisfy or fire as many rules as possible, and assert the consequents of those rules. It will continue to fire rules using the premises and current derived beliefs until no more rules can be fired. This process is called "forward chaining", and will be executed by BEMAS every time a premise is added to the database.

To specify the rule file, simply click the name of the file, and then click the button Open (or a short-cut is to double-click the name of the file, see Figure 3). If the file you wish to load is not visible, then use the up and/or down arrow keys to make it so. It is also possible that the file you want is in another directory. To go up in the tree of directories, press and hold the mouse button on the name of the directory, which is in the top middle of the window. This will show a list of ancestors of this directory all the way back to the root. Move the mouse down while pressing the button until the directory you wish is highlighted, then release the mouse button. To go down in the tree of directories, open the desired directory as you would any other file.

3.3. The BEMAS Main Menu

After the user specifies the input file(s), the BEMAS main menu will appear (see Figure 4). This menu guides the user through interaction with BEMAS. The individual options will be discussed in detail in the next section.

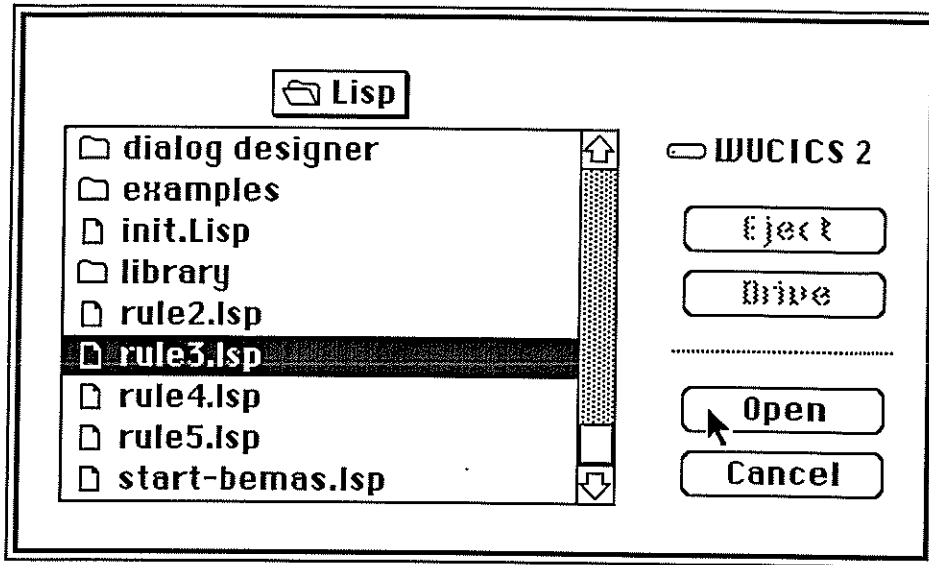


Figure 3: Selecting the Rule or Belief File.

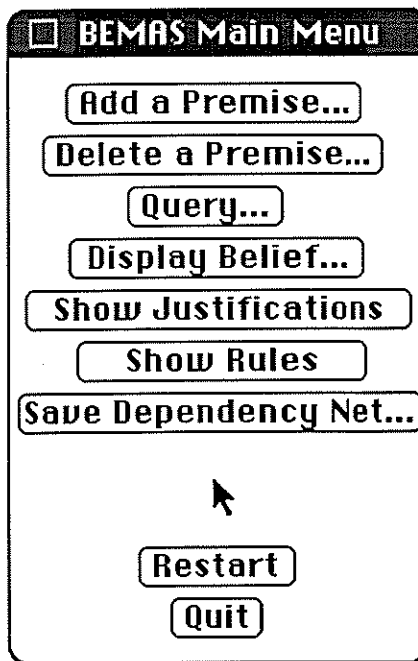


Figure 4: The BEMAS Main Menu

4. Options in BEMAS

BEMAS allows the user to execute the following eight options, as seen in the BEMAS option menu:

1. Add a premise
2. Delete a premise
3. Query
4. Display Belief
5. Show Justifications
6. Show Rules
7. Save Dependency Net
8. Restart
9. Quit

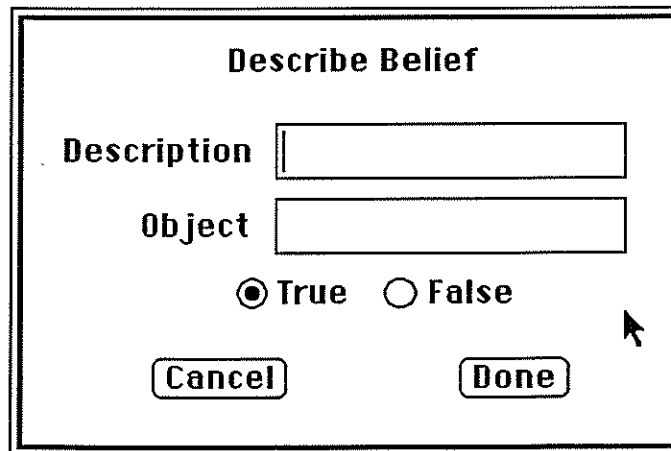
This section will explain the meaning of each option, and how BEMAS executes it and responds. In each example, the button in the BEMAS options menu will have the same name as the option described.

4.1. Adding A Premise

If the user clicks on the "Add a premise" button, BEMAS will ask for the three components of a premise (Section 2.1.1) given in the next menu (Figure 5):

4.1.1. The Belief Component Menu

This menu will be displayed every time the user must specify a belief for addition to the system, deletion from the system (Section 4.2), or querying the system (Section 4.3). We have discussed the three components of all beliefs when we discussed the format of an input belief. In the box marked **Description**, the user should type in the property of the belief. Then press the tab key to move to the box marked **Object**, where the user must enter a constant object, i.e.,



Describe Belief

Description

Object

True **False**

Figure 5: The Belief Component Menu

BEMAS: A Belief Maintenance System Prototype

no question mark should proceed it. The tab key is used to move back and forth between the description and object boxes. Once the user has correctly entered the property and object of the belief, the next step is to click the desired truth value. When the circle next to the truth value has been clicked, it will darken. A darkened circle denotes the truth value desired for the belief described.

Once the user has completely described the belief, clicking on the button marked **Done** will proceed with the operation the user requested prior to the appearance of the menu. If at any time you wish to abort this process, simply click the **Cancel** button, and the control will return to the BEMAS Main Menu.

Now we can return to our example of adding a premise. Once the three components of the belief are described, BEMAS will attempt to add the premise to the database. BEMAS will have to handle any one of four possible occurrences:

- a. The belief already exists at the premise level.
- b. The belief exists at a level other than the premise level.
- c. The belief does not exist anywhere in the database.
- d. The belief contradicts another belief at some level in the database.

We will show BEMAS's responses to any of the above occurrences. After clicking on the **Add a Premise** button, the user fills in the component menu. An example of a filled-in component menu is given in Figure 6.

Once **Done** is clicked, BEMAS will search the database for `is_mammal(tiger,true)`. Each possible occurrence above yields the following response from BEMAS.

- a. If the belief is at the premise level, BEMAS will respond with a message like that in Figure 7 and do nothing. When the user clicks on the OK button, control is returned to the BEMAS main menu.
- b. If the belief is at a lower level, BEMAS will respond with a message like that in Figure 7, followed by a message like that in Figure 8. But, depending on the belief the new premise

Describe Belief

Description

Object

True False

Figure 6: Example of Adding a Premise

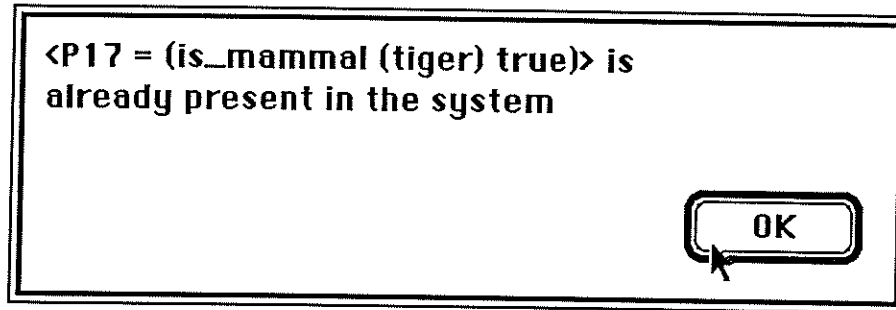


Figure 7: BEMAS response to case a.

replaces, BEMAS may have to relabel the dependency net so that each consequent dependent on the replaced belief is at the highest possible level.

Also, BEMAS will forward chain (section 4.1) using the asserted premise, i.e., it will fire as many rules as possible that are satisfied using premises and derived beliefs, asserting their consequents at the derived level, and using those consequents for further forward chaining.

- c. If the premise does not exist at all, BEMAS will respond with a message like that in Figure 8 and will then assert the belief and forward chain.
- d. If the premise contradicts another belief already in the database, BEMAS may respond in one of two ways. The first is that it may respond with a message similar to that of Figure 8.

With this message, BEMAS has been able to resolve the contradiction by changing one or more assumptions, or changing a premise that had no dependents. It then asserts the new premise and forward chains.

During this contradiction resolution, BEMAS may ask the user to remove one or more conflicting assumptions. The user will be presented with a menu of assumptions. You can click one if you want to select only one. If you want to select several, hold down the command key while clicking the ones that you wish to remove. Holding down command as you click a choice will also deselect a previously selected assumption. When you are satisfied with your choices, click the Delete button. Those assumptions will be removed, along with any inferred beliefs dependent on them, and the premise will be added.

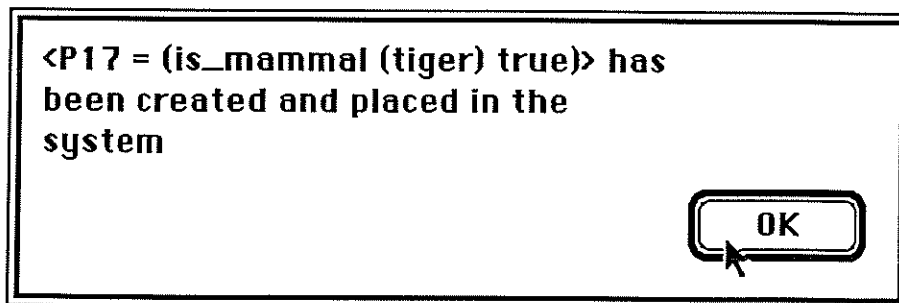


Figure 8: BEMAS response to case c.

BEMAS: A Belief Maintenance System Prototype

BEMAS could also respond in a way similar to:

CORRUPTION -- entering belief contradicts previous derived belief.

BEMAS responds in this way because it cannot resolve the contradiction, and adding the belief would make the world representation inconsistent. Thus, the user may have to find why the contradiction occurred using BEMAS's other options, and explicitly change those beliefs necessary in order to assert the premise.

4.2. Deleting A Premise

Clicking the **Delete a premise** option, BEMAS will prompt again for the three components that make up a belief. This is done with the menu in Figure 5, and it should be filled in exactly as described in Section 4.1.1.

When BEMAS attempts to delete the given belief, three possible things may occur:

- a. The belief is at the premise level
- b. The belief is not at the premise level (it may be at another level or it may not exist in the database at all)
- c. The belief exists at the premise level with the contradictory truth value

The following example will show BEMAS's response depending on which of a, b, or c occurred. BEMAS will search the database to see if the belief with its given truth value is at the premise level. If it is there, which is case 'a' above, BEMAS will remove the premise, along with any beliefs which depended solely upon its existence, without giving any message at all.

If either case 'b' or 'c' occurs, BEMAS will respond with

(error -- this premise does not exist in database...)

and do nothing but return the user to the BEMAS option menu.

4.3. Querying

Clicking on the **Query** button in the BEMAS option menu (Figure 4) allows the user to query BEMAS for the existence of a certain belief in the system. Before it begins to answer the query, BEMAS will again for the three components that make up a belief as in Figure 5.

After the **Done** is clicked in the component menu, BEMAS will begin to process the query. Four things could happen to cause BEMAS to respond differently.

- a. The belief exists at some level in the database with the desired truth value.

BEMAS: A Belief Maintenance System Prototype

- b. The belief exists at some level in the database with the opposite truth value.
- c. The belief does not exist and there is no possibility for proof.
- d. The belief does not exist and there is a possibility for proof.

BEMAS will search for the given belief to see if it is present at any level.

a. Existence of the belief makes BEMAS respond with something like Figure 9, with the level of the existing belief displayed in the window.

b. If a belief exists with the opposite truth value, BEMAS will respond with a message like in Figure 10.

c. No possibility for proof can be caused in two ways: (1) There exists no rule with which to prove the belief, or (2) There exists no well-founded support (premises) among any of the antecedents in order to satisfy any rules that do exist. Thus, BEMAS will respond in two different ways to let the user know what has gone wrong with the proof:

- (1). If there is no rule for which the belief is a consequent, then there is not enough information to achieve an answer to the question. Therefore, BEMAS will respond with a message like in Figure 11.
- (2) If there are rules, but either there is no premise to be a ground for the rule or contradictory beliefs exist to render the rule useless, then BEMAS can conclude that there is not adequate support for the belief to exist in the system. This is the response, as seen

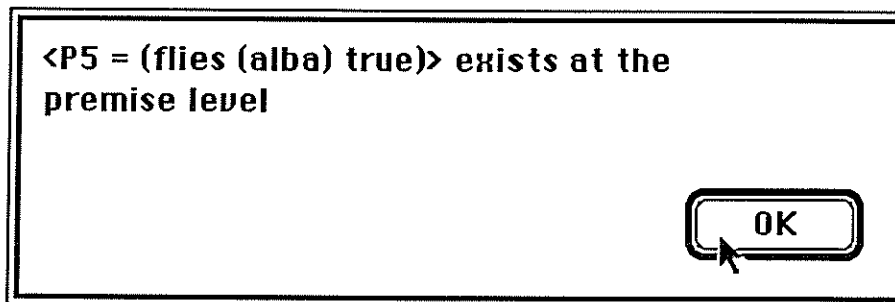


Figure 9: Querying for existing belief

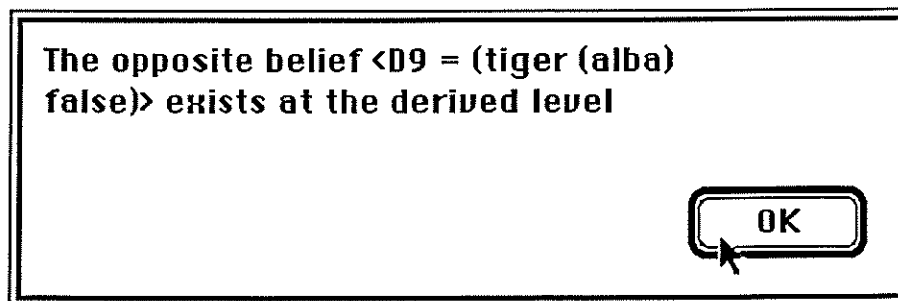


Figure 10: Querying for opposite of existing belief

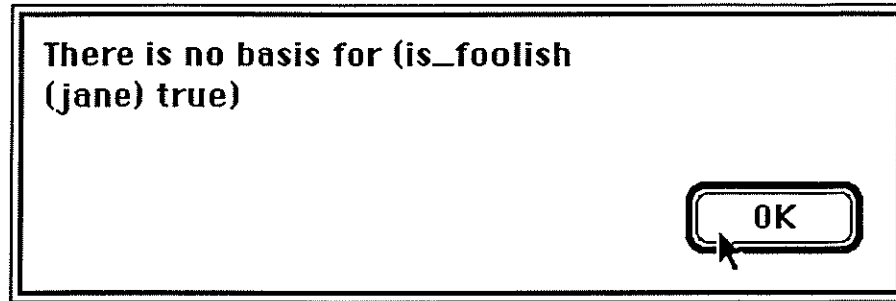


Figure 11: When a queried belief has no applicable rules

in Figure 12.

d. The possibility for proof exists if there is at least one rule, with at least one antecedent having a well-founded support. If the belief can be deduced, then BEMAS will respond with an answer as in Figure 13.

Otherwise, BEMAS may have to ask for some assumptions to be made if it cannot deduce the needed belief directly. The question will be similar to that in Figure 14.

If the yes button is clicked, then BEMAS may continue to ask for more assumptions or may obtain a proof from just one. If the no button is clicked, BEMAS may still ask for more assumptions, corresponding to other proof paths. If no other proof can be obtained, BEMAS will

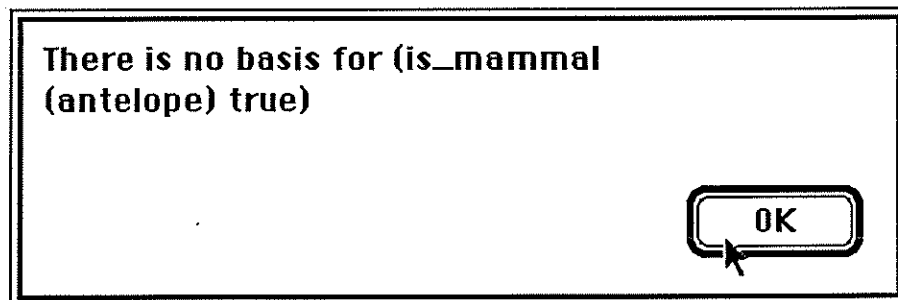


Figure 12: A queried belief does not have adequate support

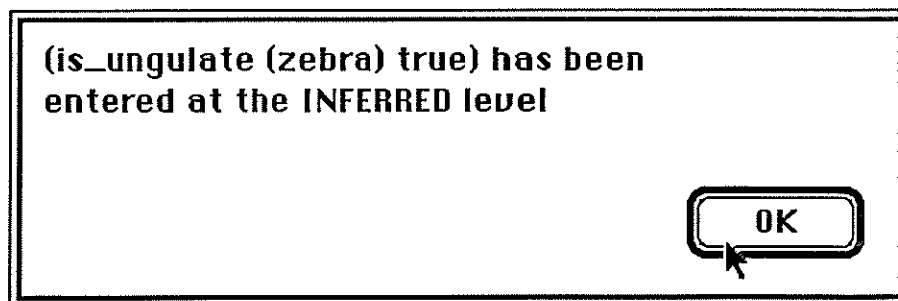


Figure 13: A successful query result

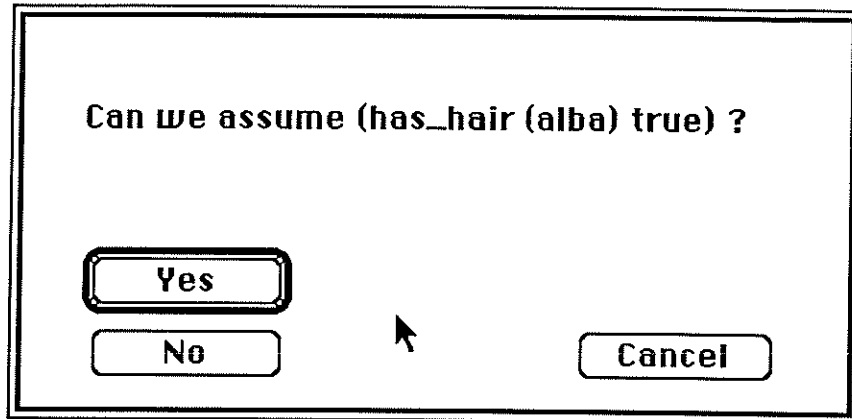


Figure 14: Assumptions need to be made to support query

respond as in case c. (2) above (Figure 12).

4.4. Display Belief

Display Belief is used for three reasons. Given some belief, let's say, I3 (an inferred belief), we want to show what beliefs, if any, are descendants of I3. Another reason would be to show what beliefs, if any, are ancestors of I3. The last use for Display Belief is to display the entire dependency net. By descendants, we mean any beliefs that exist because I3 exists. These beliefs may directly depend on I3, or may be among a chain of beliefs that depend on I3. By ancestors, we mean those beliefs that have an effect on the existence of I3. These beliefs may be direct antecedents of some justification which has I3 as a consequent, or these beliefs may be part of a chain which supports I3.

Once the **Display Belief** button is clicked in the BEMAS main menu, a sub-menu will appear as in Figure 15.

The five entries in the sub-menu allow the user to choose from which level they wish to display a belief. It also allows them to choose from the set of all beliefs, without regard to the level, by clicking on the button **Any**. If a button is "grayed out", it means that there are no beliefs of that type, and the button cannot be clicked.

Once the user clicks a button (except **All**), another menu will appear as in Figure 16. The user must scan through the beliefs displayed in the window of the menu. This window only holds a column of ten beliefs at a time. If more beliefs are present at the chosen level, horizontal scroll bars will appear at the bottom of the menu. Using the scroll bars, the user can display the beliefs by traversing through the columns.

When the desired belief is found, clicking it will darken it. Next, the user must decide whether to view the antecedents or the dependents. Clicking in the desired circle will darken it. When the **Display** button is clicked, the graph that appears will be similar to Figure 17. At any time the user may cancel viewing the beliefs.

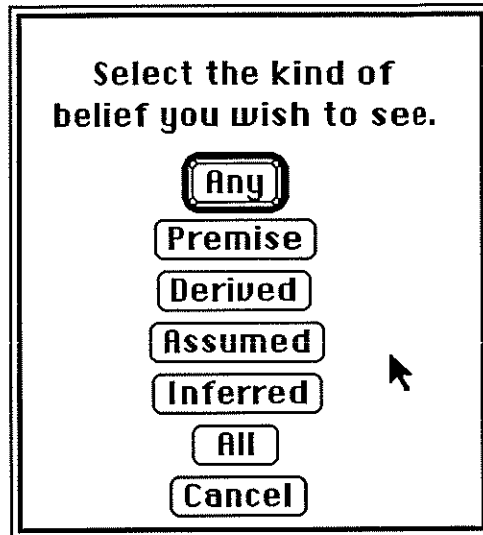


Figure 15: Display Belief Sub-Menu

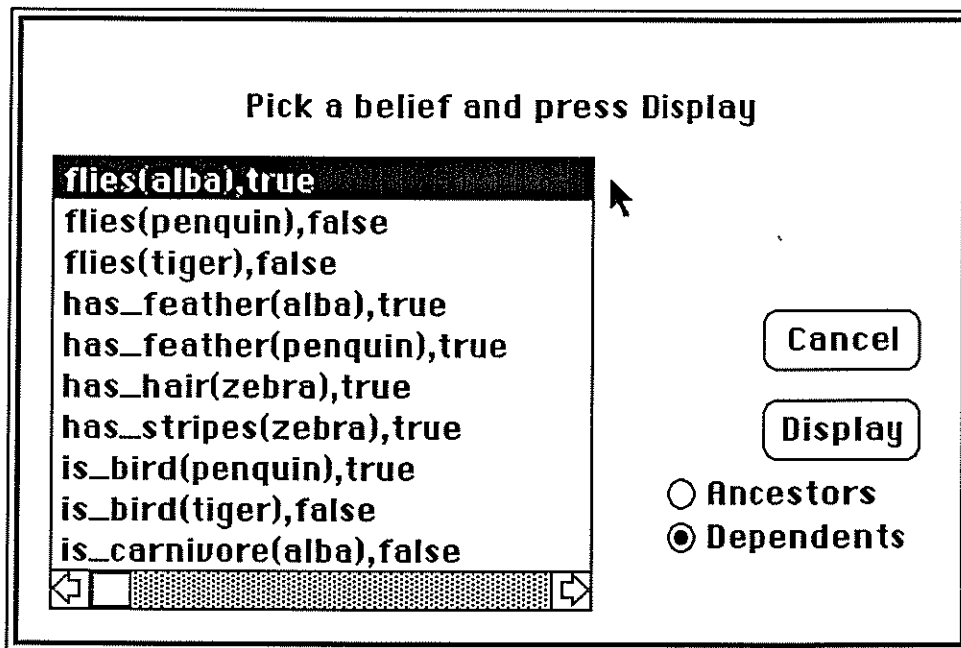


Figure 16: Select a Belief Sub-Menu

4.4.1. The Graphical Display

In Figure 17, a graphical display of the premise (*flies (alba) true*) and its dependents is shown. In this display, a premise belief is given the shape of a square, with the label *P#*, where *#* represents the unique identification number of the belief. A derived belief has the shape of a square with rounded corners containing the label *D#*. Assumed and inferred beliefs have the same oval shape, with assumed beliefs containing the label *A#* and inferred beliefs containing the label *I#*. Each justification is displayed with the label *J#* inside.

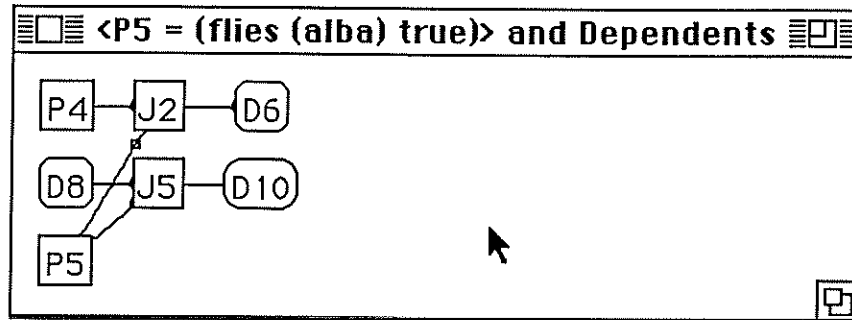


Figure 17: Example of Displayed Belief

The user can double-click on any item in the graphical display, and the corresponding belief or justification information will be printed in the Listener window. The format of this information is that discussed in Section 2.2.3.

4.5. Showing The Justifications In BEMAS Database

The user may click in the **Show Justifications** button in the BEMAS option menu, in order to show all the justifications BEMAS has created. BEMAS will print each justification, in the same format as in Section 2.2.3.2, in the Listener window. The justifications will be numbered in reverse order. This numbering shows the user when a specific justification was created with respect to the others, with the highest numbered justification being created most recently. An example of this display is shown in Figure 18.

Most likely, all of the justifications will not fit in the Listener window at one time. Therefore, the user may either expand the window to view all of them at once, or may simply use the scroll bars, or both.

Listener	
<pre> ant=<<P4 = <has_feather (alba) true>> <P7 = <is_carnivore (alba) false>> cons=<D8 = <is_bird (alba) true>> rule=<Rule 4> <Just 2 ant=<<P4 = <has_feather (alba) true>> <P5 = <flies (alba) true>> cons=<D6 = <cheetah (alba) false>> rule=<Rule 13> <Just 1 ant=<<P1 = <has_hair (zebra) true>> cons=<D2 = <is_mammal (zebra) true>> rule=<Rule 1> ? </pre>	

Figure 18: The Listener window displaying justifications

4.6. Showing The Rules In BEMAS Database

Clicking the **Show Rules** button in the BEMAS option menu will display each rule in the knowledge base inside the Listener. The rules will be presented in the same format as in Section 2.2.2. The rules will also be displayed in reverse order, even though they are theoretically entered at the same time. An example of this is presented in Figure 19.

As with justifications, all the rules in the knowledge base may not fit in the Listener at once. The Listener window must either be expanded or the scroll bars must be used.

4.7. Saving the Dependency Net

This option allows you to save the state of the system. Clicking the **Save Dependency Net** button will bring up a standard Macintosh window for specifying the directory and name of a file to save (see Figure 20). Directories can be changed in the same way as described in Section 3.2. Type the name that you wish the new dependency net file to have. To actually save the dependency net, either press Return or click the **Save** button. Click **Cancel** if you decide not to save the net.

4.8. Restarting BEMAS

To restart BEMAS and make it return to the Startup Options menu (Section 3.2), click the **Restart** button. BEMAS will first ask you if you would like to save the dependency net (see Figure 21). If you click **Yes**, then proceed as described in the previous section to save the net. After it has been saved, the menu titled **Welcome to BEMAS** will appear as the **BEMAS Main Menu** disappears. You now have a clean slate to work with.

If you don't want to save the net, click **No** and BEMAS will restart. If you decide that you do not wish to restart, but would rather continue with the current beliefs, click **Cancel** and control will go back to the main menu.

Listener	
<pre>cons=(is_mammal (?x) false) justs=() (Rule 2 ant=<<gives_milk (?x) true>> <<flies (?x) false>>) cons=(is_mammal (?x) true) justs=() (Rule 1 ant=<<has_hair (?x) true>>) cons=(is_mammal (?x) true) justs=<<Just 1>>) ?</pre>	

Figure 19: The Listener window displaying rules

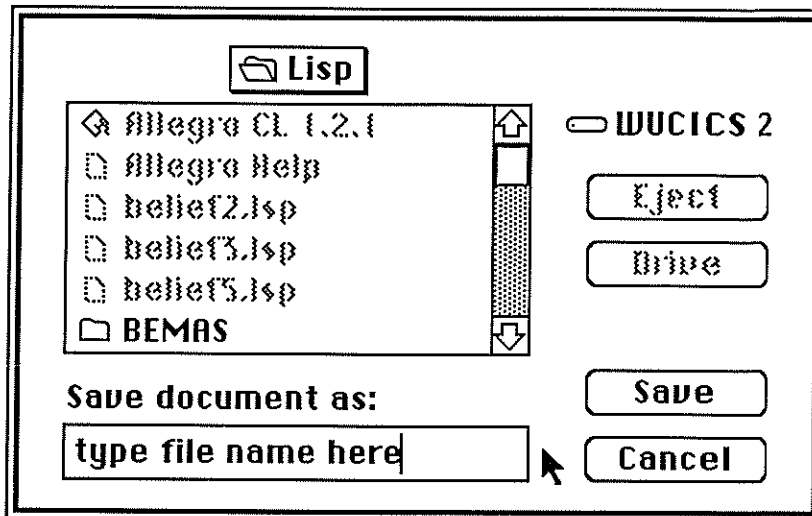


Figure 20: Saving a dependency net

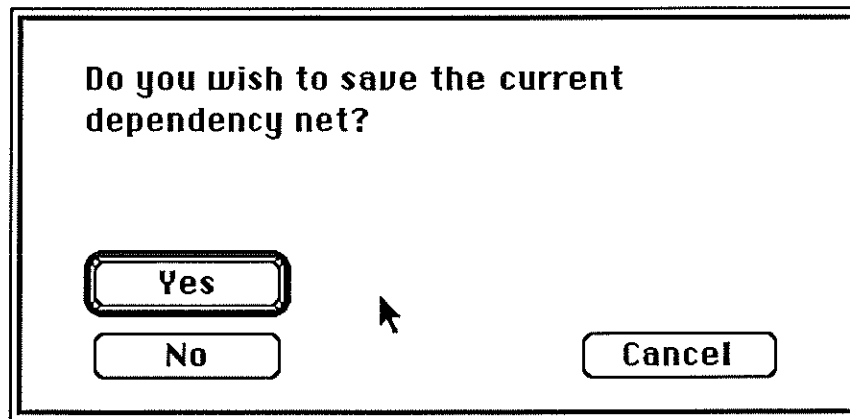


Figure 21: Last chance to save before restarting or quitting

4.9. Leaving BEMAS

To leave BEMAS, click **Quit** from the BEMAS main menu. You will be asked if you wish to save the dependency net, just as if you had clicked **Restart**. In this case, however, both BEMAS and Lisp will quit if you do not cancel.

5. Error Handling

At this time, there is very little error detection capability in BEMAS. Rule, belief, and dependency net files are assumed to be in the correct form.