Report Number: WUCS-87-14

1987-07-01

BEMAS: A Belief Maintenance System Prototype User's Manual

Roseanne M. Fulcomer, S. Melody Shief, 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 to make... Read complete abstract on page 2.

BEMAS: A Belief Maintenance System Prototype User's Manual

Roseanne M. Fulcomer, S. Melody Shief, 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: A BELIEF MAINTENANCE SYSTEM PROTOTYPE USER'S MANUAL

Rosanne M. Fulcomer, S. Melody Shieh and William E. Ball

WUCS-87-14

July 1987

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

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

BEMAS:
A Belief Maintenance System Prototype
USER'S MANUAL

Rosanne M. Fulcomer
S. Melody Shieh
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 Douglas Company under contract Z71021.

TABLE OF CONTENTS

BEMAS:
A Belief Maintenance System Prototype
USER'S MANUAL

Rosanne M. Fulcomer
S. Melody Shieh
William E. Ball

WUCS-87-14

1. Introduction

BEMAS, which stands for **BE**lief **MA**intenance System, is an Artificial Intelligence Prototype, written in C-Prolog.

The objective of the Belief Maintenance System is to deal with incomplete or uncertain information. Also it should 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 at this time.

Within this user's manual, different fonts will be used to identify the difference between the system's responses which will be in italics and the user's responses in bold font. Anything in brackets, <>, will designate that the content of the brackets is input for which the user is responsible.

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.

There are two types of files that BEMAS will accept:

Premises and Rules
Dynamic Dependency Net and Knowledge Base

"Premises and Rules" is the input file used when first entering BEMAS. This file consists of initial beliefs and rules for the system. Their format will be discussed in detail in the section 1.2. The "Dependency Net and Knowledge Base" is both an input file and an output file according to BEMAS. After using the "Premise and Rules" file initially, BEMAS will take input and transform it into the data structures it recognizes. When BEMAS writes out the contents of the file it is working on, it writes it in the form of the "Dependency Net and Knowledge Base" file. Then when the user wants to use the initial file again plus the changes made by BEMAS, the user has the "Dependency Net and Knowledge Base" file as input. The contents and format of this file will be discussed in detail in section 1.3. When the user desires to change any rules, the rules must be changed in the "Premises and Rules" file. Then this changed file must be used as the input file. We will usually refer to the "Premises and Rules" file as the initial input file, and the "Dependency Net and Knowledge Base" file as the output file.

2.1. Setting Up The Initial Input File

The following section will discuss the contents and usage of the initial input file. Instructions on formatting and loading the file will be provided.

2.1.1. Premises In The Initial Input File

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. Other types of beliefs will be discussed in section 2.2.2.1. This premise belief should be in following format:

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. All premises must be immediately followed by a period. On notation, words to describe objects and properties must be joined with an underscore, as seen the above example using can_fly.

2.1.2. Rules In The Initial Input 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 3.1 and 3.3. In the input file, the rules should be in the following format :

```
C :- A1,
     A2,
      .
      .
      .
     An.
```

where ":-" is a special symbol that acts as the reverse of implication symbol, i.e. "c :- a,b" means "a & b ==> c" or "if a and b, then c." The antecedents of the rule are a and b, and the consequent is c. Consequents and antecedents should be in the following format:

property(Variable,truth_value)

This format is similar to the format of premise, except that the first letter of the object is an upper case letter, representing a Variable, instead of a lower case letter which represents a constant. Multiple antecedents in this format are separated by a comma with the final antecedent followed by a period. An example of this follows.

```
property1(X,true) :- property2(X,true),
                     property3(X,false).
```

which says that, for all objects, X, if X has property2, and X does not have property3 then we can deduce that the X has property1. 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 follows:

```
is_bird(Y,true) :- has_feathers(Y,true),
                   flies(Y,true).
is_bird(Y,false) :- has_feathers(Y,true),
                    flies(Y,true).
```

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, i.e. 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.

Given the following rules from the domain of animal recognition:

is_bird(X,true) :- lays_eggs(X,true),
flies(X,true).

penguin(X,true) :- is_bird(X,true),
flies(X,false),
swims(X,true),
is_black_and_white(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:

is_bird(X,true) :- has_feathers(X,true)

or the antecedent is_bird(X,true) could be deleted from the penguin rule and possibly 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 (see listing of the rules in section 6).

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 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 2.2.2.2.

2.1.3. Ending the Initial Input File

To end the initial input file, after specifying every premise and rule, the statement:

stop.

exactly as written, should be placed on the last line of the input.

2.2. Contents Of The Output File

When the user quits from BEMAS, BEMAS will write to the output file the last representation of the world. BEMAS creates the output file the way that BEMAS sees it. Thus, the structure is quite different than what is seen in the input file. The output file also contains more information than the input file. This file is divided into three sections: dependency net, which holds all the beliefs and justifications in the current representation; knowledge base, which consists of the rules that were given to BEMAS from the initial input file; id roots, which are explained in the following section. This file is considered the "Dependency Net and Knowledge Base" file which can also be used as input (see section 2.1).

2.2.1. Id Roots

At the end of every output file created by BEMAS, will be three assertions called "id roots". An example is as follows:

current_num(b,9).
current_num(j,1).
current_num(r,26).

The purpose of the id roots is to let BEMAS know what was the last unique identification number given to the beliefs, justifications, and rules, so that when the output file is reloaded (as an input file) BEMAS can begin numbering the beliefs and justifications from where it left off. Any addition or deletions of rules must be done in the "Premises and Rules" file. Thus, the id root for the rules will not change or even be used when the output file is reloaded. For the user, the id roots can be used to give a quick count of how many beliefs, justifications and rules are currently in the world. For example,

current_num(b,10)

says that there are 10 beliefs currently in the world, i.e. "b" stands for "beliefs", and 10 is the count. Thus, in the first example of id roots, there are 3 beliefs, 1 justification, and 26 rules in the world.

2.2.2. Dynamic Dependency Net

As previously stated the 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 premise nor assumption, but have been deduced by BEMAS.

2.2.2.1. Beliefs

There are considered to be four levels of beliefs, which form a collection. Within this collection of beliefs each level is a field that is itself an array of records, where a single belief represents one record in the array as distinguished by its unique identification number, a "b" followed by a number. 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 3.1) and need justifications to exist.

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 3.3). Inferred beliefs need justifications to exist. The assertion of an inferred belief causes no forward chaining.

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 3.3.

For use in BEMAS, a single belief is represented as follows:

Level(Id,Description,Truth_value,L1,L2,L3,L4).

The fields of a single belief are as follows:

Description:	property(object,truth_value)
Truth Value:	(repeated from description): "true" or "false"
L1:	is a list of applicable rules for belief as a consequent
L2:	is a list of justifications with belief as its consequent
L3:	is a list of applicable rules for belief as an antecedent
L4:	is a list of justifications with belief as its antecedent

where the list of applicable rules contains the unique identification number of each rule, which is explained in section 1.3.2.3. The list of justifications will contain the unique identification number of each justification used for that belief which will be explained in next section.

An example of a belief is as follows:

derived(b1,is_bird(albatross,true),true,[r4,r2],[],[r5],[j12]).

where [ ] is the notation for a list.

2.2.2.2. 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, a "j" followed by a number.

For use in BEMAS, the justification is in the following format:

justification(Id,Rule,Ants,Con).

The fields in a justification are as follows:

Rule:	The identification number of the rule (section 1.3.2.3) that is justified by this justification
Ants:	The list of instantiated antecedents of the justified rule
Con:	The instantiated consequent of the justified rule

An example of this is:

justification(j12,r5,[has_hair(tiger,true)],is_mammal(tiger,true)).

2.2.3. The Knowledge Base

BEMAS creates a knowledge base from rules given to the system. This knowledge base is what BEMAS uses to make inferences. The rules are taken from the "Premises and Rules" 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 changed interactively with BEMAS, nor can they be changed directly within the output file. If the user desires to change the rules in the initial input file, BEMAS must be given these rules in their initial form, from the "Premises and Rules" input 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 set of rules are stored in the knowledge base in an collection of records, where each rule is distinguished by it's unique identification number, an "r" followed by a number.

For use in BEMAS, a rule in the output files will be in the format:

rule(Id,Con,Ants,Just).

The fields in each rule are as follows:

Con:	is the consequent of the rule having the format: property(object,truth_value)
Ants:	is the antecedents beliefs of the rule having the format: [property1(object,truth_value1),.....]
Just:	is a list of justifications which justifies some instantiated consequent of this rule

where the list of justifications contains the unique identification numbers of the justifications used, as discussed in section 1.3.2.2. 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 is as follows:

rule(r10,is_bird(X,true),[flies(X,true)],[j1,j8]).

3. Working In The C-Prolog Environment

All input into C-Prolog, as well as input to BEMAS, must end with a period. If the user finishes the input with a carriage return without period, C-Prolog will give the prompt

```
|:
```

indicating that it is waiting for more input. Another carriage return and the same prompt will return. This will be ongoing within both C-Prolog and BEMAS until a period is typed. The only benefit of this to the user is that the input will not be transferred by an accidental carriage return.

3.1. Entering And Exiting C-Prolog

All examples are from running C-Prolog and BEMAS on a VAX 750 running UNIX BSD 4.3. C-Prolog is loaded by typing the following command at the system prompt:

```
% prolog <cr>
```

The system will respond with

```
C-Prolog version 1.5
| ?-
```

when C-Prolog has been loaded. The "| ?-" is the C-Prolog prompt. After the user has completed the desired work in BEMAS and decides to quit from BEMAS (which will be discussed is section 3.8), the C-Prolog prompt will be returned.

The user can quit the C-Prolog environment, by typing the command:

```
| ?- exit.
```

and the system prompt will return.

3.2. How To Load BEMAS

BEMAS is loaded by typing:

```
| ?-[bemas].
```

at the C-Prolog prompt, where [file] means to consult the file, i.e. load it into C-Prolog. The word 'bemas' must be in lower case letters, since anything beginning in upper-case letters is considered a variable. The file containing BEMAS is a constant, and must be written in lower

case.

After loading BEMAS, the user will get the following prompt :

bemas consulted 38676 bytes 8.889 sec.
yes
| ?-

The first statement gives the size of BEMAS and the time it took to consult the file. The "yes" indicates that BEMAS has been successfully consulted or loaded. The "| ?-" waits for your next command.

3.3. How To Run BEMAS

Once the user has loaded the file where BEMAS is stored and receives the C-Prolog prompt, the following must be typed:

| ?-**bemas.**

exactly as seen. After this command, BEMAS will say:

Welcome to the Belief Maintenance System !

Please indicate the input data structure
1. Premises and Rules
2. Dynamic Dependency Net and Knowledge Base
|:

The first time using BEMAS, the user should respond to the prompt with '1.' to load the initial data file, the contents of which are discussed in section 1.2. BEMAS will transform the premises and rules into the structures it needs. Also, BEMAS will use the premises that it has to satisfiy 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. Next time using BEMAS, the user can load the output file directly to BEMAS by responding to the above prompt with '2.'. The detail description of this file is available in section 1.3.

After responding to the above prompt (with a '1' or a '2.'), BEMAS will ask the user to enter the name of the file BEMAS should use, by:

Please enter your input file name
|:**<enter file name>**

After specifying the input file name, BEMAS will ask the user to enter the name of the file to be output, by:

Please enter your output file name
¦:**<enter file name>**

If the same name is entered for the output file as the input file, BEMAS will overwrite the contents of the input file, thus destroying it. For this reason, it is always best to output to a different file than the one where the input is stored. The case is the same for option '1' or '2' above.

3.4. The BEMAS Option Menu

After the user specifies the output file, the BEMAS option menu will appear as follows:

BEMAS options :
1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

¦:

This menu guides the user through interaction with BEMAS. The individual options will be discussed in detail in the next section.

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. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

This section will explain the meaning of each option, and how BEMAS executes it and responds. In each example, the BEMAS options menu will be given to show the user response to that menu, and the actions that occur due to that response.

4.1. Adding A Premise

If the user responds with '1.' to execute the "Add a premise" option, BEMAS will ask for the three components of a premise (section 1.2), which are as follows:

Object : **<key in the object>.**
Property : **<key in the property>.**
Truth_value : **<key in 'true' or 'false'>.**

The same components are needed to identify any belief in the database, no matter at what level it is. Once the truth value is entered, BEMAS will attempt to add the premise to the database. BEMAS will have to handle any one of four possible occurrences:

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

The following example will show BEMAS's responses to any of the above occurrences.

BEMAS options :

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit
|: 1.

Object: **tiger.**
Property: **is_mammal.**
Truth_value: **true.**

then, BEMAS will search the database for is_mammal(tiger,true). Each possible occurrence above yields the following response from BEMAS.

a. If the premise is at the premise level, BEMAS will respond with:

No! premise is_mammal(tiger,true) is already in database.

and do nothing.

b. If the premise is at a lower level, BEMAS will respond with:

Ok! is_mammal(tiger,true) has been added into the database.

But depending on the belief the new premise 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 3.3) 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

The premise is_mammal(tiger,true) has been added.

and BEMAS will 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:

1. *Ok! is_mammal(tiger,true) has been added into the database.*

With this prompt, 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 change one or more conflicting assumptions. If the necessary assumptions cannot be changed,

then BEMAS will respond as in the next example:

Contradiction occurs !.
The following assumption must be changed :
lays_eggs(tiger,true)

Change the truth value of lays_eggs(tiger,true) ?(n/y)
|: **n.**
Sorry! Corruption.
is_bird(tiger,false) is not added into the database.

Otherwise, BEMAS will respond as follows:

2.

Sorry! Corruption.
is_mammal(tiger,true) is not added into the database.

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

Responding with a '2.' will execute the "Delete a premise" option. BEMAS will prompt again for the three components that make up a belief.

Object: **<key in the object>.**
Property: **<key in the property>.**
Truth_value: **<key in 'true' or 'false'>.**

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

a. The premise is at the premise level
b. The premise is not at the premise level (it may be at another level or it may not exist in the database at all)
c. The premise 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 options :

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **2.**

Object: **tiger.**
Property: **is_mammal.**
Truth_value: **true.**

BEMAS will search the database to see if the belief with its given truth value, is_mammal(tiger,true), is at the premise level. If it is there, occurrence 'a', BEMAS will respond with

Ok! is_mammal(tiger,true) has been deleted from the database.

With this response, BEMAS has deleted is_mammal(tiger,true) from the database along with any beliefs which depended solely upon the existence of is_mammal(tiger,true).

If 'b' or 'c' occur, BEMAS will respond with

Sorry! No such premise existed in database.

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

4.3. Querying

Responding with a '3.' to the BEMAS option menu, allows the user to query BEMAS for the truth value of a certain belief. Before it begins to answer the query, BEMAS will prompt again for the three components that make up a belief

Object: **<key in the object>.**
Property: **<key in the property>.**
Truth_value: **<key in 'true' or 'false'>.**

After the truth value is entered, BEMAS will begin to process the query. Four things could happen to make BEMAS respond differently.

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

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.

The following example will demonstrate BEMAS's responses to each of the four occurrences.

BEMAS options :

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit
|: **3.**

Object: **tiger.**
Property: **is_mammal.**
Truth_value: **true.**

then, BEMAS will search for the belief "is_mammal(tiger,true), to see if it is in any level of belief. Its response to the above occurrences depend on the outcome of the search.

a. Existence of the belief makes BEMAS respond with:

Yes, is_mammal(tiger,true) is at premise level

or

Yes, is_mammal(tiger,true) is at derived level

or

Yes, is_mammal(tiger,true) is at inferred level

or

Yes, is_mammal(tiger,true) is at assumed level.

b. For the belief with the opposite truth value in the database, BEMAS will respond:

Sorry! The opposite belief is_bird(tiger,true) is at premise level.

c. No possibility for proof is caused from two things: (1) There exists no rule with which to prove the belief, or (2) There exists no well-founded support among any of the antecedents in order to satisfy and rules that do exit. Thus, BEMAS will ask the user:

Unknown!
Do you want to try to query is_mammal(tiger,false)?(y/n)

A 'y' to the above question will send BEMAS to attempt to prove the opposite belief. If the opposite belief has the same occurrence (c.), then BEMAS will respond with:

Sorry! is_mammal(tiger,false) cannot be proved either.

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 follows:

Yes, tiger(tiger,true) is at inferred level.

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

Assume has_hair(tiger,true) true? (y/n)
|:

If 'y.' is given by the user, then BEMAS may continue to ask for more assumptions or may obtain a proof from just one. If 'n.' is answered, BEMAS may still ask for more assumptions, for other proof paths. If no other proof can be obtained, BEMAS will respond as in c.

4.4. Showing The Dependencies Of A Belief

Option '4.' Show dependencies' would be used for two reasons. The first would be to show what beliefs, if any, are dependents (or consequents) of a given belief. The other reason would be to show what beliefs, if any, are needed for a certain belief to exist in the database.

Once '4.' is chosen from the BEMAS Option menu, a submenu will appear as follows:

1. *Show consequent*
2. *Show antecedents*

|:

Responding with a '1.' to the submenu, will make BEMAS ask for the three components of a belief.

Object: **<key in the object>.**

Property: **<key in the property>.**
Truth_value: **<key in 'true' or 'false'>.**

Once the belief is described, BEMAS will return the direct consequents of that belief. For an example, see section 3.4.1.

Choice '2.' will also make BEMAS ask for the components of a belief and then will return the antecedents of that belief. In other words, BEMAS will return all of the beliefs that the given belief depends on for existence. For an example, see section 3.4.2.

4.4.1. Showing The Consequent Dependencies

The following example is what BEMAS does when the user answers the prompts as follows:

BEMAS options :

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **4.**

1. *Show consequent*
2. *Show antecedents*

|: **1.**

Object: **tiger.**
Property: **is_mammal.**
Truth_value: **true.**

Three different answers will be given depending on the status of the belief.

1. The belief, is_mammal(tiger,true), does not exist in the database. BEMAS will respond with:

   Sorry! There is no such belief, is_mammal(tiger,true) in database.

2. The belief, is_mammal(tiger,true), exists but has no justified consequent, then BEMAS will prompt with:

   Sorry! There is no consequent dependencies for is_mammal(tiger,true).

3. If the belief exists and has some number, N, justified consequents, then BEMAS will prompt with :

There are N consequent dependencies for is_mammal(tiger,true):

and BEMAS will list the consequents in the format of:

property(object,truth_value)

4.4.2. Showing The Antecedent Dependencies

The following example explains what BEMAS does for the user assuming that the user answers the previous prompts as follows:

BEMAS options :

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **4.**

1. *Show consequent*
2. *Show antecedents*

|: **2.**

Object: **tiger.**
Property: **is_mammal.**
Truth_value: **true.**

BEMAS will respond differently to the following three circumstances:

1. The belief, is_mammal(tiger,true), does not exist in the database. BEMAS will prompt with:

   Sorry! There is no such belief, is_mammal(tiger,true) in database.

2. The belief, is_mammal(tiger,true), exists but has no justified antecedent(s). BEMAS will prompt with:

   Sorry! There is no antecedent dependencies for is_mammal(tiger,true).

3. The belief exists and has some number, N, needed antecedents. BEMAS will prompt with:

   There are N antecedent dependencies for is_mammal(tiger,true):

and list out all the antecedents in the following format:

property(object,truth_value)

4.5. Showing The Beliefs In The BEMAS Database

The fifth option in the BEMAS option menu gives the user a chance to see what beliefs are at which level. BEMAS will bring up the following submenu:

Show beliefs :
1. Show premise
2. Show derived
3. Show inferred
4. Show assumed

|:

The options in this menu give the user the choice of which level to examine. The beliefs printed will be in the format:

property(object,truth_value)

1. If the user responds with '1.', BEMAS will display the premise beliefs on the screen.

2. If the user responds with '2.', BEMAS will display the derived beliefs on the screen.

3. If the user responds with '3.', BEMAS will display the inferred beliefs on the screen.

4. If the user responds with '4.', BEMAS will display the assumed beliefs on the screen.

5. If belief exists on the level of belief requested, BEMAS will prompt:

   Sorry! No premise (or derived or inferred or assumed) belief in current database.

4.6. Showing The Justifications In BEMAS Database

The user may type a '6.' in response to the BEMAS option menu, in order to show all the justifications BEMAS has created. BEMAS will print each justification in the following format:

[A11, A12, ...A1n]
===>
C1

[A21, A22, ...A2p]
===>
C2
.
.
.
[Am1, Am2, ...Ams]
===>
Cm

where A11 ... Ams are antecedents and C1 ... Cm are consequents. Each antecedent and consequent are printed using the following format:

property(object,truth_value)

An example of a justification printed in the above format is shown below.

[has_hair(tiger,true)]
===>
is_mammal(tiger,true)

4.7. Showing The Rules In BEMAS Database

Choosing the seventh option in the BEMAS option menu, will show each rule in the knowledge base, one at a time. The rules will be presented in the following format:

[A11, A12, ...A1n]
===>
C1

where C1 is a consequent and A11, A12,... are antecedents as in section 3.6, except that a dash followed by a number will replace the variable used in each rule. BEMAS reorganizes a variable to be a dash and a sequence of numbers so each distinct variable has a unique number, for instance : _2345, _2346. An example of such a rule is as follows:

[flies(_21,true)]
===>
is_bird(_21,true).

After each rule will appear:

Another rule (y/n)?

A 'y' will tell BEMAS to print the next rule. Continuous y's will give each rule until all have been printed and the following response is given.

No more rules in current knowledge base.

An 'n' will return the user to the BEMAS option menu.

4.8. Leaving BEMAS

To leave BEMAS, choose option '8.', the "Quit" option from the BEMAS option menu. BEMAS will automatically write the most recent world representation to the given output file (section 1.3). As BEMAS does this, it will print to the screen:

Writing updated database to <file name>.....

When BEMAS is through, it will signal the user again with

Done! Bye-bye....

and return the user to the C-Prolog environment by prompting with :

```
yes
| ?-
```

4.9. Reentering BEMAS

While still in the C-Prolog environment, the user can reenter BEMAS to work on the previous world representation that BEMAS had before the user exited by typing:

re_enter_bemas.

There will be no need to enter any files as BEMAS will work on the last one it was working with. Thus, the user will immediately receive the BEMAS option menu to continue working.

5. Error Correction

At this time there is no error correction capability in BEMAS. For instance, the user makes a mistake in entering the three components of a belief. The method to cancel any command to BEMAS is to type a 'control-c', i.e. hold the control key down while typing an 'c'. BEMAS will repond with:

Action (h for help):

responding with an <cr> will give the response

[execution aborted]

and return the user to the C-Prolog prompt, where the user can use the re_enter_bemas command (section 3.9), to go back to the program before the error.

6. An Example Using BEMAS

The following example should give the user an idea of what to expect from BEMAS. Rules from the animal recognition problem will form the knowledge base. At initialization, there are no premises in the database. The listing of the rules for this problem follows the example.

% **prolog**
C-Prolog version 1.5

| *?-*[**bemas**].

bemas consulted 38676 bytes 13.9833 sec.

yes
| *?-* **bemas.**
Welcome to the Belief Maintenance System !

Please indicate the input data structure
1. Premises and Rules
2. Dynamic Dependency Net and Knowledge Base
|: **1.**

Please enter your input file name
|: **mydata.**

Please enter your output file name
|: **out.**

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **1.**

Object: **bomber.**
Property: **flies.**
Truth_value: **false.**

Ok! flies(bomber,false) has been added into the database.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **3.**

Object: **bomber.**
Property: **is_mammal.**
Truth_value: **true.**

Unknown!
Do you want to try to query is_mammal(bomber,false)?(y/n)

|: **n.**

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **1.**

Object: **bomber.**
Property: **swims.**
Truth_value: **false.**

Ok! swims(bomber,true) has been added into the database.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **3.**

Object: **bomber.**
Property: **is_bird.**
Truth_value: **true.**

Unknown!
Do you want to try to query is_bird(bomber,false)?(y/n)

|: **y.**

Assume has_feather(bomber,false) true? (y/n)
|: **n.**

Sorry! is_bird(bomber,false) cannot be proved either.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **1.**

Object: **bomber.**
Property: **has_feather.**
Truth_value: **true.**

Ok! has_feather(bomber,true) has been added into the database.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **1.**

Object: **bomber.**
Property: **is_bird.**
Truth_value: **true.**

Yes, is_bird(bomber,true) is at derived level.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **3.**

Object: **bomber.**
Property: **albatross.**
Truth_value: **true.**

Sorry! The opposite belief albatross(bomber,false) is at derived level.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **3.**

Object: **bomber.**
Property: **ostrich.**
Truth_value: **true.**

Assume has_long_neck(bomber,true) true? (y/n)
|: **n.**

Unknown!
Do you want to try to query ostrich(bomber,false)?(y/n)

|: **y.**

Sorry! ostrich(bomber,false) cannot be proved either.

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **3.**

Object: **bomber.**
Property: **penguin.**
Truth_value: **true.**

Assume is_black_and_white(bomber,true) true? (y/n)
|: **y.**

Yes, penguin(bomber,true) is at inferred level.

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **6.**

[flies(bomber,false)]
--->
albatross(bomber,false)

[has_feather(bomber,true)]
--->
is_bird(bomber,true)

[is_black_and_white(bomber,true),swims(bomber,true),
flies(bomber,false),is_bird(bomber,true)]
--->
penguin(bomber,true)

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query

4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **5.**

Show beliefs :

1. Show premise
2. Show derived
3. Show inferred
4. Show assumed
5. Show all beliefs

|: **5.**

has_feather(bomber,true)
swims(bomber,true)
flies(bomber,false)
albatross(bomber,false)
is_bird(bomber,true)
penguin(bomber,true)
is_black_and_white(bomber,true)

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **4.**

Show dependencies :

1. Show consequent
2. Show antecedents

|: **1.**

Object : **bomber.**
Property : **is_bird.**
Truth_value : **true.**

There is only one consequent dependency for is_bird(bomber,true):
penguin(bomber,true)

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **4.**

Show dependencies :

1. *Show consequent*
2. *Show antecedents*

|: **2.**

Object : **bomber.**
Property : **is_bird.**
Truth_value : **true.**

There is only one set of antecedent dependency for is_bird(bomber,true):
[has_feather(bomber,true)]

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **2.**

Object: **bomber.**
Property: **has_feather.**
Truth_value: **true.**

Ok! has_feather(bomber,true) has been deleted from the database.

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **6.**

[flies(bomber,false)]
--->
albatross(bomber,false)

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **2.**

Object: **bomber.**
Property: **flies.**
Truth_value: **false.**

Ok! flies(bomber,false) has been deleted from the database.

BEMAS options:

1. Add a premise
2. Delete a premise
3. Query
4. Show dependencies
5. Show beliefs
6. Show justifications
7. Show rules
8. Quit

|: **1.**

Object: **bomber.**
Property: **flies.**
Truth_value: **true.**

Ok! flies(bomber,true) has been added into the database.

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **6.**

[flies(bomber,true)]
--->
ostrich(bomber,false)

[flies(bomber,true)]
--->
penguin(bomber,false)

BEMAS options:

1. *Add a premise*
2. *Delete a premise*
3. *Query*
4. *Show dependencies*
5. *Show beliefs*
6. *Show justifications*
7. *Show rules*
8. *Quit*

|: **8.**

Writing updated database to out.....
Done! Bye-bye.....

yes
| ?-

The following is a listing of the rules in the animal recognition problem as seen in the previous example.

```
/*      rules           */

is_mammal(X,true) :-
        has_hair(X,true).
is_mammal(X,true) :-
        gives_milk(X,true).
is_mammal(X,false) :-
        has_hair(X,false),
        gives_milk(X,false).


is_bird(X,true) :-
        has_feathers(X,true).
is_bird(X,true) :-
        lays_eggs(X,true),
        flies(X,true).
is_bird(X,false) :-
        has_feathers(X,false),
        lays_eggs(X,false).
is_bird(X,false) :-
        has_feathers(X,false),
        flies(X,false).


is_carnivore(X,true) :-
        eats_meat(X,true).
is_carnivore(X,true) :-
        has_pointed_teeth(X,true),
        has_claws(X,true),
        has_forward_eyes(X,true).
is_carnivore(X,false) :-
        eats_meat(X,false),
        has_pointed_teeth(X,false).
is_carnivore(X,false) :-
        eats_meat(X,false),
        has_claws(X,false).
is_carnivore(X,false) :-
        eats_meat(X,false),
        has_forward_eyes(X,false).


is_ungulate(X,true) :-
        is_mammal(X,true),
        has_hoofs(X,true).
is_ungulate(X,true) :-
```

```
        is_mammal(X,true),
        chews_cud(X,true).
is_ungulate(X,false):-
        is_mammal(X,false).
is_ungulate(X,false) :-
        has_hoofs(X,false),
        chews_cud(X,false).


cheetah(X,true) :-
        is_mammal(X,true),
        is_carnivore(X,true),
        has_tawny_color(X,true),
        has_stripes(X,false).
cheetah(X,false) :-
        is_mammal(X,false).
cheetah(X,false) :-
        is_carnivore(X,false).
cheetah(X,false) :-
        has_tawny_color(X,false).
cheetah(X,false) :-
        has_stripes(X,true).

tiger(X,true) :-
        has_tawny_color(X,true),
        is_mammal(X,true),
        is_carnivore(X,true),
        has_stripes(X,true).
tiger(X,false) :-
        is_carnivore(X,false).
tiger(X,false) :-
        is_mammal(X,false).
tiger(X,false) :-
        has_stripes(X,false).
tiger(X,false) :-
        has_tawny_color(X,false).

giraffe(X,true) :-
        is_ungulate(X,true),
        has_long_neck(X,true),
        has_long_legs(X,true),
        has_stripes(X,false).
giraffe(X,false) :-
        is_ungulate(X,false).
giraffe(X,false) :-
        has_long_neck(X,false).
giraffe(X,false) :-
        has_long_legs(X,false).
giraffe(X,false) :-
        has_stripes(X,true).

zebra(X,true) :-
        is_ungulate(X,true),
```

```
        has_stripes(X,true).
zebra(X,false) :-
        is_ungulate(X,false).
zebra(X,false) :-
        has_stripes(X,false).

ostrich(X,true) :-
        is_bird(X,true),
        flies(X,false),
        has_long_neck(X,true),
        has_long_legs(X,true).
ostrich(X,false) :-
        is_bird(X,false).
ostrich(X,false) :-
        flies(X,true).
ostrich(X,false) :-
        has_long_legs(X,false).
ostrich(X,false) :-
        has_long_neck(X,false).

penquin(X,true) :-
        has_feathers(X,true),
        flies(X,false),
        swims(X,true),
        is_black_and_white(X,true).
penquin(X,false) :-
        has_feathers(X,false).
penquin(X,false) :-
        flies(X,true).
penquin(X,false) :-
        swims(X,false).
penquin(X,false) :-
        is_black_and_white(X,false).

albatross(X,true) :-
        is_bird(X,true),
        flies_great(X,true).
albatross(X,false) :-
        is_bird(X,false).
albatross(X,false) :-
        flies_great(X,false).


stop.
```

7. References

1. R. M. Fulcomer, W. E. Ball, and J. P. Tadlock, Belief Maintenance Systems: Definition and Prototype Specification, WUCS-87-15, July 1987.