M.Tech (Computer Science) Dissertation Series

A Software for Maintaining a Conceptual Hierarchy of Objects to Aid in *Theta-Mapping* in a Natural Language Processing System For Bangla.

a dissertation submitted in partial fulfilment of the requirements for M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

JYOTISHMAN CHATTERJEE

under the supervision of

Probal Sengupta
B.B.Chaudhuri

INDIAN STATISTICAL INSTITUTE
203,Barrackpore Trunk Road, Calcutta-700 035

ACKNOWLEDGMENTS

I am indebted to Mr. Probal Sengupta and Dr. B.B. Chaudhuri for introducing me to the areas of Natural Language Processing(NLP) and guiding me throughout this work. Interests of Suresh and Panda in the present work are thankfully acknowledged.

CONTENTS

1 Introduction

Natural Language Processing (NLP) is primarily concerned with making the machine understand inherent meanings of sentences in a natural language. The concept of "inherent meaning" or "semantics" of a sentence is however quite fuzzy. Most workers on NLP explain the semantics of a sentence in the following manner. A (simple) sentence describes an "action". The action can be thought of as a drama being staged. The role of the name of the drama is played by the *verb*. Just like in scripts of actual dramas, various "roles" are specified for a verb. In a sentence, various *phrases* play the different roles of the verb of the sentence. In literature [1], the roles described above are called *thematic roles* or *theta* roles (θ−roles). Different verbs have different number of theta roles. Although linguists have not yet been able to propose a mutually agreeable set of θ-roles, some of the roles are quite standard. For example, almost all verbs have an **AGENT** θ-role, which signifies "who" performed the action described by the verb. Similarly, there are some verbs which describe actions where some object gets "affected", as in "Ram killed the tiger", where "tiger" gets affected. Such verbs signify the affected object by a **PATIENT** θ-role. In the appendix, a list of the θ-roles considered in the present project have been listed. The accepted definition of the semantics of a sentence is a mapping that associates the θ-role(s) of the verb of the sentences to the various phrases. For Bangla (Bengali), the phrases that play a θ-role in a sentence of any language are either *Noun Phrases (NP)* or *Prepositional/Post-positional Phrases (PP)*. In either type of phrases, there is a main *noun* or *pronoun* (A pronoun ultimately is just a wild-card for a noun) with other optional structures (words or subwords) decorating the main noun or pronoun. In the semantic definition of a sentence, the theta-mapping involves only the main or *head* noun of a phrase.

Since human beings are also required to understand meanings of sentences, it is imperative that they are able to perform theta-mapping in a fast and computationally efficient manner. In order to aid the brain to carry out the computation effectively, speakers of a particular language agree upon certain rules of *syntax* for the language. The syntactic rules specify how different entities — phrases, words, parts of words (called morphemes), etc. are arranged in a sentence. Generally speaking, and more so when one attempts to emulate human linguistic behaviour as in NLP, only syntactically correct sentences can have semantic descriptions. The rules of syntax are normally such that most of the theta-mapping can be done in the process of verifying the syntactic correctness of a sentence, (as in *syntax-directed translation* in compilers). In computer science, there is a branch of study dealing with algorithms for proving sentences to be syntactically correct. Such algorithms are called *parsers.*

Natural Language parsers are primarily involved in detecting the verb and the other phrases of a sentence along with the *case* of each phrase. Case is overt (normally) syntactic manifestation signified by a) order of occurrence of the phrase (as in English and similar languages), or b) Conjoining between the *stem* morpheme of the head noun of a phrase and a *case-marking inflection* (as in Indian languages), or c) a combination of a) and b) above. Different existing theories on NLP have different approaches for performing theta-mapping, the primary goal, from case-analysis of sentences. In NLP studies on Bangla (a major natural language of India and Bangladesh) carried out at the ISI [4,5,6,2], it was found that due to various reasons, the results of case-analysis at the parsing stage can not be directly used for theta-mapping. *An a-priori knowledge of which objects can play a given θ-role of a given verb, is necessary.* The present paper concerns the creation and maintenance of such a knowledge-base.

A naive method of storing the above knowledge-base is to have *sets* S_{ij} of objects (nouns), where i ranges over all verbs and j ranges over all θ-roles. A reasonable vocabulary must contain about 500 verbs. The number of theta roles used in the present project is 6, as given in the Appendix B.1. Thus in the simplest approach, there may be $500 \times 6 = 3000$ different *databases* corresponding the different S_{ij} s, where each database could be of maximum size l, the total number of objects in the vocabulary! Apart from the astronomical space requirement, this approach suffers from another major defect — the knowledge base does not have any *redundancy.* Also, it is quite difficult to keep track of the *criteria* by which pairs of sets differ. The facts that one set is superset of another, or that two sets are disjoint and that the disjoint union of the sets have been *split* on some *feature*, can not also be encoded in the naive approach.

The approach here is basically a "class based" approach. The knowledge base may be augmented in steps. Right at the beginning, no knowledge is assumed, i.e., all S_{ij} s are identical. Periodically, the linguist (the "expert" in our case) decides to concentrate on some θ-role of some verb. At any point of time, the set currently pointed at for some θ-role of some verb is *over estimated*, i.e., the set contains *all* objects from the overall set of objects that may play the theta role, and some more. The linguist is expected to *filter* out the spurious entries through interaction with the system. So as not the disturb the condition for over estimation, the linguist must consciously attempt not to filter out any object that he feels might have the slightest chance of remaining in the set. All the objects filtered out are assumed to belong to a single *exception* set. Either or both of the filtered set and the exception set may be already existing. However, if any new set is created, it is added to the existing knowledge base in such a manner that minimum of data movement is necessary. Also, feature-based reasoning for splitting a particular set is requested from the linguist and copied at the appropriate place. Later, the linguist may request the system to enumerate the reasoning on which the assignment of a certain set of objects to some θ-role of some verb was achieved. The class based approach is a practical application of conceptual clustering [3] to the field of NLP. It achieves maximum redundancy and captures the concepts on which clustering is carried out.

Throughout our discussion, we shall assume that the user of the system is an *expert* who can guide the system properly for any further rectification of the knowledge accumulated so far. Hence the user in our case is a linguist. Thus, it is tacitly assumed that:

$$\boxed{\textbf{User} = \textbf{Expert} = \textbf{Linguist}}$$

2 Approach to the Solution

Ask somebody the question - "Who all can play the role of *Agent of eat*"? Most likely the answer will be - *animates*. The general nature of the above question is "who can play the Θ_i -role of V_j"! Here, $\Theta = \{\Theta_1, \Theta_2, \ldots, \Theta_k\}$ is the set of θ-roles and $V = \{V_1, V_2, \ldots, V_m\}$ is the set of verbs. Thus, there can be $n = k \times m$ different query targets. We shall consider the set $\{\theta_1, \theta_2, \ldots, \theta_n\}$ to be the set of query targets where θ_i refers to some θ-role Θ_j of some verb V_k. With this, we get a one-dimensional set of queries. We shall call a query target θ_i to be a **T-role**. In a fully classified knowledge base (as in the naive approach discussed in section-1), a T-role θ_i is mapped to some set S_i of objects, where S_i could be the null set $\emptyset$. However, conceptually, we human beings adopt a class-wise approach in answering such questions. This gives rise to an answer "animates" for the query given at the beginning of the paragraph. A class is different from a set in the sense that while in a set the members are simply listed, a class has tagged along with it a *reasoning* or *concept* of why or how the members belonging to it were obtained. This is the central idea of *conceptual clustering* [3].

In the present work, we propose a formalism, using which a linguist can create a hierarchy of classes through step-wise refinement. At any instant, every class will contain objects that satisfy the tagged concept of the class and some more. Every T-role will point to zero or more classes. The idea about the hierarchy of classes is explained below.

The class called *Object-space* is in the top of the hierarchy and contains *all* objects. This is *divided* into different subclasses depending on different *features*. Each feature *divides* the *Object-space* into several non-overlapping classes. But classes arising from different features may overlap.

Example:

Feature No.1 $\xrightarrow{animacy}$ *plant kingdom, animal kingdom, inanimate.*

Feature No.2 $\xrightarrow{visibility}$ *visible, invisible.*

Plant kingdom, animal kingdom and *inanimate* are three classes given by the same feature *animacy*, so they should be pairwise disjoint. But *plant kingdom* and *visible* are two classes given by two different features viz. *animacy* and *visibility*. Thus there is no problem if they overlap and they will indeed do so.

The classification has been shown in Figure-1. The dash line separates the classification done by two features. We will follow this convention whenever it is needed to represent some hierarchy.

All the classes appearing below the class *Object-space* are first level classes and they can also be divided using the same technique. For example, take the class *inanimate*. It is divided into different subclasses depending on three features as follows:

Feature No.1 $\xrightarrow{naturality}$ *natural (not human created), artificial.*

Feature No.2 $\xrightarrow{colour}$ *red, green, blue, other.*

Feature No.3 $\xrightarrow{use}$ *used in daily life, useful but not of daily use, of no use.*

See Figure-2.

Suppose, the class *animal kingdom* is divided based on only one feature viz. whether they are *higher level* or *lower level* (Figure-3). As there is only one feature, no dash line was used. If that is all about our classification then the total hierarchy will be given by Figure-4. The other classes can as well be divided into different subclasses using the same technique.

One important thing should be noted. When we divide a class based on some feature, the subclasses are expected to be non-overlapping but it is not demanded that their union should be the divided class. The division is similar to the *specialization concept* used in the *Database.*

Every class in the hierarchy contains several objects. The hierarchy is incremental towards the root i.e., a class contains all the objects appearing in its descendent classes.

In order to reveal the importance of the class-hierarchy, it should be connected with the T-roles. For any T-role θ_i, there is a possibility *set* S_i whose elements can play the role of θ_i. As already discussed in Section 1, it is quite time consuming to get hold of the exact S_i. But the *linguist* has some intuitive idea about each S_i. and finds out that class C_i in the hierarchy which differs minimally from S_i and contains S_i(based on intuition, as he does not know exactly the elements of S_i). More generally, the expert can find out C_1, C_2, ..., C_k in the hierarchy so that C=union of C_i's , contains S_i and the difference (C-S_i) is minimum. For the sake of simplicity, let us assume that n is 1 i.e., the user has got a class C_i for each θ_i. Till further knowledge is not entered, C_i will be used as the possibility set of θ_i. On query θ_i, the system lists the elements of C_i.

Our aim is to proceed towards perfection i.e., making $C_i - S_i$ as small as possible. Suppose, the linguist feels that for some i, C_i is not a good approximation of S_i i.e., it includes too many objects which do not really belong to S_i. The system is capable of supporting the expert to correct such instances. The objects in C_i are listed by the system. At present each one of them can play the T-role θ_i. From this list, the linguist ticks the objects which he thinks to be correct choice for $\theta - i$. These ticked objects comprise one class $C_i^!$. The condition $C_i^! \subseteq C_i$ is obviously satisfied. While striking off the spurious objects the user must be very sure to choose only those that can never play the T-role under correction. Once some object is excluded from the possibility set of a θ-role, there is no way to recover it. If the expert commits error, some earlier instance of the database must be invoked. It is the responsibility of the user to maintain the relation $C_i^! \supseteq S_i$. Subject to this constraint, the aim is to make $C_i^!$ as small as possible. It is found out if some immediate subclass of C_i contains $C_i^!$. Suppose A is such a subclass. Then the system finds out if some subclass of A contains $C_i^!$ and so on. At some stage it will come to a set B which contains $C_i^!$ but none of whose subsets does so. Then $C_i^!$ is placed as a subclass of B as follows. Suppose B is divided into different subclasses depending on 3 features.

$B \xrightarrow{feature-1} B_{11}, B_{12}$

$B \xrightarrow{feature-2} B_{21}, B_{22}$

$B \xrightarrow{feature-3} B_{31}, B_{32}, B_{33}$

See Figure-6.

It is checked if any one of the conditions:

1. $B - (B_{11} \cup B_{12}) \supseteq C_i^!$
2. $B - (B_{21} \cup B_{22}) \supseteq C_i^!$
3. $B - (B_{31} \cup B_{32} \cup B_{33}) \supseteq C_i^!$

is satisfied.

Suppose the first condition is satisfied. Then C_i^l is placed as a subclass of B with respect to the first feature, as shown in Figure-7.

If none of the above three conditions is true then the system divides B with respect to another feature and C_i^l is the solitary subclass under that feature (Figure-8).

One preliminary class-hierarchy is created. Each θ-role can carry one or more of the classes as its possibility set. This initial classification is very important. Any further improvement will be based on this classification only and hence, if the initial one is chosen to be bad hierarchy then by no means can we expect very good result even after several corrections of theta roles.

3 Detailed Description

The classification hierarchy may be considered as a tree whose root is the class *Object-space*. Each vertex has some name e.g.*animate, inanimate, visible* etc. Each vertex in the hierarchy-tree has a path from the root. We will associate some unique names with these paths. The purpose is two-fold:

a) Given any two path-names it is possible to identify the relationship between the two classes with those paths from the root i.e., whether one is descendant of the other or whether they should be disjoint etc.

b) If B is a class with path name p and C is one of its descendants connected by link l, then the path name of C is pl i.e., l appended to p. Starting from the root one can recursively find out the path names for all the classes using this technique. Hence, if the system can efficiently search the records corresponding to classes with respect to their path names from the root, no pointer will be needed to represent the tree-structure.

Suppose C is a subclass of B. B is divided based on some features numbered 1, 2, 3, ... If C is the q-th subclass according to the p-th feature then the edge connecting B and C is numbered (p, q). From any class all the fan outs get distinct names(tuples).See the names of links in Figure-4. If the path p to class B consists of edges $e_1, e_2, \ldots, e_k$ with names $l_1, l_2, \ldots, l_k$ respectively, then the name of p is taken to be $l_1, l_2 \ldots l_k$. For computational ease, the level of hierarchy at which the class appears, is also kept alongwith the path information. Suppose the following conditions are satisfied.

1. There are at most f many discriminating features for a class
2. A feature can give no more than c subclasses
3. The height of the hierarchy-tree is at most h.

In this case, to store the path and level informations one requires $(\lceil \log h \rceil + h \lceil \log f \times c \rceil)$ many bits. The first component is for the depth information and the second one is for path code. For implementation, we took $f = 4, c = 4, h = 7$. So the bit requirement is:- $(\lceil \log 7 \rceil + 7 * \lceil \log(4 * 4) \rceil) = 31$ i.e. one 32 bit integer is sufficient.

Path information coding of two classes are given below. Each square stands for one nibble (4 bits). The first nibble keeps the level of the class in the hierarchy. The other seven squares are for storing the names of the links required to reach a class starting from the root.

3	l_1	l_2	l_3	0	0	0	0

5	l_1	l_2	l_3	l_4	l_5	0	0

So far, objects have had nothing to do with the hierarchy tree. Now they will be connected together. The object names are inserted in a direct-access file. The position of an object in the file is unique. These positions(integers) are used to represent objects in the classes of hierarchy. For that purpose, another intermediate file is used. See Figure-13.

The intermediate file is divided into blocks of equal size. When the class-hierarchy is created then one block is allotted to each class and the record corresponding to each class points to the first position of that block. This first position keeps count of objects included to the class and is initialized to 0.

Each time a new object is taken in B, the counter is increased by one. When the allotted block becomes full, new block (which has not been allotted to any class so far) is taken for B and a pointer is placed at the last position of the filled up block to the new block. New blocks are consumed one by one and they are all linked in a linear chain.

Whenever some object is inserted, its position i (integer) in the file is known. This integer index i works as the *representative* of the object when it is intended to be included in some class. The user is asked about which all classes include it. In response to the question, a number of classes $B_1, B_2 \ldots, B_k$ are mentioned. The path names $p_1, p_2, \ldots p_k$ corresponding to $B_1, B_2, \ldots, B_k$ respectively are found out. There is one procedure called *Relationship* which accepts two path names and works out the relation between those two classes. If any two classes among $B_1, B_2, \ldots, B_k$ are to be disjoint, the object cannot be included to both of these conflicting classes and some measure is taken under such circumstances.

If there is no conflict, the object is included to each class. The method of inclusion is illustrated with class B_1. The object counter of B_1 (content of the first position of the first allotted block to B_1) is increased by one. Then the object-index i is inserted in the first empty position of the last allocated block of B. For this purpose one may need to traverse several links. The same process is followed to include the object to classes $B_2, B_3, \ldots, B_k$.

One speciality of the hierarchy is that every class contains all its descendants. But it is not demanded on part of the user that whenever an object is included in class A, the same object should also be included in all the ancestors of A. The content of a class A is found out by taking union of objects which are directly included to A and those included to some descendent of A. A recursive *depth first search* procedure is used for that purpose.

There is one indexed file (with respect to verb name) for the verbs. The record contains different θ roles of the verb. For each θ role there are several classes in the hierarchy which can play the role of that theta role in some sentence. The possibility set for a theta role is the union of such classes.

In many occasions sets of objects are talked about and the set operations are required for that purpose. Bit representation of sets has been used. For every object one particular bit is allotted. A set is represented by turning on the bits corresponding to objects in S. Some of the used set operations and their corresponding operations in our representation are given below:

Union: Bitwise OR

Intersection: Bitwise AND

Inclusion of an object to a set: Turning on the bit corresponding to the object included.

Exclusion of an object from a set: Turning off the bit corresponding to the object excluded.

The other operations are also done similarly.

Initially a classification is done and objects are included to different classes one by one. All the objects need not be inserted at one stage. Along with each verb, its θ-roles are also specified. One or more classes are mentioned whose objects can play that θ-role. The number and names of classes usually differ from one θ-role to another. The union of such classes for a particular θ-role will provide the objects which can play that role. New verbs can be accommodated any time.

If the *linguist* feels that there are too many spurious objects in the possibility set of some θ-role, the system is able to support him to make correction. In response to a user supplied verb-name, the possibility set of each θ-role is corrected in turn.

For the sake of definiteness, assume that the θ-role θ of verb v (i.e., some T-role θ) has three classes C_1, C_2, C_3. Each element of C_1 is listed and the user is asked to strike off an entry if in his opinion it appears definitely spurious for θ. From this, the system creates a subset C_1' of C_1 and according to the expert, this new subset is the possibility set of θ. Similarly C_2' and C_3' are produced. C_1' , C_2' , C_3' replace C_1 , C_2 , C_3 respectively as possible classes for θ. As the relation between θ-roles and objects is through the classification-hierarchy, these new sets of objects must be absorbed there.

If C_1' is already present in the hierarchy or it is null, no action is needed. Otherwise the system has to put it in the hierarchy following the method discussed in Section 2. Starting from C_1, a *depth first search* is done to find out the class B, which contains C_1' but none of whose subclasses does so. C_1' is placed as a subclass of B as discussed in Section 2.

The database also provides the following facilities:

1. Class-hierarchy creation.
2. Object-inclusion.
3. Thematic-roles entry.
4. Accumulating a subset of a class in the hierarchy.
5. Theta-role possibility set correction.
6. Display of objects included in a class.
7. Display of possibility sets of theta-roles.
8. Display of hierarchy.
9. Storing the files in the back-storage. Several versions are kept. If the user feels that some error has been committed, any earlier version can be invoked.

4 An Example Session with the User

One initial classification given by Figure-1 is created. Several objects are included to different classes as given below.

object	name of classes
cat	animal kingdom, visible
dog	animal kingdom, visible
cow	animal kingdom, visible
Ramesh	animal kingdom, visible
Punam	animal kingdom, visible
watch	inanimate, visible
water	inanimate, visible
paper	inanimate, visible
biscuit	inanimate, visible
weather	object-space
problem	object-space
idea	object-space
spoon	inanimate, visible
virus	invisible
bacteria	animate, visible
atom	inanimate, invisible
apple	plant-kingdom, visible
mango	plant-kingdom, visible
rose	plant-kingdom, visible
lily	plant-kingdom, visible

As an effect the content of the classes become:

1)	object-space:	all the objects, as it is in the top of hierarchy.
2)	animal-kingdom:	cat, dog, cow, Ramesh, Punam, bacteria.
3)	plant-kingdom:	rose, mango, apple, lily.
4)	inanimate:	watch, water, paper, biscuit, spoon, atom.
5)	visible:	cat, dog, cow, Ramesh, Punam, bacteria, rose, mango, apple, lily, watch, water, paper, biscuit, spoon.
6)	invisible:	virus, bacteria, atom.

Note that, *virus* is not included to any of the classes- *plant kingdom, animal kingdom, inanimate.*

Suppose the following four verbs are inserted alongwith their *theta*-roles and the possibility sets for each *theta*-role.

1. Eat:

Agent:	animal-kingdom
Theme:	object-space

2. Speak:

Agent:	animal-kingdom

3. Give:

donor:	animal-kingdom
recipient:	animal-kingdom
Theme:	object-space

4. Bite:

Agent:	animal-kingdom
Theme:	object-space

According to the database only animates i.e., *cat, dog, cow, Ramesh* and *Punam* can eat. The possibility class of *theme of eat* is whole *object-space.* Thus, so far everything can be eaten including *watch, atom, virus etc..* When the user wants to correct it, all the elements in *object-space* are listed in turn and the user strikes off the objects which he thinks cannot be eaten, like *watch, atom etc.*. Possibly the user will keep ***biscuit, cow, apple, mango*** as the set of objects which can be eaten. After correction, the verb ***eat*** becomes,

Eat:
 Agent: animal-kingdom
 Theme: edible

The set *edible* appears as a subclass of the class *visible*, as shown in Figure-9.

The correction can be done in any order. Here it is done according to the order of appearance. So, next comes the verb *Speak*. At present ***cow, dog, cat, Ramesh and Punam*** - can ***speak***. Obviously ***cow, cat and dog*** are to be eliminated. A subclass ***human*** is created and placed in the hierarchy as in Figure-10. The θ-role is changed to:-

Speak:
agent: human

The verb *Give* has three θ-roles. So far the whole object- space is mentioned under ***donor*** θ-role of this verb. After rectification it becomes ***human***. Same thing happens for ***recipient*** of the verb. Thus no change of the hierarchy is required. While correcting ***theme*** of the verb, the database acquires a subset including everything, except :- ***Ramesh, Punam, weather, virus, bacteria, atom***. This requires the ***object- space*** to be divided using a new feature, under which only one class, ***transferable*** say, is there. It has been shown in Figure-12. Thus the changed possibility sets will look like:-

Give:
 donor: human
 recipient: human
 Theme: transferable

If *agent* of the verb *bite* is corrected then only ***dog and cat*** will be there. Call this class *beast*. After its inclusion, the hierarchy is shown in Figure-11.

5 Integration with an NLP System

Let us try to explain the utility of the database with the help of one example. e.g. *Ram amake ekta boi debe* has the same meaning as *ekta boi Ram amake debe.*

Theta roles : p_1=agent, p_2=recipient, p_3=theme
Objects : o_1=Ram, o_2=ekta boi, o_3=amake

When we try to analyze it syntactically, some ambiguity is found. There are more than one objects competing for the theta role *agent of debe* (and also for *theme of debe*). Such conflicts must be resolved in order to interpret the sentence. The parser helps us narrow down the possibility of different objects playing different θ-role. There are three objects in the sentence. After parsing some of them remain as valid competitors for θ_1; Call this set of objects S_1. Similarly S_2 and S_3 are also formed.

$S_1 = \{o_1, o_2\}, S_2 = \{o_3\}, S_3 = \{o_1, o_2\}.$
In order to interpret this sentence, it is required to reduce these sets to singletons.

When some human being is faced with the above sentence then the ambiguity is resolved by recognizing the fact that *ekta boi* is inanimate and hence can never be the agent of the verb *debe*. Thus S_1 reduces to $\{o_1\}$. Once the two matchings - (p_1, o_1) and (p_2, o_3) are made, the other matching viz. (p_3, o_2) necessarily follows.

If we want the machine, to resolve the ambiguity in the same way as discussed above, then one **database** keeping the possibility sets of each θ-role, is needed. Then the machine can also get the information that *ekta boi* is *inanimate* and so can never be the *agent of debe.*

In general there may be n-many theta roles p_1, p_2, ..., p_n of a verb. o_1, o_2, ..o_n - n many objects play different theta roles of that verb in some sentence. One has to detect which role is played by which object in the sentence. S_1, S_2, ..., S_n are subsets of the objects present in the sentence i.e. o_1, o_2, ..o_n . After syntactic analysis each element of S_i is vying for the theta role p_i. It is checked which all elements of S_i can never play the role of p_i and such elements are removed from S_i. This process reduces each S_i to some U_i. U_i now represents the possibility set for p_i. One bipartite graph emerges out, with vertices $p_1, p_2, \ldots, p_n, o_1, o_2, \ldots, o_n$ and edges (p_i, o_j) : o_j is present in U_i . Each p_i has to be matched with exactly one o_j which is adjacent to p_i.

If only one matching is possible then the sentence is unambiguous and can be interpreted otherwise it is ambiguous. The number of different possible matchings can be taken as a measure of ambiguity of the sentence. There are many efficient algorithms available in the literature to find out the different possible matchings; those can be employed for this purpose.

While shrinking the set S_i to U_i we need to know which all objects can never play the role of p_i in any sentence. For that purpose, the **database**, as described in Section 1, is used.

References

[1] Allen J. 1987 *Natural Language Understanding*, Benajamin Cummings: Menlo Park, CA, 1987.

[2] Bandyopadhyay D. (in Press, 1992) The Valency of Bangla Verb and Some related Issues. *Monograph of Institute of Linguistic Sciences*, India. Calcutta.

[3] Michalski R.S., Carbonell J.G., and Mitchell T.M.; *Machine Learning: An Artificial Intelligence Approach*

[4] Sengupta P.; Chaudhuri B. B. 1989 A Morphological Verb and Case Analyzer for NLP of a Major Indian Language. In: *Proc. National Symposium for Natural Language Processing*. Vishakapatnam, India.

[5] Sengupta P. and Chaudhuri B. B. 1991 On Parsing of a Class of "Free Word-Order" Languages. *(Communicated)*.

[6] Sengupta P. and Chaudhuri B. B. 1992 A Morpho-Syntactic Analysis Based Lexical Sub-System. In: *CPAL-2, Second Regional Workshop on Computer Processing of Asian Languages*. Kanpur, India.

A Lists of Verbs and Objects Included in the System

A.1 List of Objects

aJ, aKas, aLU aLU, aMOD, ANANDA, aNGUL, ANNA, ANUbUTI ANUS-taN, ANYaY, APaRtIBA, ApIS, AsRU, ASUK, aTa, baB, BaBa, BADAN, BAdU, Bag, bAGABaN, bAGINI, baGNE, BAI, baI, baJa, BaJaR, BaKSA, bAKTI, BaLI, baLOBaSa, BAN, BANCANa, BANGA, BanI, BaNLa, BaRaNDa, BarI, BaSAN, BASTRA, baT, baTa, BaTaS, BaTI, BaTI, BAU, BAUDI, BEGUN, BELa, BEraL, BAI, BIBaHA, BIcaNa, BIHaR, BIJNYaN, BIKaL, BINa, BON, bRaTa, BRIsTI, BUK, BuK, bUT, cABI, CACCARI, CaKAR, CaMrA, CaN, caNa, CANDRA, CAraI, CaT, CaTNI, CEaR, cELE, CETANa, COk, cOLa, CO, DAI, DaL, dAM, daMa, dAN, daRa, DARJa, dARMA, DaT, DAYa, DEaL, DEHA, DIM, DIN, DIN, DOKaN, dOPa, DUd, DUk, duLa, DUNIYa, DUR, Gada, GALa, GALPA, GaN, GANGa, gAR, GarI, gArI, GARU, GAURAB, GELaS, GIRI, GOaLa, gOra, GRIHA, GUHa, HALUD, HarI, HaS, HaT, HaTI, HaTU, HIYa, HRIDAY, IDUR, IT, JAL, JAL, JaMa, JaMa, JAMUNa, JaNaLa, JaNLa, JANMA, JANTRA, JANTU, JaTI, jI, JINIS, JNyaN, JOG jOL, JUG, JUTO, kABAR, KABITa, kaDYA, KaGaJ, KaGAJ, KaJ, KaK, KaL, KALAM, KaLI, KaLO, KALSI, KaMaR, KaN, KANYa, KaPAr, KARaT, KARMA, KARUNa, kaT, KATA, KAta, KaYa, kELa, kELa, KLaNTI, KOMA, KUKUR, KUMAR, LaL, LatI, LEka, LOK, Ma, Mac, MAILa, MAJa, MAMATa, MAN, MaNAB, MaNSA, MaNUs, MaTa, MaYa, MAYDa, MEg, MEYE, MISTI, MItYa, MRITYU, MUk, MULO, MURGI, NAD, NADI, NaK, NaLa, NaM, NaPIT, NaRI, NIJ, NJaY, NONRa, NUPUR, Pa, Paca, PADMa, PaHar, PAISa, PAKET, PakI, pAL, PaN, Para, PaRaBa, PARBAT, PARISKaR, PARsU, PaRtIBA, PAsU, PAt, PAt, PATAL, PatA, PATI, PATNI, PATRA, PaYRa, PET, PETNI, PITa, PITasRI, POsaK, PRAHaR, PRAJa, PRAJaPATI, PRAMOD, PRASaD, PRATIJOGITa, PRA-TIMa, PREM, PRITI, PUKUR, PUTRA, RaJa, RAKTA, RaM, RaT, Ra-TRI, ROUDRA, RUPa, RUPA, RUTI, SaBaN, SABUJ, SaDa, SadANa, SadU, SaGAR, SAKaL, sAKTI, SAMaJ, SAMRaT, SAMUDRA, SANBaD, SAN-BaDPATRA, SANdYa, SANSKRITI, sARI, SATYA, SBaMI, SBAPNA, SHYaM, sILPI, SInHA, SIRI, SNEHA, SONa, STRI, SUk, SUNDA, SUNDARI, SURJ, TABLa, TaKa, taKUR, taLa, TaNPURa, TaRa, TIa, TIL, UPABAN, UPADEs, UPANYaS,

A.2 List of Verbs

UPCONO, gOTa, JOMa, BaJa, paTa, GALa, jara, NEba, pOTa, pOLa, RaGa, janPaNO, LapaNO, cOTa, DOUrONO, HaSa, Kasa, KanDa, gU-MONO, MORa, HaRa, DOBa, KanPa, NEOa, sONa, kaOa, GELa, sEka, LOTa, CaOa, BOja, Banda, canKa, canTa, baJa, TaNa, daRa, daKa, KU-TONO, cOLa, CaBKaNO, MaRa, anTa, aTKaNO, LEka,

B Current Status of the System Knowledge-Base

B.1 List of Thematic Roles Considered

AGENT (Ag): Who performs the action.

PATIENT (Pt): Who or what gets affected as a result of the action.

THEME (Th): What was intended in the action. Also called GOAL.

FROM-PLACE (FrP): The place from where the action was initiated. Also called SOURCE.

AT-PLACE (AtP): The place where the action was performed. Also called LOCATION.

TO-PLACE (ToP): The place to which the effect of the action will proceed. Also called DESTINATION.

B.2 Table of Different Classes

B.3 Table of Verbs, Their θ-Roles and Classes They Point To

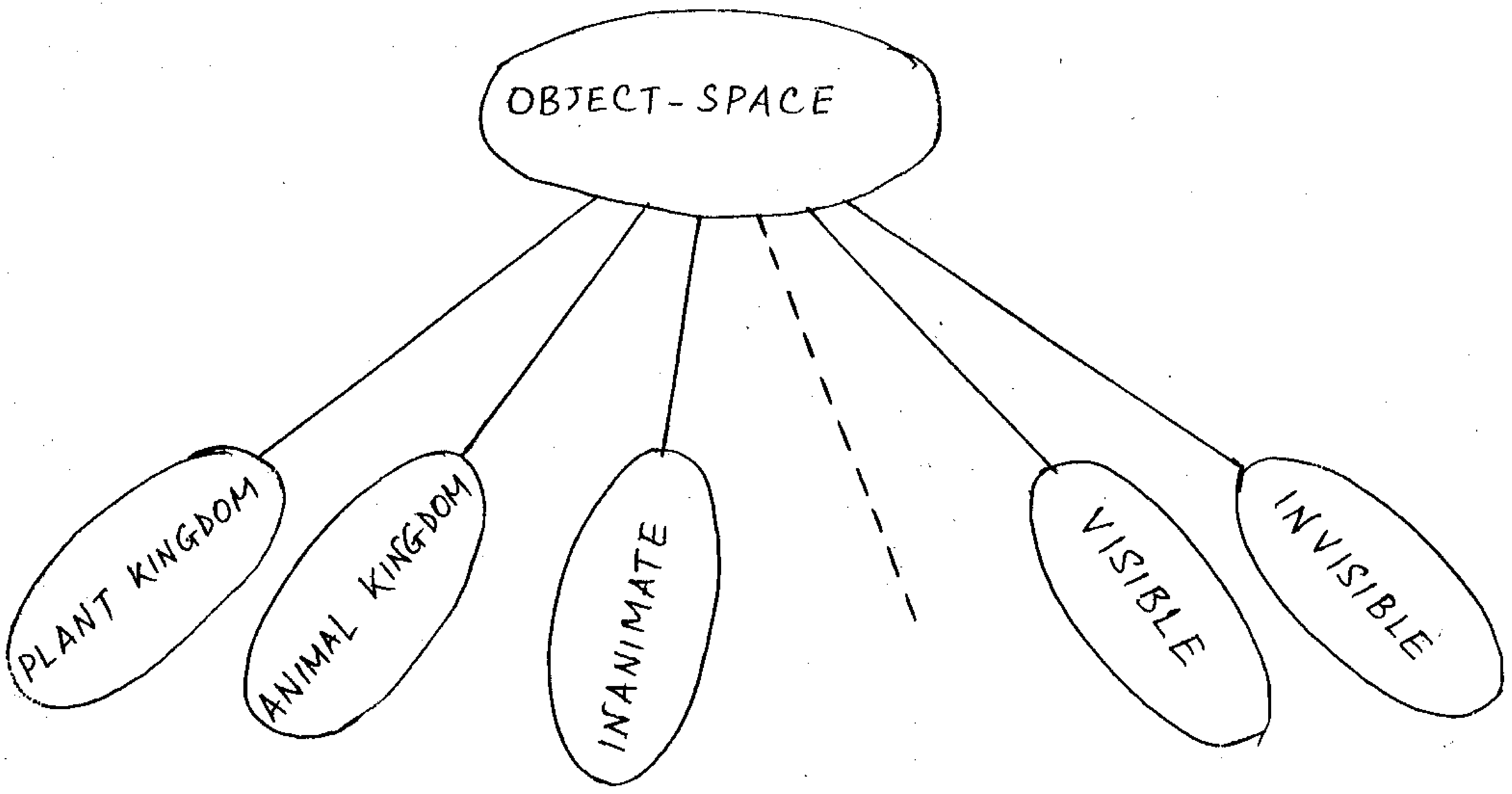

FIGURE-1

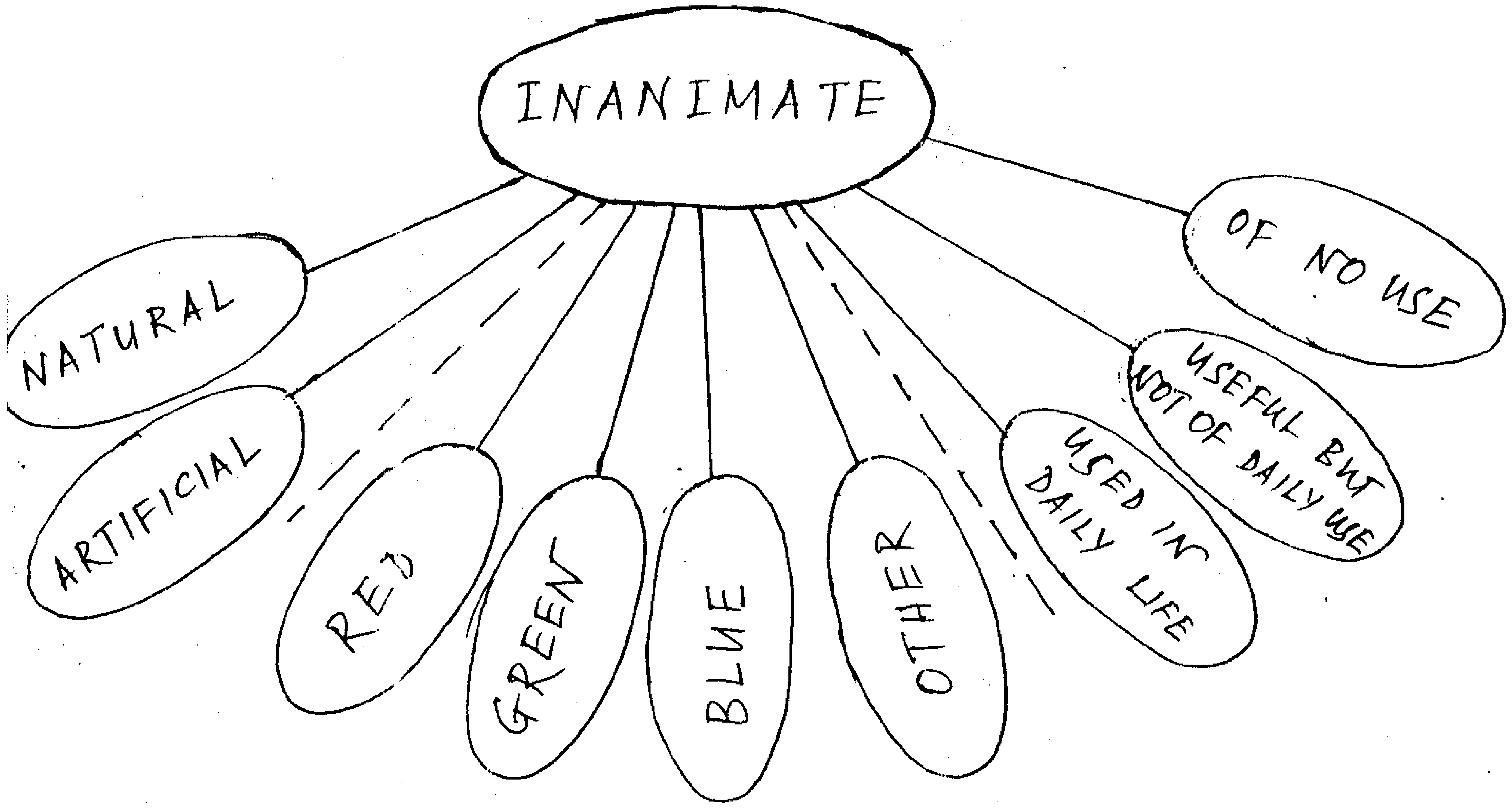

FIGURE-2

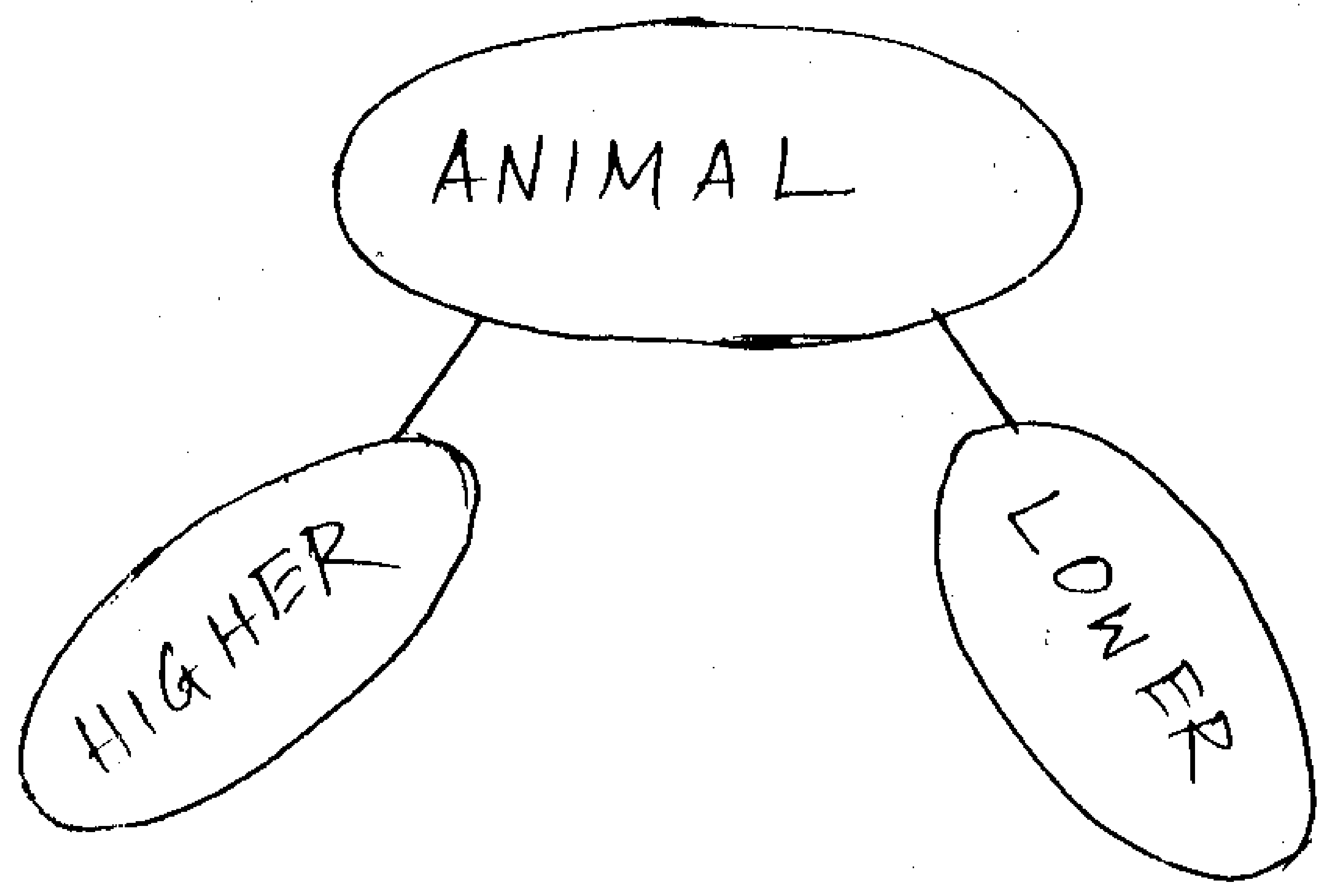

FIGURE- 3

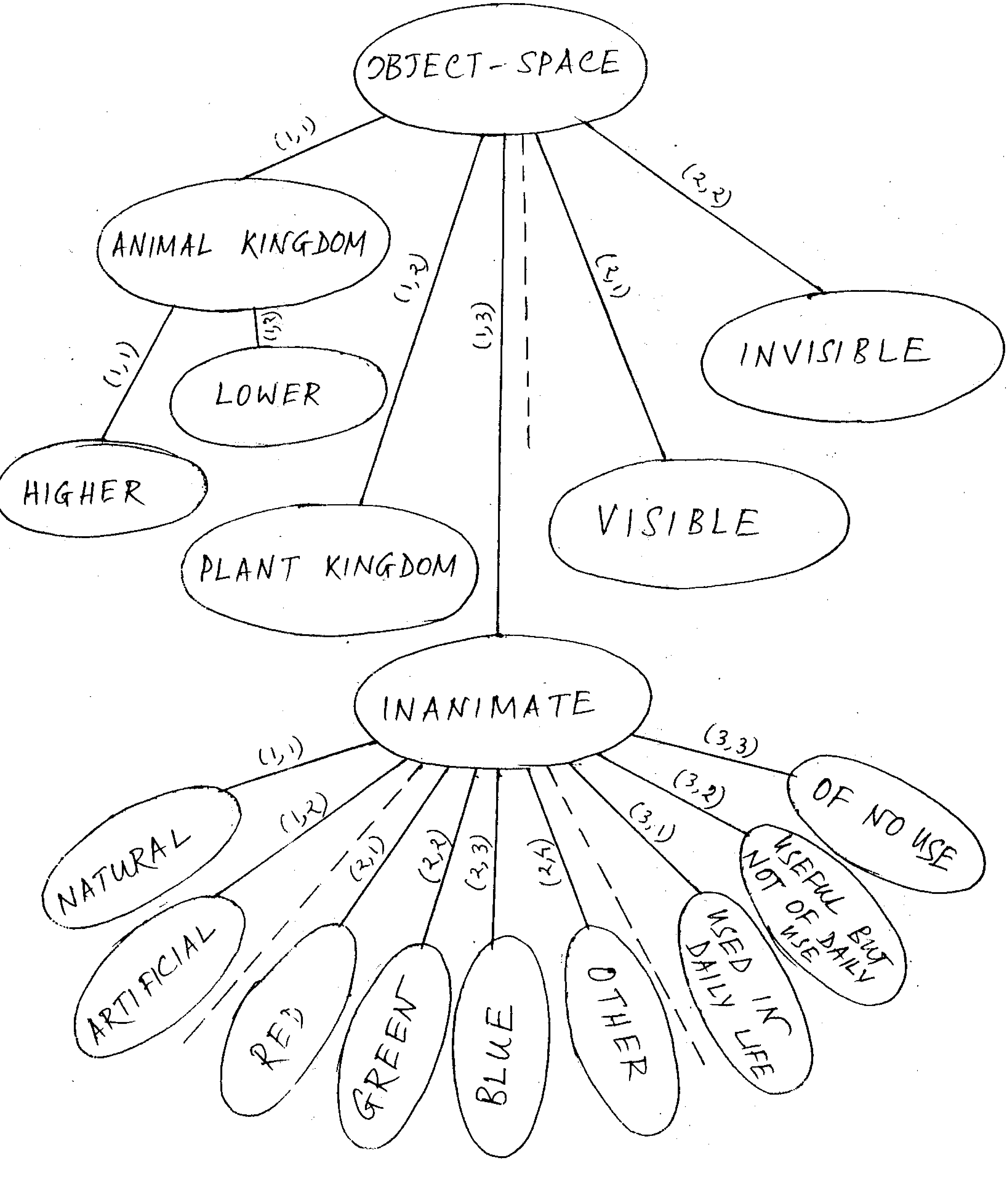
OBJECT-SPACE
(1,1)
(1,2)
(1,3)
(2,1)
(2,2)
ANIMAL KINGDOM
(1,1)
(1,2)
HIGHER
LOWER
PLANT KINGDOM
INANIMATE
VISIBLE
INVISIBLE
(1,1)
(1,2)
(2,1)
(2,2)
(2,3)
(2,4)
(3,1)
(3,2)
(3,3)
NATURAL
ARTIFICIAL
RED
GREEN
BLUE
OTHER
USED IN DAILY LIFE
USEFUL BUT NOT OF DAILY USE
OF NO USE

FIGURE 4

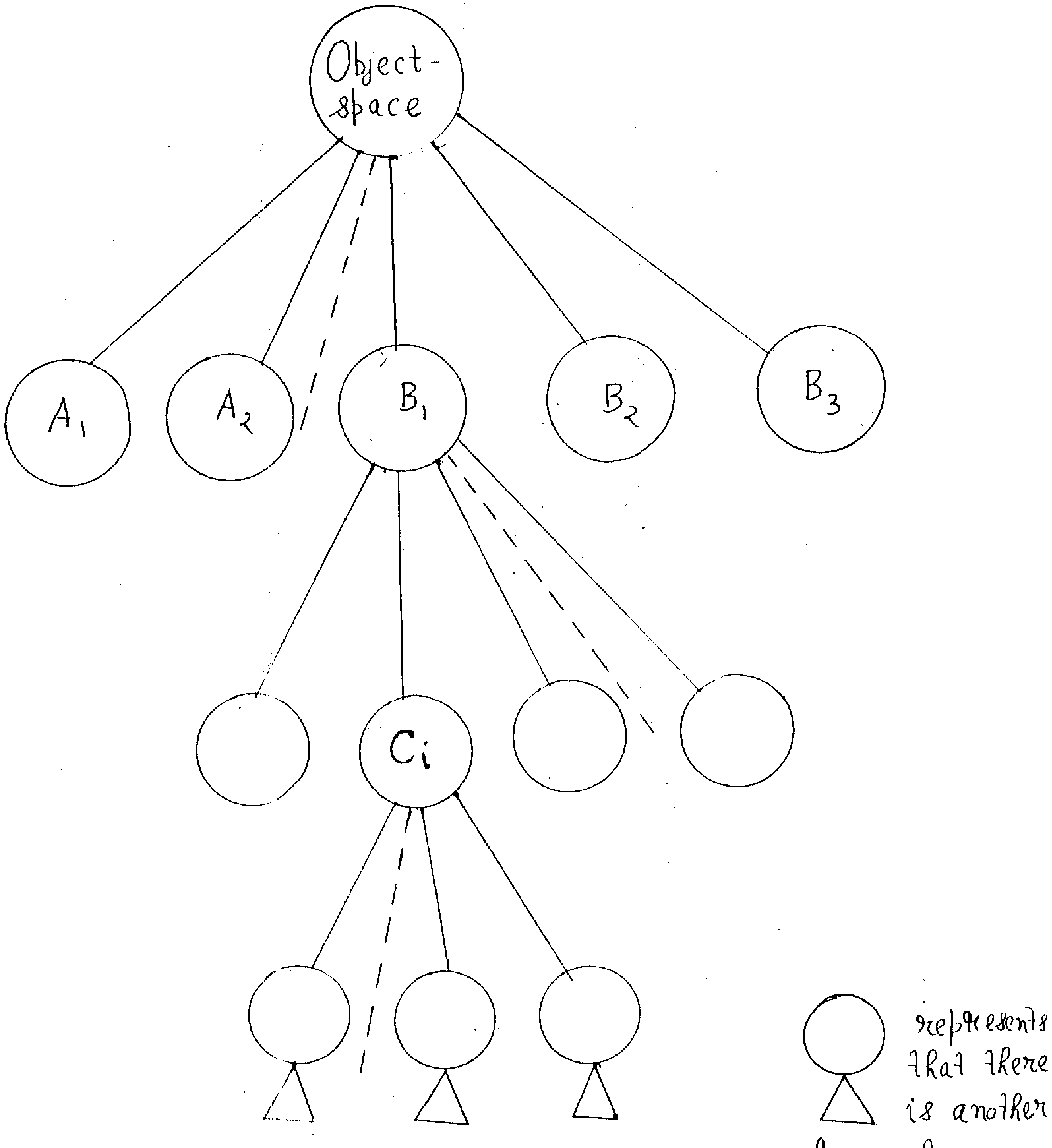
Object-space
A_1
A_2
B_1
B_2
B_3
C_i
represents that there is another hierarchy starting at this node.

FIGURE-5

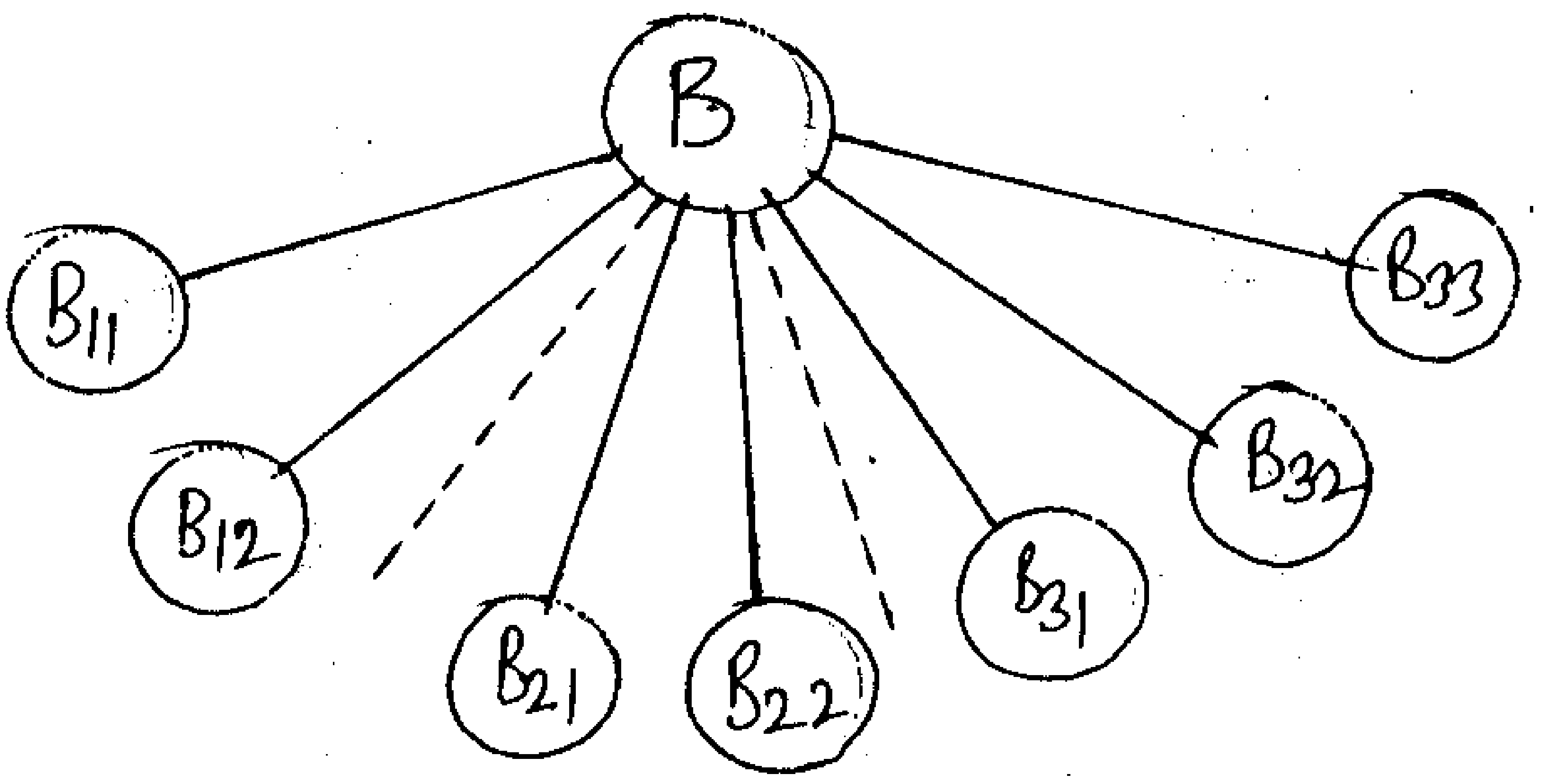

FIGURE-6

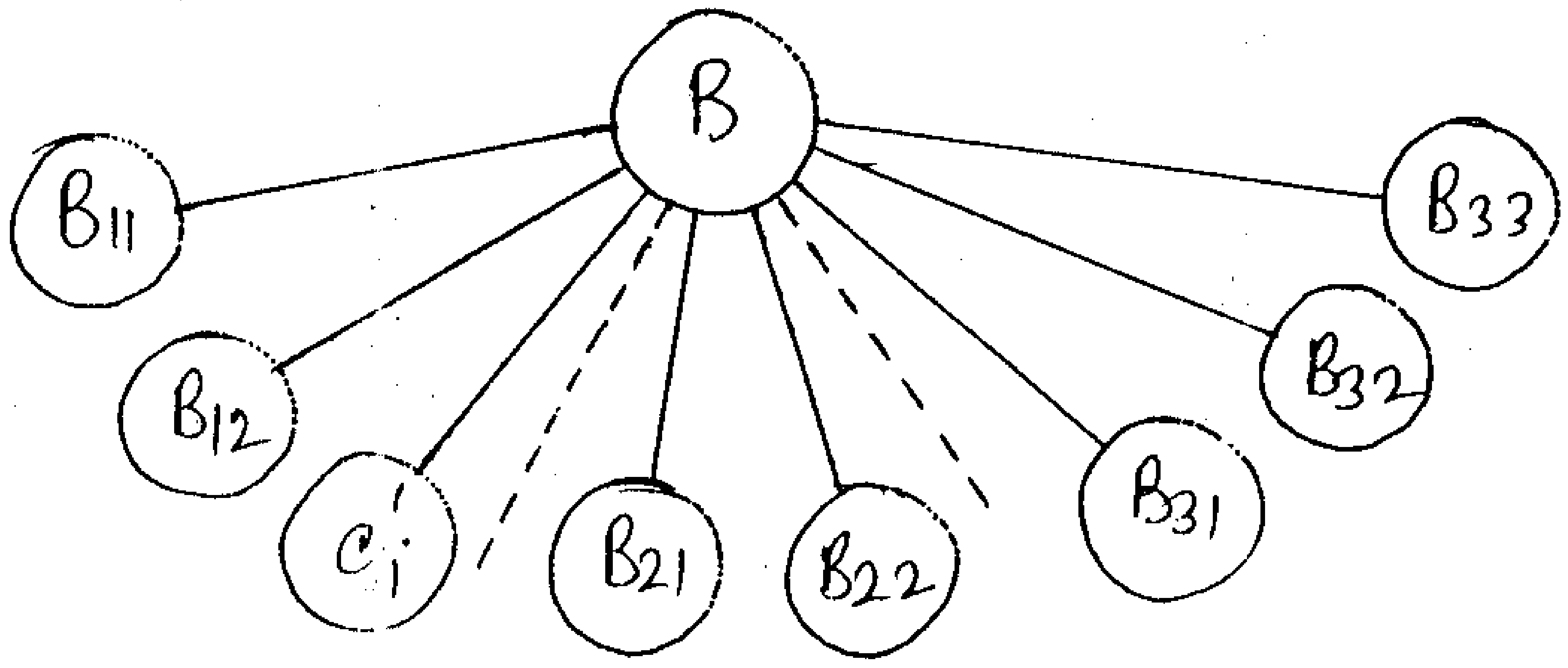

FIGURE-7

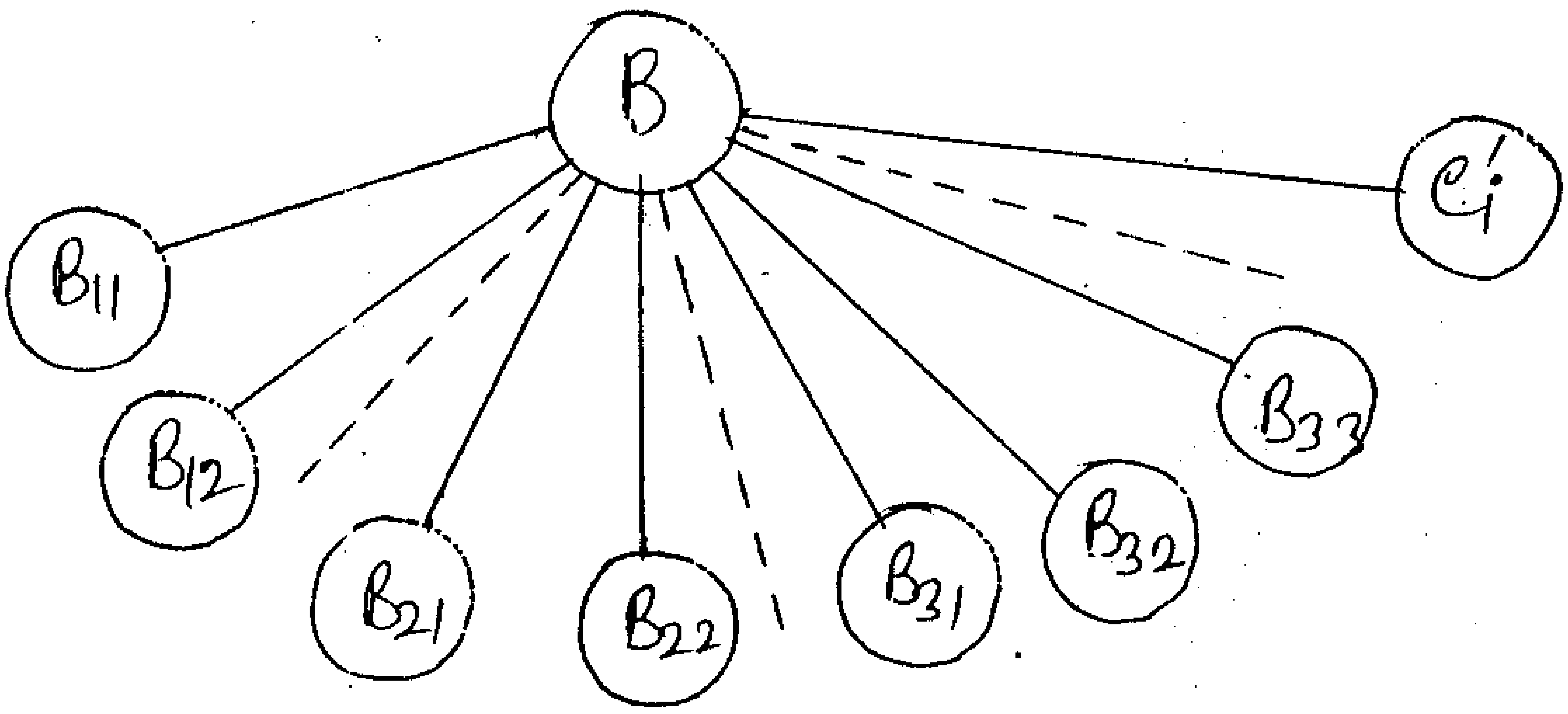

FIGURE-8

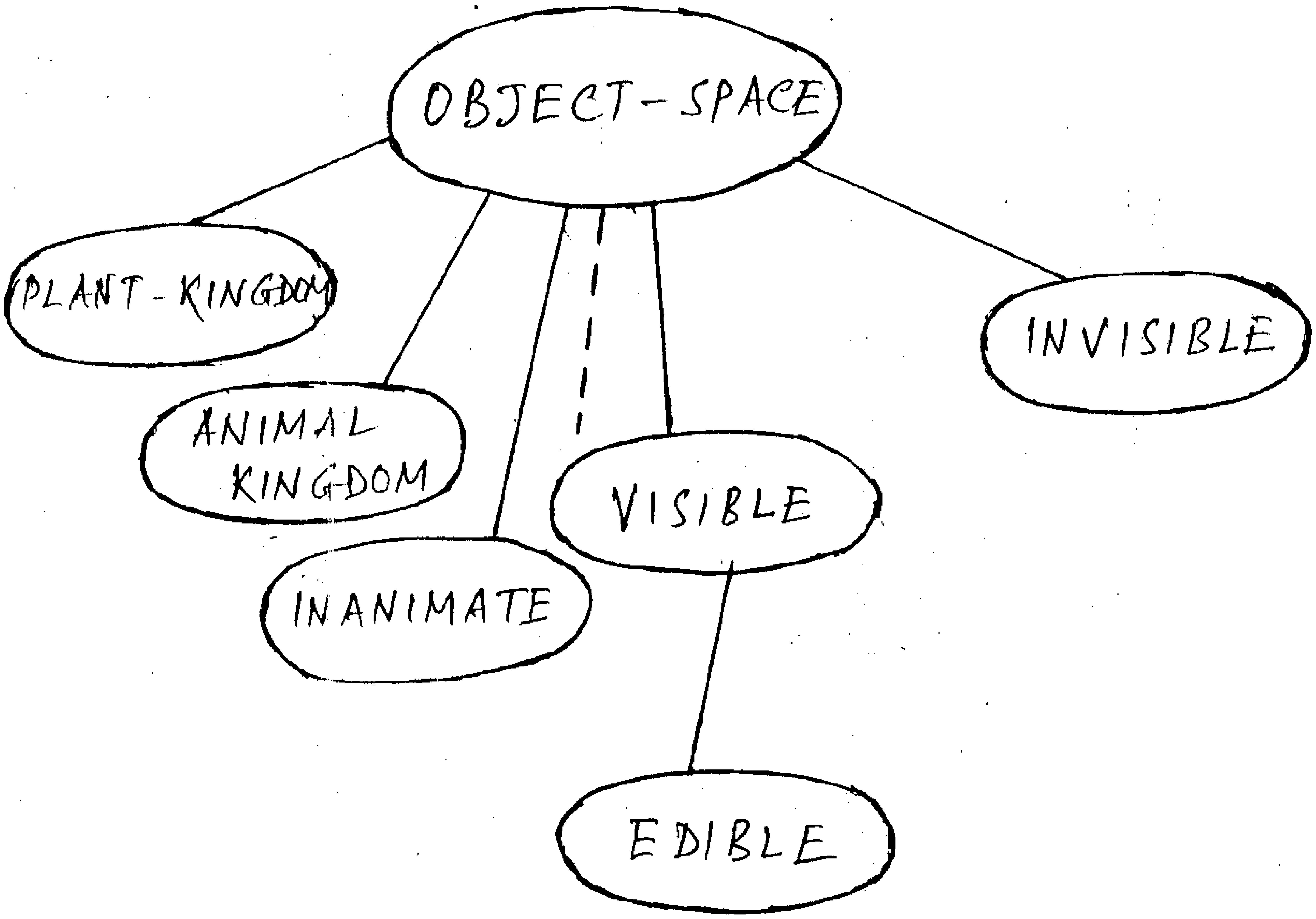

FIGURE-9

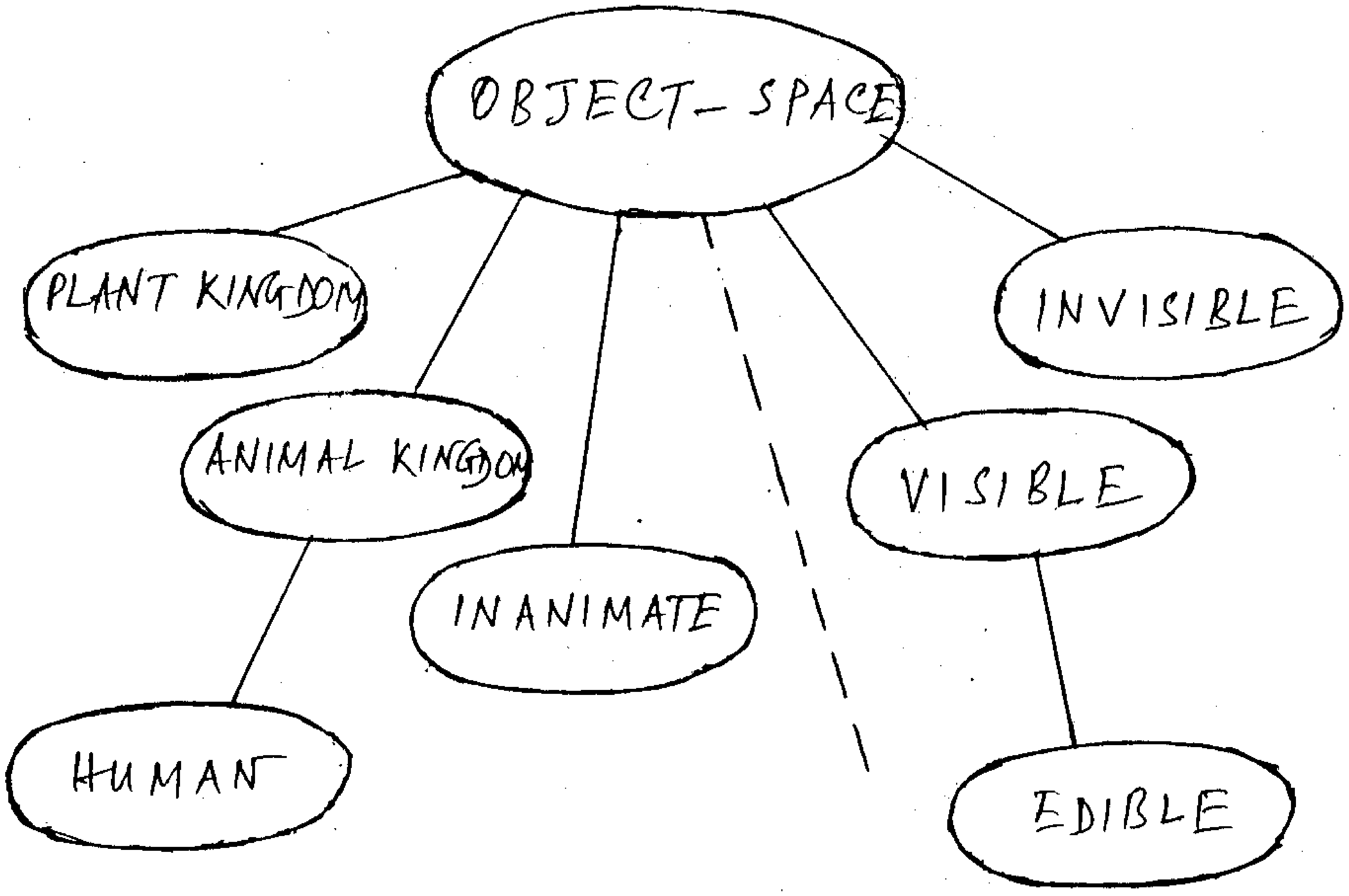

FIGURE 10

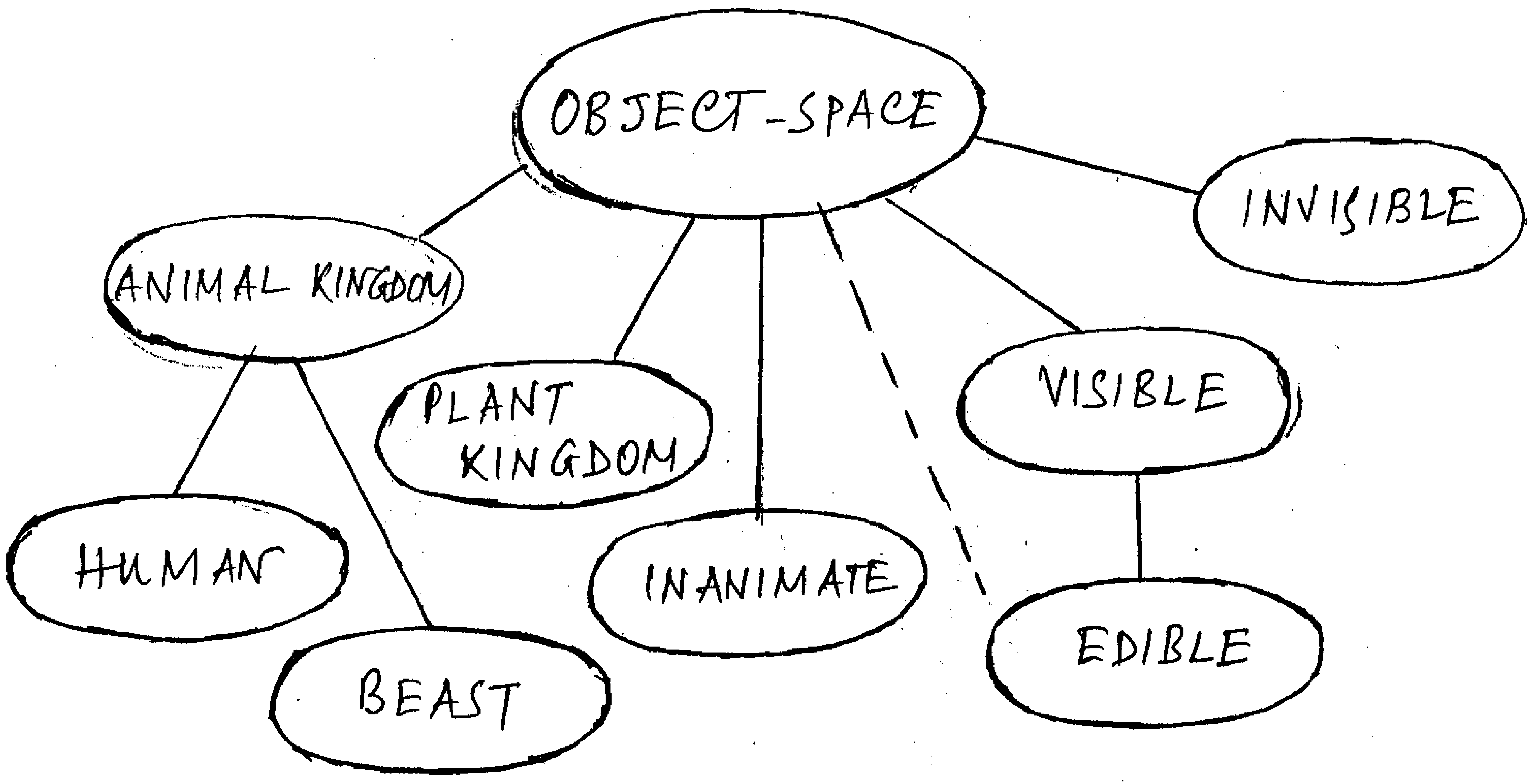

FIGURE- 11

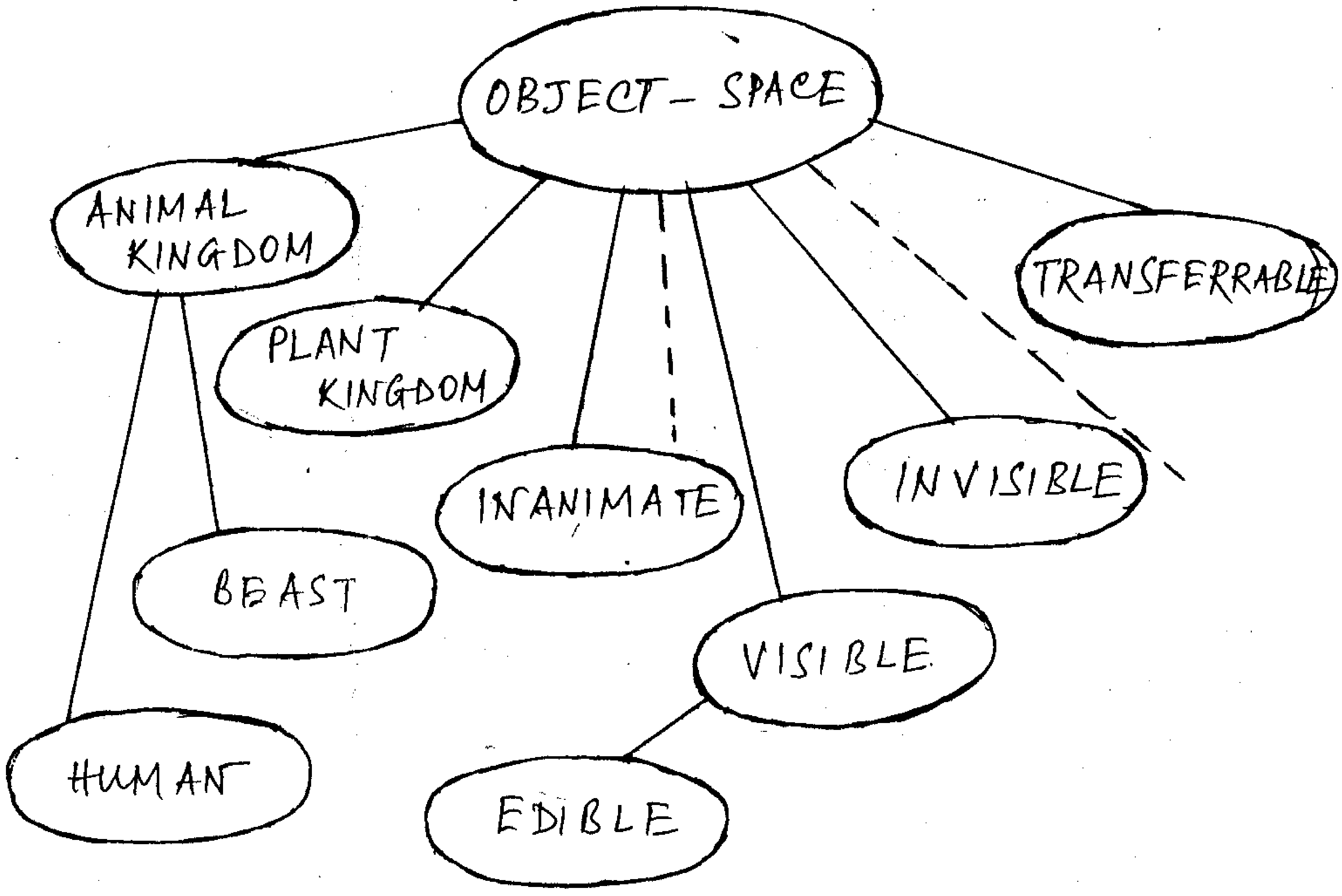

FIGURE 12