AN EXPERT SYSTEM FOR THE QUALITATIVE ANALYSIS IN CHEMISTRY

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements
for the Award of the Degree of
MASTER OF PHILOSOPHY
(COMPUTER SCIENCE)

K. SITARAMA RAO

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

1987

CERTIFICATE

This work embodied in the dissertation titled, " An Expert System for the Qualitative Analysis in Chemistry ", has been carried out by Mr.K.SITARAMA RAO, a bonafide student of School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi - 67.

This work is original and has not been submitted for any other degree or diploma of any other University.

Sri.C.RAVI SHANKAR
Systems Specialist
R & D Centre,
CMC Ltd.
115 SD Road
SECUNDERABAD - 3

Dr.K.K.BHARADWAJ
Associate Professor
School of Computer &
Systems Sciences
Jawaharlal Nehru University
NEW DELHI - 67

Prof.K.K.NAMBIAR
Dean, School of Computer &
Systems Sciences
Jawaharlal Nehru University
NEW DELHI - 67

0. Acknowledgements

1. Introduction to Expert Systems

   1.1 What are Expert Systems?
   1.2 Architecture of Expert Systems
   1.3 Alternative Architectures
   1.4 AI languages for implementing expert systems
   1.5 Some advantages of Expert System Technology

2. Introduction to the project

   2.1 What this is about?
   2.2 What this project can do?
   2.3 Some representational details

3. Inferencing

   3.1 Preliminary Tests
   3.2 Confirmatory Tests

4. Justification

5. Some comments on likely improvements

6. Listings of the program

7. Appendices

   7.1 Appendix A
   7.2 Appendix B
   7.3 Appendix C
   7.4 Appendix D

ACKNOWLEDGEMENTS

This project would not have happened without the help of many people, for all of whom I am deeply indebted. First I would like to experess my sincere thanks to my supervisor Dr K K Bharadwaj, for his understanding and encouragement throughout this period. I would also like to acknowledge Sri Kapoor, Systems Manager, CMC Secunderabad for allowing me to work at CMC, and providing me with excellent working facilities. Words reflect poorly my gratitude to Sri C Ravi Shanker for his immense patience and constant guidance during my stay at CMC Secunderabad. I am thankful to Mr U Bhaskar and Mr Sethuraman of CMC, for their invaluable suggestions. My acknowledgements would not be complete without thanking Prof P C P Bhatt, Head of Computer Science Department, IIT Delhi and Prof K K Nambiar, Dean School of Computer & Systems Science, JNU, New Delhi for the interest they had taken in guiding me to CMC Secunderabad. Lastly but not leastly, I thank Mr Meshack Ponraj for his timely help in printing this thesis.

INTRODUCTION TO EXPERT SYSTEMS

1.1 WHAT ARE EXPERT SYSTEMS?

Expert systems are problem solving programs which behave like human experts in specific domains. Like human experts they are capable of advising, diagnosing,justifying and learning. Some examples of human experts are a DOCTOR who diagnoses the disease (or diseases) in a patient and gives a therapeutic advice, a COMPUTER EXPERT who can advise a client on the configuration of a computer system depending on the requirements of the client, a CHEMIST who can advise a student on the qualitative analysis of a compound etc..

In each of the above examples, a substantial problem is being solved that requires special knowledge pertaining to the problem domain. The expert must garner relevant details of the problem concerned and apply the special knowledge in a selective manner to arrive at one or more solutions. If the details are incomplete, the expert should still solve the problem partially or go about desigining experiments by which the missing information can be obtained. Normally, even if a solution has been found, the task of the expert is not over as the expert is expected to explain and defend his suggestion. Besides all this the the expert must be capable of acquiring more knowledge.

Comparing with a human expert, an expert system should possess the following characteristics, in addition to its problem solving ability:

* Engage in a dialogue with the user to acquire the relevant details of the problems.

* Be able to explain its problem solving process.

* Be able to take care of new discoveries or lacunae in the domain either by experience or through a dialogue.

* Be capable of dealing with partial information.

1.2 ARCHITECTURE OF EXPERT SYSTEMS

In the last decade, research on expert systems [stefik 82,Davis 81], found that trying to build an Expert System in a procedural manner makes the program rigid. It has been found that such procedural programs cannot provide a flexible dialogue, cannot deal with partial information and are not easy to change.

The most important lesson that has been learnt is that knowledge about the domain of theproblem must be separate from how the knowledge is to be applied or used [Sangal 85,Davis 82]. The knowledge should be represented declaratively, and a separate interpretive component should select and apply it. It has also been found that the knowledge of the domain can be expressed naturally in the form of if-then rules. The organization that has emerged most popular is called RULE-BASED system as depicted in figure 1.1 below:

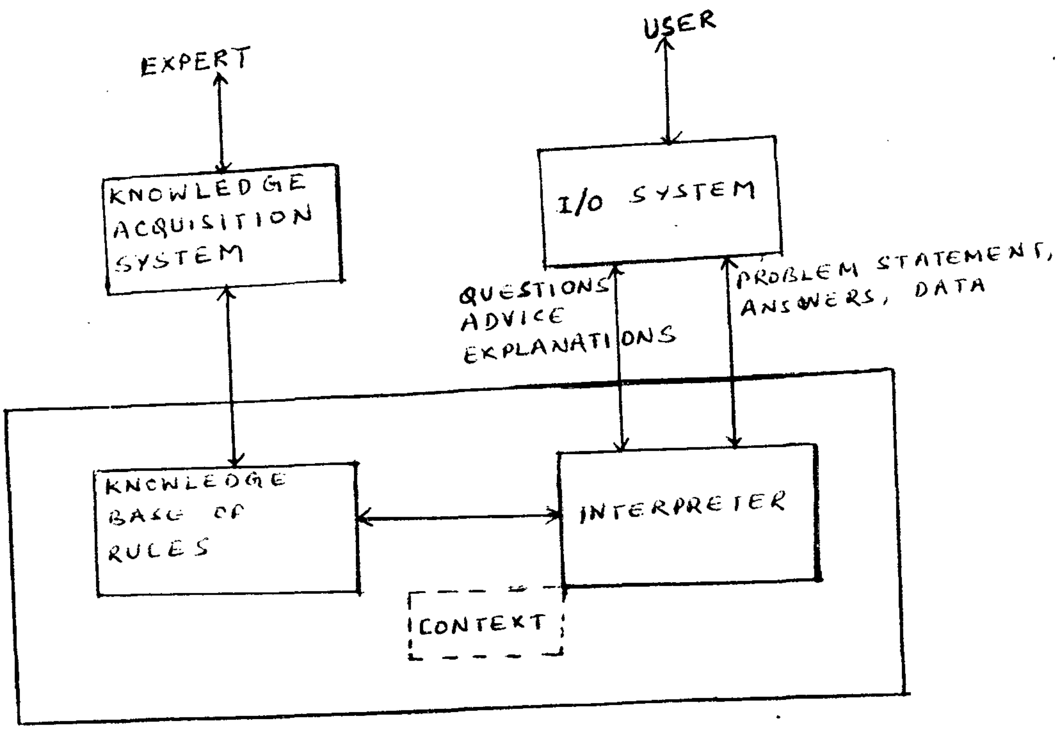


Fig 1.1

A rule based system consists of three major components:

1. a knowledge base consisting of if-then rules (also called productions).

2. a current context or facts pertaining to the particular problem being solved by the system and

3. an interpreter that decides what is to be done next i.e what rule is to be applied.

The task of the interpreter is to

1. match the rules against the context

2. if more than one rule match, resolve conflict and choose one of them and

3. apply the chosen rule

The interpreter is in a loop performing these three steps until no more rules are applicable or a solution has been found.

Besides the production system there are two other major components of an expert system.

1. I/O SYSTEM

This is an interface which puts questions to the user and passes the answers to the production system. Similarly it displays advice or explanations to the user from the production system.

2. KNOWLEDGE ACQUISITION SYSTEM

This is an interface to a human expert who monitors the performance of the system and updates rules in the knowledge base.

1.3 ALTERLNATIVE ARCHITECUTRES

Though the above mentioned structure is central to all expert systems,the nature of the problem demands different ways of interpretation. Depending on the nature of interpretation an expert system may be classified into two types as follows:

1. FORWARD REASONING

Forward reasoning builds up from the available facts about a situation to deduce conclusions. It is appropriate where the possible conclusions cannot be prespecified, as in designing a computer configuration, where an endless variety of end results are feasible.

2. BACKWARD REASONING

Backward reasoning involves working back from a conclusion or goal to see if the conditions which would make it true are satisfied. It is appropriate where the possible conclusions can be specified in advance _ for example in medical diagnosis or diagnosing faults in equipment.

The above classification is too broad in the sense that reasoning forward or backward is an overall problem solving strategy. However the search strategy may itself use heuristics or fuzzy algorithms or any other conflict resolving method in search which suits the domain. The concepts involved in some of them are discussed below.

1. HEURISTIC SEARCH

In order to solve many hard problems efficiently, it is often necessary to cnstruct a control structure that is no longer guaranteed to find the best answer but that will almost always find a very good answer. Such a technique is called a heuristic which improves the efficiency of a search process by resorting to rules of thumb. One example of a good general-purpose heuristic that is useful for a variety of combinatorial problems is the nearest neighbour algorithm, which works by selecting the locally superior alternative at each step.

2. PROBABILISTIC REASONING

So far we have assumed that all our facts are either known to be 'true' or 'false'. We have essentially not considered the possibility that we might know something that is 'probably true'. The mathematical theory of probability provides a way of describing and manipulating such uncertain knowledge. Sometimes very simple techniques of probability can be used effectively in AI.

One of the most useful results of probability theory is Bayes' Theorem, which provides a way of computing the probability of a particular event, given some set of observations.

Let

P(Hi|E) = the probability that hypothesis Hi is true given evidence E

P(E|Hi) = the probability that we will observe evidence E given that hypothesis i is true

P(Hi) = the a priori probability that hypothesis i is true in the absence of any specific evidence.

k = the number of possible hypothesis

The theorem states that

$$P(Hi|E) = \frac{P(E|Hi) * P(Hi)}{\sum_{n=1}^{k} P(E|Hn) * P(Hn)}$$

For more information on Bayes' Theorem and probabilistic reasoning, the reader can reder to [Charniak 82].

3. FUZZY REASONING

Let the symbol U, denote a universe of discourse, which may be an arbitrary collection of subjects or mathematical constructs. If A is a finite subset of U whose elements are u1,u2,.....,un, then A is expressed as

A = { u1,u2,......,un}.

A finite fuzzy subset A of U is a set of ordered pairs:

A = { (ui,mu(ui) }

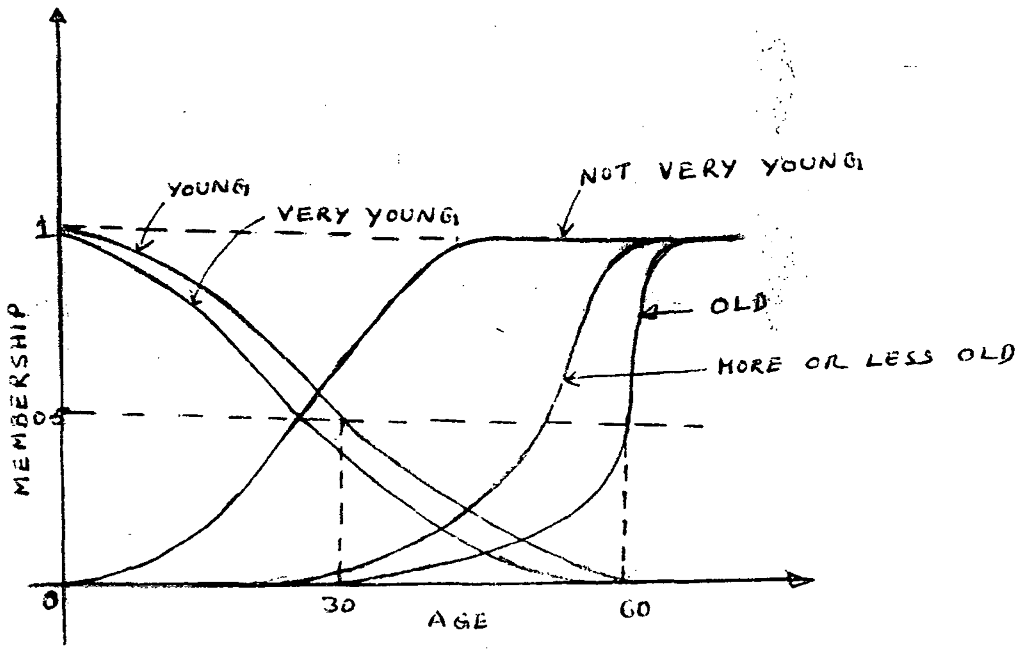
where ui belongs to U, and mu(ui) represents grades of membership (or membership functions) which indicate the degreee of membership. If all mu(ui) belong to {0,1}, the "fuzzy subset" will be understood as a "nonfuzzy subset" or "ordinary subset". The functions mu(ui)are then beinary boolean functions with 0 and 1 denoting no membership and full membership respectively.

LINGUISTIC VARIABLES AND FUZZY SUBSETS

The concept of fuzzy subsets is exemplified with linguistic variables. Informally, a linguistic varibale, L is a variable whose values are words or sentences in a natural language or in a subset of it. If age is interpreted as a linguistic variable, then its term-set T(age) might be

T(age) = { young, old, very young, not young, very old,
very very young, more or less young, ---- }

where each of the terms in T(age) is expressed by a fuzzy subset of a universe of discourse, say U = [0,100].



VARIOUS LINGUISTIC VALUES ARE EXPRESSED BY FUZZY SETS

4. INDEXING AND RULE SETS

Matching is the most expensive step in the application of rules. To make it more efficient, rules can be indexed by predicates or parameters. Whenever a value of the parameter is obtained, it can be used to determine which rules match due to the new value being available.

Indexing does not change the problem solving behaviour of an expert system, except perhaps in making it faster. The notion of rule sets originally suggested to deal with the efficiency issue is a variation in the architecture. In this rules are partitioned into sets. At a given time one set is active. What that means is that matching is attempted with rules in the current rule-set only. Switching among the rule sets is carried out by the interpreter.

5. FUNCTIONAL ATTACHMENT

Context is an efficient data structure for storing different parameters, and the parameters are stored and retrieved using functions attached to predicates. An alternative to this is to store parameters as assertions.

1.4 AI LANGUAGES FOR IMPLEMENTING EXPERT SYSTEMS

Higher level computer languages tend to fall into two broad classes. The programs that are written in the ALGOL-like, or block structure, languages are recognizable by the many block-delimiting BEGIN and END statements. These languages usually allocate space for varibales, arrays and other data before the program is executed(at compile time), so that during execution the space available for its data is fixed. The nested structure of the blocks defines the scope of the program variables and similarly defines which procedures can call which other procedures.

The LISP-like languages are characterised by dynamic allocation and dynamic scoping. Dynamic allocation means that the space to be used by a data object is not fixed ahead of time but is allowed to grow and shrink as needed - an essential attribute for list processing. Dynamic scoping means that any procedure can call any other, and variable values are passed down the control chain rather than being determined by the static block structure. That is once a varibale is declared, say in procedure A, it can be accessed from within any procedure B that A calls or any procedure C that B calls and so forth regardless of when A,B and C appear in actual program text.

Some of the desirable feautures of an AI language are:

1. Good facilities for manipulating lists, as lists are widely used in AI programs.

2. Late binding times so that the size of the data structre or the type of an object to be operated on, are not fixed before hand.

3. Pattern matching facilities, both to identify data and to determine control.

4. Facilitites for performing some kind of automatic deduction and for storing a database of assertions that provide the basis for deduction.

5. Facilities for building complex knowledge structures, such as frames, so that related pieces of information can be grouped together and accessed as a unit.

6. Control structures that facilitate goal-directed behavious(top-down processing) in addition to the more conventional data-oriented(or bottom-up) processing.

IPL, LISP, INTERLISP, SAIL, PLANNER, KRL, PROLOG are some of languages implemented for AI applications. Their feautures are discussed below in brief:

IPL

IPL (Information Processing Language) is a very early list-processing language. The language resembled a machine language more than a high-level language and is no longer in use.

LISP

LISP is the most established AI language invented at MIT by John Mc Carthy in the 1950s. LISP is more convenient for AI work than conventional data-oriented languages. One reason is that it allows the direct representation of symbolic concepts and the relationships between them in the form of data structures called lists - in fact lists are the only data structures in LISP. Another convenience of LISP is that it does not require the data types of each variable and the allocation of memory to each data type, to be specified at the beginning of the program; instead data types are determined at run time,and memory is alllocated flexibly according to requirements.

INTERLISP

There are many dialects of LISP, varying on everything from the names of standard functions and the order of their arguments to substantive issues involving the kinds of feautures provided. One of them INTERLISP, is sufficiently different from others. It has all of the capabilities of basic LISP, and provides additional feautures, which include:

1. a variety of data types, like arrays and bit strings, in addition to lists.

2. a spaghetti stack, in which several program contexts are stored simultaneously, so that control can be passed back and forth between co-routines.

3. or variety of tools to facilitate programming. DWIM an acronym for Do What I Mean, is a tool which

interfaces the system and the user, and does such useful things as correct spelling mistakes.

SAIL

SAIL is an ALGOL derivative and is the most similar to conventional general purpose programming languages. Since SAIL provides all the standard feautures of a programming language, it has been used in speech-recognition which involves a good deal of conventional computing.

PLANNER

PLANNER is a language built on topo LISP and designed for representing both traditional, forward-reasoning as well as goal-directed, backward reasoning. Programs in PLANNER consist of two types of statements:

1. Assertions, which simply state known facts.
2. Theorems, which describe how new facts can be informed from old ones.

There are three kinds of theorems that can occur in PLANNER programs:

1. Consequent theorems, that describe backward or goal-directed reasoning
2. Antecedent theorems, that describe forward, or data-directed, reasoning.
3. Erase theorems, that delete assertions from the database.

One of the main difficulties that arose with PLANNER was that the only available control structure was backtracking, which was automatic rather than being under the control of the programmer. To remedy this, a new language CONNIVER was built in which the programmer can explicitly direct the control flow of this program.

KRL

KRL is a language built on top of INTERLISP, that facilitates the representation of knowledge in frame structures (slot-and-filler structures). Its design was motivated by the following assumptions about knowledge representations and programs that use them.

1. Knowledge should be organized around conceptual entities with assoiciated descriptors and procedures.

2. A description must be able to represent partial knowledge about an entity and accommodate multiple descriptors which can describe the associated entity from different viewpoints.

3. An important method of description is comparison with a known entity with further specification of the described instance with respect to the prototype.

4. Reasoning is dominated by a processs of recognition in which new objects and events are compared to stored sets of expected prototypes, and in which specialized reasoning strategies are keyed to these prototypes.

PROLOG

PROLOG originated as an attempt to design a language which would allow the programmer to specify the objectives of a task in terms of symbolic logic, developed by Alan Colmeraur in Europe in the 1970s.

PROLOG originated as an attempt to design a language which would allow the programmer to specify the objectives of a task in terms of symbolic logic. A PROLOG program is predominantly "DECLARATIVE" in that it is concerned with stating WHAT has to be done, in the form of rules and facts, while a conventional program is more "PROCEDURAL" and concerned with HOW the task should be done. A major advantage of PROLOG is that the expert systems concept of an inference engine working on a knowledge base, and seeking to satisfy assigned goals by fixing rules, maps very directly on to the language; in a sense any PROLOG program can be seen as a sort of expert sytem.

A LISP program consists of a series of commands that manipulate symbols while a PROLOG program consists of statements of facts and rules. The powerful pattern matching capability and an automatic backtracking facility in PRFOLOG are an added advantage over LISP. PROLOG procedures are also flexible in the sense that the input and output paramerters are not predetermined but may vary from call to call.

1.5 SOME ADVANTAGES OF EXPERT SYSTEMS

Expert systems have emerged in the last few years as the leading practical application of the techniques developed in AI research. A new generation of expert systems are now put to use on day_to_day problems as they provide the most cost_effective means of doing the job. Some of the advantages offered by the expert system technology are given below.

1. Expert systems allow the computerisation of tasks which were previously unprogrammable. A leading example is the system used by DEC to configure their VAX minicomputer installations, originally known as R1 and now popular as XCON.

2. Expert systems are easier for users who are not programmers to understand. Because a knowledge base

is a fairly direct representation of human knowledge non_specialists can monitor its correctness and progress. At the same time the inference engine/knowledge base architecture allows the system to run even before the knowledge base is complete, and to provide some explanation of the reasoning behind a given conclusion. These feautures can be invaluable in the development of phase of a system.

3. Expert systems can allow a spectacular increase in programming productivity. Though productivity claims range from 10 to 50 times that achieved by conventional methods this is one area where the advantages of expert systems technology need to be proved in use.

4. Expert systems can provide a genuine extension of human capabilities. Expert Systems have already shown the capability of exceeding human performance in certain circumstances. A system developed at the University of Illinois for diagnosing disease in soya_bean plants can now produce more reliable diagnosis than the leading expert who set it up.

More dramatically in the foreseeable future expert systems will be able to take decisions in fast changing environmens from battles to foreign exchange trading, more effectively than humans ever could.

Where such systems will eventually lead is impossible to predict; their capabilities could be virtually boundless.

INTRODUCTION TO THE PROJECT

2.1 WHAT THIS PROJECT IS ABOUT?

This is a rule based expert system which identifies qualitatively the cation and the anion in a given compound interactively with the user, asking only a minimal number of questions. The whole problem can be visualised as a tree as shown in figure. The whole domain can be split into two individual sub tasks consisting of (1) the preliminary tests and (2) confirmatory tests.

The preliminary tests give a fairly good idea of the constituent radicals. They are used to eliminate other cations and anions from the complete set of possible anions and cations. These tests have been so designed that each test is capable of indicating a subset of cations and/or anions in the complete set of radicals identifiable by the test. For more information on the preliminary tests refer to appendix A or any standard text book on qualitative analysis in chemistry.

Once an estimate of the radicals present in the compound is obtained in the preliminary test, we proceed to the confirmatory tests. In the preliminary

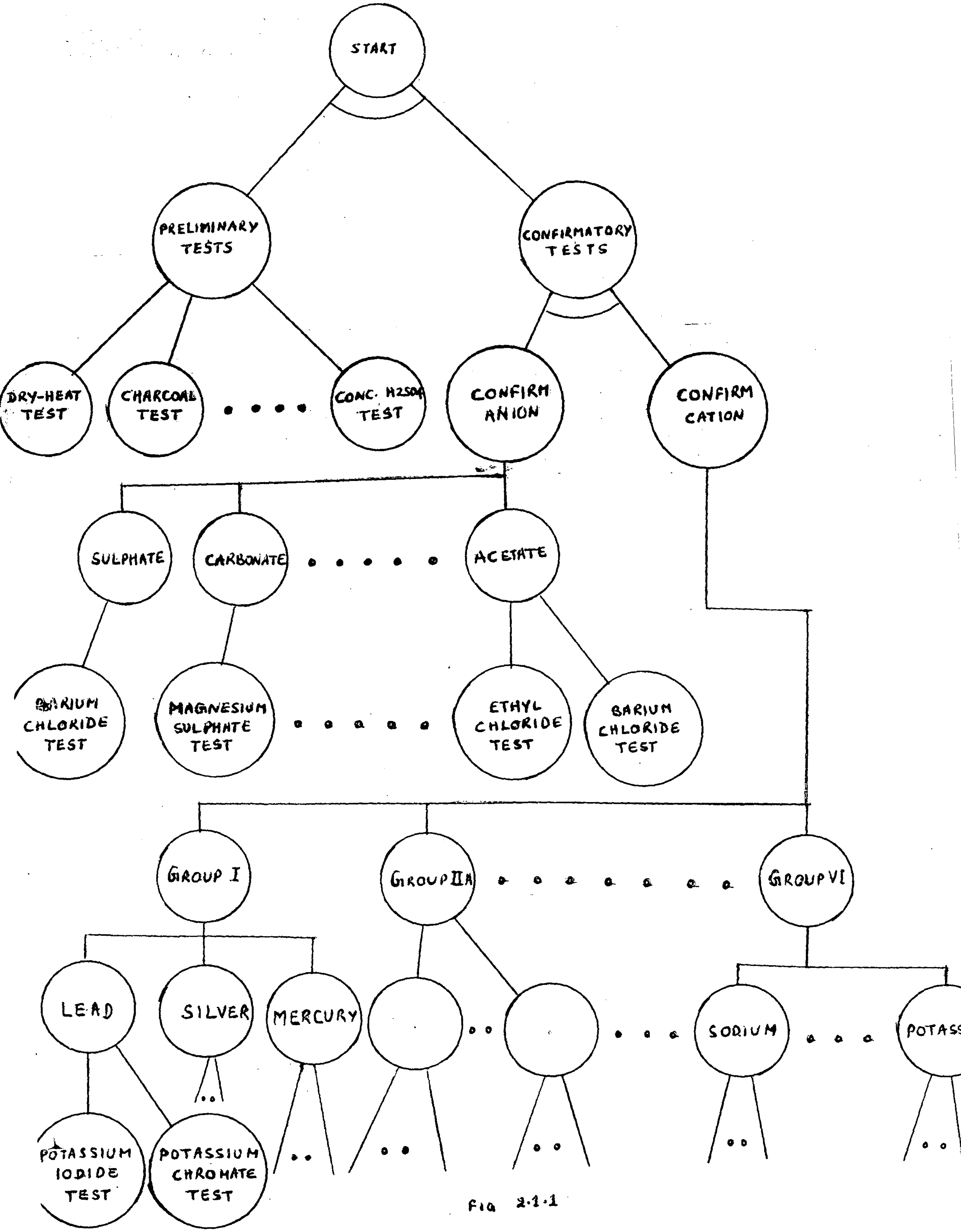

START
PRELIMINARY TESTS
CONFIRMATORY TESTS
DRY-HEAT TEST
CHARCOAL TEST
CONC. H2SO4 TEST
CONFIRM ANION
CONFIRM CATION
SULPHATE
CARBONATE
ACETATE
BARIUM CHLORIDE TEST
MAGNESIUM SULPHATE TEST
ETHYL CHLORIDE TEST
BARIUM CHLORIDE TEST
GROUP I
GROUP IIA
GROUP VI
LEAD
SILVER
MERCURY
SODIUM
POTASS
POTASSIUM IODIDE TEST
POTASSIUM CHROMATE TEST

Fig 2.1.1

tests, we infer from a set of reactions the possible radicals, while in the confirmatory tests we proceed from the radical to a set of definitely known conditions to confirm the radical.

For example in the preliminary tests,if carbon dioxide is liberated on heating the salt then the presence of carbonate or bicarbonate is indicated. While in the confirmatory tests for carbonate and bicarbonate if a white precipitate is formed on adding Magnesium Sulphate solution to the salt then carbonate can be confirmed. If a white precipitate is formed on heating the solution, then bicarbonate can be confirmed.

The results obtained in the preliminary tests must be confirmed before declaring the final result. It may be noted that the confimatory tests need not necessarily confirm the results obtained in the preliminary tests. This necessitates a means by which the radicals can be confirmed independent of the preliminary tests.

Confirming an anion is fairly simple and consists of identifying a test which results in a known condition like the formation of a precipitate or evolution of gas etc. On the other hand, to confirm a cation, a fixed set of sequential steps may have to be performed depending on the group to which the cation belongs. This is because all the cations are divided into six groups depending on their behaviour with specific group reagents, and a progressive elimination between members of the same group introduces an element of sequentiality. For more information on groups and group reagents in the confirmation of cations refer to Appendix C. All groups have a particualr group reagent excepting the fifth and sixth groups.

2.2 WHAT THIS PROJECT CAN DO?

This project is a rule based expert system in PROLOG which performs the following functions:

1. Determines the radicals in a compound interractively by asking a minimal number of questions(inferencing).

2. Provides justification at the user's request at any stage of execution as to WHY a question has been asked or as to HOW a deduction has been arrived at.

3. Can perform any of the preliminary tests or the confirmatory tests for any radical independent of the inferencing mechanism in 1.

The user is expected to give only an integer or an 'yes' or 'no' as a reply in response to a question or menu posed by the expert, making it easier for him to communicate with the expert. However in addition to these the user may request the expert to justify its questions either by a WHY or HOW.

2.3 SOME REPRESENTATIONAL DETAILS

RULE TYPE I

The PRELIMINARY routine uses indexed rule sets in its inferencing process. The rule sets are indexed by their first argument. A typical rule in a rule set has the following configuration.

```
ctxt(indx,conm1,conm2,..,conmn,
[catm1,catm2,..,catmr],[anm1,anm2,..,anmq]).
                              .....   rule type I
```

where

ctxt: takes one of the following predicates; gas_evolved,chn_colour_residue, colour_sublimate,dilh2so4,conch2so4, cobalt_nitrate,charcoal_cavity, colour_flame,colour_bead.

indx: is an integer value which indicates the index of the rule in the rule set.

conmn: is the nth condition for the mth rule in the rule set for the inference.

catmr: is one cation which can be inferred when (conm1,conm2,...,conmn) are all true.

anmq: is one anion, which can be inferred when (anm1,anm2,...,anmn) are all true.

For a listing of these rules see pages p.8 and p.9 in the program listings.The example below illustrates the meaning of the rule in more detail. In dry heat test the salt is heated in a test tube. A change in the colour of the residue on heating the salt is one likely result, which indicates the cation in the salt. The first three rules in this rule set are given below:

```
chn_residue(1,yellow,white,['Zn+2','Sn+2'],[]).
chn_residue(2,brown,brown,['Cd+2'],[]).
chn_residue(3,brown,yellow,['Pb+2','Bi+3'],[]).
```

A close look at these rules shows that the first argument is an integer value. This value is used as the index in retrieving the last arguments of 'ctxt', which are the lists required for inference. These rules can be translated into english as

If the change in colour of the residue is yellow when hot and white when cold then infer Zinc and Tin as the possible cations.

If the change in colour of residue is brown when hot and brown when cold the infer Cadmium as the possible cation.

If the change in colour of residue is brown when hot and yellow when cold then infer Lead and Bismuth as the possible cations.

RULE TYPE II

The confirmatory tests for anions use a different type of rules for their inference, which has the format as shown below:

```
test([  [rad1,[do11,do12,...,do1p],[result1]],
        [rad2,[do21,do22,...,do2p],[result2]],
        ...              ...            ...
        [radn,[don1,don2,...,donq],[resultn]]
     ]).
                              ..... rule type II
```

where

test: is the name of the test. This name is the reagent used in the test.

radn: is the nth radical which can be confirmed by the 'test'.

donq: is the qth step in the procedre to be performed.(don1,don2,...,donq) together form the complete procedure for the test.

resultn:is the result in which test should end for confirmation of the radical 'radn'.

The list [don1,don2,...,donq] shall be referred to as the DO list and the list [resultn] as the RESULT list hereafter. The example below illustrates rules of type II.

```
potassium_permanganate
([
    [nitrite,[write_sol],
        [disappearing,of,the,pink,colour,of,'KMnO4']],
    [sulphite,[write_sol],
        [disappearing,of,the,pink,colour,of,'KMnO4']],
]).
```

In the above rule, 'potassium_permanganate' is the name of the test, which is named after the reagent used. This reagent is capable of confirming two radicals viz nitrite and sulphite.

The DO list consists of a single predicate 'write_sol', which writes a sentence as below:

To 2-3 ml of Z, add a few drops of X

where Z and X are variables and

Z can take the values 'salt_solution' or 'sodium carbonate extract depending on whether the salt is soluble in water

or a sodium carbonate extract has been prepared for the
salt solution.

X takes on the reagent used in the test, which is the same
as the name of the test.

Depending on the user's response, the 'name of the test' and the 'solubility of the salt' are stored in the knoledge base as facts, which are retrieved later by the 'write_sol' predicate in writing the sentence. Refer to page p.20 for a listing of these predicates which, fill a sentence dynamically depending on the context and display the sentence on the screen.

Summarising, rules of type II can be translated into english as

If the test to be performed is 'test' and the radical is 'radical' then if do1, do2, ... , don are performed in that order then the result should be 'result' to confirm the radical

RULE TYPE III

The CONFIRM_CATION routine used another type of rules which look like rules of type II in their structuring, except that the DO list is replaced by a list which has the name of the solution to be taken for the test.

```
test([  [radical1,[solution1],[result1]],
        [radical2,[solution2],[result2]],
        ...              ...           ...
        [radicalm,[solutionm],[resultm]]
    ]).
                              ...  rule type III
```

The ASK_C routine asks the next question in the confirmation of cations. It gives the procedure and then asks a question. The procedure consists of a sentence which looks like as given below

To 2 ml of the X1 add a little of X2

where

X1 is the value of the 'solution' in the above rule for the test

X2 is the name of the test which by itself is the name of the reagent used in the test.

The rule translated into english looks like as below

If the test to be performed is 'test' and the radical is 'radical' then on adding the reagent 'test' to 2 ml of the solution the result is 'result' then the radical is confirmed.

The example below illustrates the above rules.

```
potassium_chromate
([
   [lead,[above,solution],[yellow,precipitate]],
[silver,[original,solution],[brick,red,precipitate]],
   [barium,[above,solution],[yellow,precipitate]]
]).
```

In the above rule 'above solution' is the solution in hand which is obtained after one or more reactions as requested by the expert; 'original solution' is solution of the salt obtained in the beginning for performing the wet tests for cations. This rule can be translated into english as

If the test to be performed is 'potassium_chromate test' and the radical is

'lead' then on adding the reagent
potassium_permanganate to 2 ml of
the 'above solution' the result is
'yellow precipitate' then the radical
lead is confirmed;

'silver' then on adding the reagent
potassium_permanganate to 2 ml of
the 'original solution' the result is
'brick red precipitate' then the
radical silver is confirmed;

'barium' then on adding the reagent
potassium_permanganate to 2 ml of the
'above solution' the result is
'yellow precipitate' then the radical
is confirmed;

IO_MODULE:

The rules of type II mentioned above are used by the IO_MODULE in questioning the user. The working of the IO_MODULE better illustrates the selection of the structure for rule type II.

The IO_MODULE performs three things always.

1. Questions the user depending on the context.
2. Accepts the user's response to the question.
3. Checks the value returned by the user and returns the same if it is an integer or an 'yes' or 'no', to the calling routine.

The context that is passed consists of the 'radical' and the 'test'. If the radical in the context is an anion, then the ASK_N routine retrieves the structure stored in the rule II for the test. The GET_PROCEDURE routine takes this structure and returns the DO list and the RESULT lists for the radical in the context.

The EXECUTE routine next executes the DO list, which consists of a series of evaluable predicates. The RESULT list is used in questioning the user on the result. After questioning the user on the likely result that confirms the anion, IO_MODULE waits for the response of the user and expects him to reply an 'yes' or 'no'.

If the radical is a cation, the ASK_C routine poses the next question. Unlike the ASK_N routine, ASK_C consists of an explicit listing of all the contexts as arguments of ASK_C. ASK_C matches against a different context each time to pose the question. It is possible to dispense with the explicit listing of all the contexts. More of this shall be discussed in chapter 5.

INFERENCING

This chapter consists of two sections, one on the preliminary tests and the second on confirmatory tests. The implementation of these two tasks have been discussed in detail. The user is expected to read this chapter with a constant reference to the listings of the program. The names of routines performing different tasks are written in bold letters, of the program, though the actual program uses small case letters.

3.1 PRELIMINARY TESTS

There are six tests in this module, and one or more of the rules corresponding to these tests(pages p.4 to p.6), are triggered by an interpreting module, which incorporates a best first strategy and a heuristic. Each test has an indexed rule set in the knowledge base. A typical set of rules for a test has the following representation.

```
ctxt(Indx1,con11,con12,--,con1n,
        [cat11,cat12,--,cat1p],[an11,an12,--,an1q]).
ctxt(Indx2,con21,con22,--,con2n,
        [cat21,cat22,--,cat2r],[an21,an22,--,an2s]).
        ...               ...               ...   ...
ctxt(Indxm,conm1,conm2,--,conmn,
        [catm1,catm2,--,catmo],[anm1,anm2,--,anma]).
```

52: Warning, can not justify.
--rule type I

A listing of these rules is found on pages p.8 and

When the test to be performed is decided by the control strategy the test is invoked. The procedure as to how the test is to be performed is read from a file. The file consists of the 'procedure' for the test and also a menu with the Index numbers and the corresponding conditions as listed in the rule type I (see Figure 3.1.1) are read from a file by the IO_MODULE. The IO_MODULE then waits for the input from the user. If a 'why' or a 'how' is keyed in, the MAP_W and ANS_HOW modules justify the question asked by giving explanations, depending on the context, as explained later in chapter four. If the user keys in an integer from the menu the IO_MODULE returns, the value read to the test. With the usr keyed_in value as the index number, the facts stored in the rule type I of the test are retrieved. The values in the conm1,conm2,---,conmn columns are used to answer the how explanations, while the last two columns which are lists containing the possible cations and anions respectively for the conditions as given by the user are returned as the inference from the test.

ake a platinum wire and make a loop at its end. Clean it thoroughly.Dip this wire in a test tube containing a little of concentrated HCl and then heat it in the oxidising flame. If the platinum wire is not clean,

some colour will be imparted to the flame. Repeat the process till the platinum wire does not impart any colour to the flame. Now take a pinch of the salt under analysis on a watch glass and make its paste with a few drops of concentrated HCl. Touch this paste with the platinum loop and introduce it into the oxidizing flame. Note the colour of the flame with the nacked eye as well as through a blue glass.

	COLOUR OF THE FLAME	
	WITH NAKED EYE	THROUGH BLUE GLASS
1	golden yellow	invisible
2	pale_violet	pinkish
3	bluish green	visible
4	crimson	crimson
5	brick red	yellow
6	grassy green	green
7	bluish white	none
8	no colour	none

Figure 3.1.1

The above procedure is performed for any test invoked by the interpreting module. The interpreting module uses the following algorithm in triggering a test.

1. Forms a list Q consisting of all the preliminary tests using the heuristic that the test which is capable of detecting the highest number of radicals occupies the first position in the list and second highest second and so on i.e. all the tests are

ordered in a descending order of the number of radicals detectable by the test.

2. UNTIL the list Q is EMPTY or the lists returned by a test M and N are SINGLETONS the following is done:

2a) Elimination of those tests in Q which cannot qualify as children.

2b) Evaluation of the static evaluating function for all the tests in Q.

2c) Performing that test for which the static evaluating function is maximum and removing it from the list.

3. If there is success M and N are returned else empty lists are passed.

The above three steps are detailed below.

Once the user requests the expert to assist him in the analysis of the compound, keying ANALYSE the PRELIMINARY routine is called which performs the preliminary tests.The SEARCH for identifying the radicals begins with all the tests qualifying to be performed. Before going into the details of the SEARCH routine, we shall see the format of the parameters passed between call to call of the SEARCH_routine.

< Q,M,N,RetP,RetQ >

where

Q : is the current list of remaining tests

M : is the running list of cations

N : is the running list of anions

RetP : is the list of cations to be returned by the test to be performed next

RetQ : is the list of anions to be returned by the test to be performed next

Once the search routine is invoked, the ELIMINATE routine eliminates all those tests which are not eligible as children depending on the running lists M and N as given below:

1. If both M and N are empty lists then Q is returned, as no elimination is possible.

2. If M is a singleton then all those tests which can identify ONLY cations are eliminated from Q and the remaining list is returned.

3. If N is a singleton then, all those tests which can identify ONLY anions are removed from Q,and the remaining list is returned.

4. If either M or N is an empty list the list

Q is retained.

The list returned by the ELIMINATE routine is passed on to the EVALUATE routine. This routine returns the next test to be performed as that test which has the highest value for the Static Evaluating Function(SEF).

SEF = |intersection of| + |intersection of|
|M and Cation | | N and Anion |

where
Cation: is the complete set of cations that can be detected by the current test

Anion: is the complete set of anions that can be detected by the current test

The EVALUATE routine makes use of the following points in conflicting cases where the test to be evaluated cannnot be determined on the basis of SEF:

1. If more than one test has the same value for SEF, that test which occurs first in the list Q will be returned. The heuristic that has been used in ordering the tests in Q in the beginning is made use here.

2. If the list returned by the ELIMINATE routine is not empty and the running lists M and N are not singletons and the maximum value for SEF for all the tests in the above list is zero then that test which occurs as the head of the list Q is returned.

The DEL routine deletes the test returned by the EVALUATE routine, from the list returned by the ELIMINATE routine and returns the deleted list as T3.

The JUSTIFY module requests the user to perform the next test and waits for the user's reply as to whether he wants justification for the same. If the user's response is affirmative, the following two types of justifications are given depending on the values of M and N.

1. If M and N are empty lists then the number of radicals detectable by a test is taken as the criterion, which is displayed for all the tests in Q and the test with the highest value will be the test to be performed.

2. If M and N are not empty lists then the intersection of the running lists M and N with those cations and anions which can be detected by the test

is displayed.

Figure 3.1.2 illustrates how the JUSTIFICATION module asks the user whether he is interested in the justification or not. It takes the case when
Q =
[charcoal_test,borax_bead_test,dil_sulphuric_acid_test,

conc_sulphuric_acid_test]

M = [Na+]

N = [NO3-,I-,Cl-]

The 'List of remaining preliminary Tests' in figure 3.1.2 is pruned after performing the 'conc_sulphuric_acid_test'. The tests 'borax_bead_test' and 'dil_sulphuric_acid_test' have been eliminated from the list after performing the test. The name of 'borax_bead_test' has been removed from the list as it can detect only cations and the 'list of possible cations' is a singleton and contains 'Na+' as its element. The name of 'conc_sulphuric_acid_test' has been removed by the DEL routine as it has been performed. The name of 'charcoal_test' has been removed from the list as it can detect both cations and anions.

```
List of remaining preliminary Tests= [charcoal_test,
        borax_bead_test,dil_sulphuric_acid_test,
        conc_sulphuric_acid_test]
List of possible cations= [Na+]
List of possible anions= [NO3-,I-,Cl-]
ok

I want you to perform  conc_sulphuric_acid_test

Do you want me to justify?

>yes.

Out of the above possible cations and anions
charcoal_test can identify the radicals  Pb+2

flame_test can identify the radicals  Na+ K+ Pb+2

borax_bead_test can identify the radicals  none

dil_sulphuric_acid_test  can  identify  the  radicals
none

conc_sulphuric_acid_test  can  identify  the  radicals
NO3- I- Cl-

ok
```

Tell me whether the evolving gas is a 1. colourless and odourless gas 2. colourless gas with odour 3. coloured gas with pungent smell

>2.

Tell me whether the evolving gas is COLOURLESS and 1. smells like rotten eggs and
turns lead acetate paper black 2. a characteristic suffocating smell
and turns acidified K2Cr207 paper green 3. has a pungent smell and
produces white fumes with ammonia and
a white ppt with AgNO3 solution 4. characteristic vinegar like smell 5. sweet smell and vapours catch fire 6. characteristic ammoniacal smell and
turns moist turmeric paper brown

>3.

List of remaining preliminary Tests= [charcoal_test,dil_sulphuric_acid_test] List of possible cations= [Na+] List of possible anions= [Cl-]

ok

FIGURE 3.1.2 ~~3.1.2~~

The DO routine does the next test. It invokes the test to be done as decided by the previous routines. All the tests return two lists, the first being the list of possible cations and the second that of anions. Each test also returns a trace list which sumps up in a sentence the result of the test. The con1,con2,---,conn in rule type I as explained in the beginning of the chapter are stored in this trace list. Once a test is completed, the DO routine asserts in the knowledge base two kinds of facts.

1. Firstly the result of the previous test is stored as follows

< result(Test,Cation,Anion) >

where result is the predicate used Test is the name of the test performed last Anion is the inferred list of anions in the test and Cation is the inferred list of cations in the test

2. The 'Trace' list returned by a test is also asserted as a fact in the knowledge base as how(Trace) where 'how' is the predicate and 'Trace' is the list returned by the test. The results of the test stored as result(Test,Cation,Anion) in the knowledge, is utilised in the CHECK_CONSISTENT routine which is explained next, while the facts how(Trace),

are used to give 'how' explanations as explained in the next chapter.

The CHECK_CONSISTENT routine checks for consistency in the results of the previous test.If the skein of logic in choosing the next test to be performed in the previous modules is true, then the result of the present test must yeild in the pruning of running list. In other words, the results of the present test must be a subset of the running list. If this condition is not satisfied, the CHECK_CONSISTENT routine displays the results of previous and the present tests and offers five options as below to remove the inconsistency.

1. retain the results of the previous test 2. retain the results of the present test 3. take union of the results of the present and previous tests 4. take intersection of the results of the present and previous tests 5. you want to perform the last test once again.

For the first four options corresponding values are returned while in the fifth option, the result of the last test is retrieved from the knowledge base and the test is performed once again. The results obtained in the second performance are returned.

After the consistency check on the results obtained succeds, the results are output by the OUTPUT_STATUS routine. With T3 as the current list and results returned by the CHECK_CONSISTENT routine as the running lists, the SEARCH proceeds, till both the running lists are singletons or T3 is empty. In Figure 3.1.2 the status regarding Q, M and N are output by the OUTPUT_STATUS routine.Figure 3.1.3 gives a complete listing of a session with the PRELIMINARY tests routine.

```
| ?- |    preliminary(M,N).

I want you to perform  dry_heating_test

Do you want me to justify?
>yes.

Total number of anions and cations which can be identified by
                    dry_heating_test is 25
                    charcoal_test is 14
                    flame_test is 9
                    borax_bead_test is 7
                    dil_sulphuric_acid_test is 6
                    conc_sulphuric_acid_test is 6

ok

Take a pinch of the salt in a dry test tube and heat it.
Keep the test tube rotating to ensure uniform heating so that the
condensed water vapours if any do not fall back on the residue.
The following changes may take place:

        1. A gas or vapour is evolved
        2. change in colour of the residue
        3. A sublimate is formed
        4. Crackling noise is produced
        5. Fusion or swelling of the salt is observed
        6. Water of Crystallization is produced
        7. No reaction is observed
```

Indicate ONE or MORE of your options after carefully observing the reaction. DELIMIT your options by a COMMA.

>4.

Is there a crackling noise on heating the salt?

yes.

List of remaining preliminary Tests= [charcoal_test,flame_test,borax_bead_test, dil_sulphuric_acid_test,conc_sulphuric_acid_test]
List of possible cations= [Na+,K+,Pb+2]
List of possible anions= [NO3-,I-,Cl-]

ok

I want you to perform flame_test

Do you want me to justify?

>yes.

Out of the above possible cations and anions
charcoal_test can identify the radicals
Pb+2

flame_test can identify the radicals
Na+ K+ Pb+2

borax_bead_test can identify the radicals
none

dil_sulphuric_acid_test can identify the radicals
none

conc_sulphuric_acid_test can identify the radicals
NO3- I- Cl-

ok

Take a platinum wire and make a loop at its end. Clean it thoroughly.Dip this wire in a test tube containing a little of concentrated HCl and the heat it in the oxidising flame. If the platinum wire is not clean, some colour will be imparted to the flame. Repeat the process till the platin wire does not impart any colour to the flame. Now take a pinch of the Sal under analysis on a watch glass and make its paste with a few drops of concentrated HCl. Touch this paste with the platinum loop and introduce into the oxidising flame. Note the colour of the flame with the nacked ey as well as through a blue glass.

	COLOUR OF THE FLAME WITH NAKED EYE	THROUGH BLUE GLASS
1	golden yellow	invisible
2	pale violet	pinkish
3	bluish green	visible
4	crimson	crimson
5	brick red	yellow
6	grassy green	green
7	bluish white	none
8	no colour	none

```
List of remaining preliminary Tests= [charcoal_test,borax_bead_test, ...
ic_acid_test,conc_sulphuric_acid_test]                          sulph
List of possible cations= [Na+]
List of possible anions= [NO3-,I-,Cl-]

ok

I want you to perform  conc_sulphuric_acid_test

Do you want me to justify?

>yes.

Out of the above possible cations and anions
charcoal_test can identify the radicals
none

dil_sulphuric_acid_test can identify the radicals
none

conc_sulphuric_acid_test can identify the radicals
NO3- I- Cl-

ok


Take a little of the salt in a test tube and treat it wit
a few ml of concentrated sulphuric acid. Heat the content
if no gas is evolved
Tell me whether a gas is being evolved?

>yes

 ell me whether the evolving gas is a
     1.  colourless and odourless gas
     2.  colourless gas with odour
     3.  coloured gas with pungent smell

>2

Tell me whether the evolving gas is COLOURLESS and
        1.  smells like rotten eggs and
                 turns lead acetate paper black
        2.  a characteristic suffocating smell
                 and turns acidified K2Cr2O7 paper green
        3.  has a pungent smell and
                 produces white fumes with ammonia and
                 a white ppt with AgNO3 solution
        4.  characteristic vinegar like smell
        5.  sweet smell and vapours catch fire
        6.  characteristic ammoniacal smell and
                 turns moist turmeric paper brown

>3.

List of remaining preliminary Tests= [charcoal_test,dil_sulphuric_ac
List of possible cations= [Na+]                              id_test]
List of possible anions= [Cl-]
```

```
ok

: ?- :  how.

In dry_heating_test
if there is crackling noise then infer
Na+ k+ Pb+2

No3- I- Cl-

ok

In flame_test
if the colour of the flame is golden yellow with naked eye and
invisible with blue glass then infer
Na+ []

ok

In conc_sulphuric_acid_test
on adding conc sulphuric acid
if the gas is colourless and purcent smell and white fumes with ammo
white ppt with silver nitrate solution then infer HCl
then infer [] Cl-

ok

M=[Na+]
N=[Cl-]
```

Fig 3.1.3

CONFIRMATORY TESTS

The PRELIMINARY routine as explained in chapter 3.1 yields two lists which may or may not be singletons. This module confirms one of the radicals in each of the above lists. So it consists of two phases viz confirmation of anion and confirmation of cation.

CONFIRMATION OF ANION

```
test([ [radical1,[do11,do12,...,do1n],[result1]],
[radical2,[do21,do22,...,do2n],[result2]],
   ...              ...              ...
[radicalm,[dom1,dom2,...,domq],[resultm]]
]).
                                        ... rule type II

radical([test1,test2,...,testn]).

                                        ... rule type IV
```

Rule type IV can be translated into english as, the tests which can confirm the 'radical' are test1, test2, ... , testn . 'radical' can take the names of any of the radicals but not their formulas.

The PRELIMINARY module uses the formulas of the radicals while the confirmatory tests module uses their names. This has been done for

1. the user's convenience in case he intends to confirm some radical independent of the inferencing

2. to make the rules of type II and type III more meaningful

The CONFIRM_ANION* routine makes use of the following algorithm:

1. DISPLAY all the tests which can confirm the given radical and SELECT that test which is chosen by the user. If there is only one test which can confirm the radical then proceed with it as there is no choice.

2. If the test requires the preparation of the salt_solution, do so.

3. Request the user to perform the test by giving the procedure. Also question him, as towhether the test

Refer to page p.13 for CONFIRM-ANION routine.

has RESULTED in a condition that confirms the test.

4. Succeed if the user's reply is 'yes' or else fail if it is 'no'.

The above four steps are discussed in more detail with the names of the routines that perform the different parts of the algorithm.

The SYNONYM routine returns the 'name' of the radical, if the 'formula' is given. If a 'name' is its argument then it returns the same.

The GET_TESTS routine takes the name of the radical given by the SYNONYM routine and returns the tests that can confirm the radical. The list of tests is retrieved from the rule of type II corresponding to the 'radical' in question.

Since the test to be performed to confirm the radical has beeen decided, we can proceed to the IO_MODULE to give the procedure for the test. But before that most of the tests require that the solution of the salt be prepared either in water or in sodium carbonate solution.

The SALT_SOLUTION routine requests the user to prepare the solution of the salt in water or sodium carbonate solution depending on the test to be performed and radical in question as below:

radical	test
carbonate	-
bicarbonate	-
borate	-
acetate	ethyl_alcohol
-	chromyl_chloride
-	manganese_dioxide
nitrate	copper_turnings

2. If the radical and the test do not come under any of the conditions listed above, the user is asked to prepare the aqueous solution of the salt in water.

3. Finally if the salt is insoluble in water the salt is mixed with sodium carbonate in distilled water and boiled to obtain the SODIUM CARBONATE EXTRACT of the salt.

The IO_MODULE routine asks the next question and accepts the user's response as input. The first and the second arguments of IO_MODULE are the 'radical' and the 'test' to be performed. When the io_module is called from the CONFIRM_ANION routine the ASK_N routine matches against a unique question.

The above routines are illustrated below with an example taking the anion 'sulphite'. Figure 3.2.1 gives a complete session of questions of the expert and the user's response. The SYNONYM routine converts the formula of sulphite into its name. As there are more than one test which can confirm sulphite the user is given the information that the 'barium_chloride_test', 'ferric_chloride_test' and the 'potassium_permanganate_test' can confirm sulphite. This question is posed together by the WRITE_ANY routine and the DISPLAY_SELECT routine. The user's response '1.' selects 'barium_chloride_test'. As the solution of the salt is required for this test the user is asked to dissolve the salt in water. Since the salt is not soluble in water as indicated by the user's response 'no', he is asked to perform the 'sodium_carbonate extract'. This is done by the SALT_SOLUTION routine. Tkhe SALT_SOLUTION routine asserts the information that the sodium carbonate extract has been prepared using the predicate SOLUBLE_IN. This information is used in asking the next question by the ASK_N routine. The next question is given here for convenience

To 2-3 ml of SODIUM CARBONATE EXTRACT add a few drops of barium_chloride. Filter the ppt and treat the residue with dilute HCl.

Tell me whether
DISSOLUTION of the ppt with the evolution of SO2
is observed?

The words, 'SODIUM CARBNONATE EXTRACT' and 'barium_chloride' are inserted into the sentence by the ASK_N routine. The fourth sentence is the 'result' in the rule type II for barium_chloride test and sulphite radical. The user's response is read in by the IO_MODULE

ROUTINE. The response 'why' invokes the MAP_Y routine which gives the explanation. This explanation is read from a file whose name is the same as the name of the radical in context. As any response of the user other than an integer or an 'yes' or 'no' does not result in the success of the IO_MODULE routine. The same question is posed again by the ASK_N routine. An 'yes.' from the user resulted in the confirmation of sulphite.

```
| ?- |     confirm_anion('SO3-2').

Any of the following tests confirm the presence of
sulphite radical. Indicate your choice by keying the
number against the test

1. barium_chloride test
2. ferric_chloride test
3. potassium_permanganate test

>1.

Mix a little of the salt in water and
Tell me whether a(an) dissolution of the salt in water
is observed?

>no.

If the salt is insoluble in distilled water prepare the
sodium carbonate extract of the salt as given below:
Take about 1gm of the salt under analysis in a boiling
test tube. Mix it with about 2gms of sodium carbonate.
Put in it for about 5 minutes and filter. The filtrate
is called SODIUM CARBONATE EXTRACT.

Sodium Carbonate Extract contains unused sodium
carbonate. It must be destroyed before confirming an
```

acid radical. Otherwise precipitate due to insoluble carbonate may result. Dilute acid may be used for this purpose. Add dilute acetic acid to the sodium carbonate extract DROP BY DROP TILL THE EFFERESCENCE CEASES.

ok

To 2-3 ml of SODIUM CARBONATE EXTRACT add a few drops of barium_chloride. Filter the ppt and treat the residue with dilute HCl.
Tell me whether
DISSOLUTION of the ppt with the evolution of SO2
is observed?

>why.

SULPHITE

(i) BARIUM CHLORIDE TEST

$$Na_2SO_4 + BaCl_2 \longrightarrow 2\ NaCl + \underset{\text{(white ppt)}}{BaSO_3}$$

$$BaSO_3 + 2\ HCl \longrightarrow BaCl_2 + SO_2 + H_2O$$

(ii) FERRIC CHLORIDE TEST

$$3\ Na_2SO_4 + 2\ FeCl_3 \longrightarrow \underset{\text{(dark red)}}{Fe_2(SO_3)_3} + 6\ NaCl$$

(iii) POTASSIUM PERMANGANATE TEST

$$2\ KMnO_4 + 3\ H_2SO_4 \longrightarrow K_2SO_4 + 2\ MnSO_4 + 3\ H_2O + 5\ O$$

$$Na_2SO_3 + O \longrightarrow Na_2SO_4$$

Colour disappears as nascent oxygen is taken up by sodium sulphite.

ok

To 2-3ml of SALT SOLUTION add a few drops of barium_chloride. Filter the ppt and treat the residue with dilute HCl.
Tell me whether DISSOLUTION of the ppt with the evolution of SO2 is observed?

>yes.

Sulphite is confirmed.

Figure 3.2.1

CONFIRMATION OF CATION

The confirmation of cation is very similar to that of the anion. The CONFIRM_CATION rutine uses rules of type III as explained in chapter 2.2. The function of SYNONYM routine is the sameas explained before. The FIND_GROUP routine determines the group to which the cation belongs. This routine uses the data structure

t(Gr,Gr_members,Next)

where

Gr	is the name of the group
Gr_members	is the list of the names of members of the group
Next	is the remaining 't structure' as explained above

See page p.14 for this 't structure'.

After the group to which the cation belongs, has been determined the SALT_SOL routine gives the procedure for preparing the solution of the salt. Then the group_procedure is called using the name of the group obtained in the FIND_GROUP routine. For a listing of these procedures see pages p.21 and p.22. Within the group the other memebers are progressively eliminated and the CONF routine is called for the radical which is to be confirmed. It may be noted here that the name of the radical handed to the CONF routine uses a logic which is very much parallel to the CONFIRM_ANION routine. Figure 3.2.2 gives a session with the CONFIRM_CATION routine for the cation 'chromium'.

¦ ?- ¦ confirm_cation(chromium).

PREPARATION OF THE ORIGINAL SALT SOLUTION

For the wet tests of cations, the first step is the preparation of salt solution. The salt may dissolve in one of the solvents given below. Tkhe following solvents are to be used in the ORDER given:

1. Water, cold and boiled.
2. Dilute Hydrochloric acid, cold and hot.
3. Concentrated Hydrochloric acid, cold and hot.
4. Dilute Nitric acid, cold and hot.
5. Aqua regia (a mixture of 3 volumes of conc. HCl and 1 volume of concentrated HNO3).

In case a gas comes out, boil off the gas completely and get a clear solution. After selecting the right solvent with a pinch of the salt, prepare its concentrated solution. It is called the SALT SOLUTION.

ok

To 5ml of the original solution add 4-5 drops of concentrated nitric acid. Boil the solution for some time. Add excess of Ammonium Hydroxide to it and shake. A precipitate indicates the presence of cation of groupIII. If a ppt is formed give me the colour of the ppt by keying the number against the colour given below.

1. reddish brown
2. green
3. gelatinous white

If no ppt is formed reply 'no'.

>why.

EXPLANATION

The group III cations are precipitated as hydroxides on the addition of excess of NH4OH.

```
FeCl3 + 3 NH4OH -->  3 NH4Cl + Fe(OH)3
                               (reddish brown)

CrCl3 + 3 NH4OH -->  3 NH4Cl + Cr(OH)3
                               (green)

AlCl3 + 3 NH4OH -->  3 NH4Cl + Al(OH)3
                               (white)
```

ok

To 5 ml of the original solution add 4-5 drops of concentrated nitric acid. Boil the solution for some time. Add to it 1.5 gms of NH4Cl and boil again. Cool the solution under tap water. Add excess of Ammonium Hydroxide to it and shake. A precipitate indicates the presence of cation group III. If a ppt is formed give me the colour of the ppt by keying the number against the colur given below.

1. reddish brown
2. green
3. gelatinous white

If no ppt is formed reply 'no'.

>2.

Any of the following tests confirm the presence of chromium radical. Indicate your choice by keying the number against the test.

1 lead_acetate test
2 hydrogen_peroxide test

>why.

*** BAD INPUT ***
>1.

To 2ml of the solution obtained by extracting the above ppt and NaOH and NaNO3 with water. Add a little of lead acetate. Tell me whether a(an) yellow ppt soluble in NaOH is observed?

>why.

LEAD ACETATE TEST

$KNO_3 \longrightarrow KNO_2 + O$

$2\ Cr(OH)_3 + 4\ NaOH + 3\ O \longrightarrow 2\ Na_2CrO_4 + 5\ H_2O$
(green)

$Na_2CrO_4 + (CH_3COO)_2Pb \longrightarrow 2\ CH_3COONa + PbCrO_4$
(yellow)

$PbCrO_4 + 4\ NaOH \longrightarrow Na_2PbO_2 + Na_2CrO_4 + 2\ H_2O$
(sodium plumbite).

ok

To 2ml of the solution obtained by extracting the above ppt and NAOH and NaNO3 with water. Add a little of lead acetate. Tell me wheteher a(an) yellow ppt soluble in NaOH is observed?

>yes.

chromium is confirmed.

Figure 3.2.2

JUSTIFICATION

There are two kinds of justifcations the expert offers. The first one is a 'why' explanation and the second is 'how' explanation. In response to a question posed by the expert the user may request for any of these explanations. The 'why' explanation contains a contextual reasoning as to 'why' the question has been asked. On the other had a 'how' explanation containes the information as to 'how' the deduction has been arrived at.

In the PRELIMINARY routine the 'why' explanations are generated using the context number which is an index number. The 'why' explanations in the CONFIRM_ANION and the CONFIRM_CATION routines are generated by the MAP_Y and the ANS_WHY routines respectively. For a listing of these routines refer to page p.19.

As already explained in chapter 3, each test returns a 'Trace' list which is asserted in the knowledge base as a fact, 'how(Trace)'. The 'how' explanations are generated by the ANS_HOW routine which retrieves the information stored in the facts 'how(trace)' one by one and displays the lists 'Trace' for different tests on the screen.

SOME COMMENTS ON FURTHER ENHANCEMENTS

A knowledge acquisition module can be written which incorporates new rules into the knowledge base. This module should question the user on whether the new rule he intends to insert comes under the PRELIMINARY routine or CONFIRM_ANION routine or CONFIRM_ANION routine as three different types of rules are used in each of these routines. After this has been determined, the user may be given the format of the rule type. A consistency check on the input of the user should also be introduced to check whether the new rule is in accordance with the already existing rules in the knowledge base.

However it is possible to reorganise the knowledge base to improve the knowledge content of the program as discussed below:

1. The input/output of the program is done by the IO_MODULE routine which puts a question and accepts the user's response. The tasks of 'asking a question' and 'accepting the input' have been merged into one single routine. This may be broken up into two for greater efficiency and flexibility.

2. The history of the past questions and user's responses are not being retained by the IO_MODULE

routine. To do so, the rules of type II and type III have to be re_oriented to make them more self_explanatory. This can be done by difining predicates for each real world operation like heating, adding,filtering etc. Now the complete procedure for any test can be converted into a lsit of above defined predicates. This list containing the procedure can be interpreted in different ways by the question routine and the ANS_HOW routine.

Such a difinition of real world operations into predicates will also facilitate in the development of a two more rutines which will enhance the capability of the knowledge base. The first one being the capability of the knowledge base. The first one being, 'an equation generator', which generates a chemical equation for a reaction between two or more reagents. Such a routine improves the capability of the knolwedge base in answering the user's questions, because the complete history can be stored in terms of the real world operations, which when handed to this routine generates the likely output. Moreover it gives room for writing a second module which can demonstrate graphically the complete reaction between the reagents involved. Such a knowledge base can almost replace an actual chemical laboratory.

3. The IO_MODULE routine can be enhanced to provide a

'help' facility to the user, which explains the different terms used in the text of the question or the procedure displayed on the screen. Such a routine, acquaints the expert to the user quickly.

```
analyse:-repeat,preliminary(P,Q),
        ((Q\==[],confirm_one_anion(Q,Y));
         (an(List),confirm_one_anion(List))),
        ((P\==[],confirm_one_cation(P,Y2));
         (Z==no,confirm_groupwise)).

confirm_one_cation([],Y):-!.
confirm_one_cation([H|T],Y):-    ((confirm_cation(H),Y=H);
                                 confirm_one_cation(T,Y)).
confirm_one_anion([],Y):-!.
confirm_one_anion([H|T],Y):-     ((confirm_anion(H),Y=H);
                                 confirm_one_anion(T,Y)).

/* preliminary tests module */
list_tests([dry_heating_test,charcoal_test,flame_test,borax_bead_test,dil_sulph
uric_acid_test,conc_sulphuric_acid_test]).

preliminary_tests:-preliminary(P,Q),
        displist(['List',of,possible,'cation(s)',=,P]),nl,
        displist(['List',of,possible,'anions(s)',=,Q]).

preliminary(P,Q):- repeat,list_tests(List),(abolish(how,1);true),
        search(List,[],[],P,Q),!.

/* search routine */
search([H|T],[],[],RetP,RetQ):-repeat,justify(H,[],[],[H|T]),
                do(H,Rcat,Ran),
                check_consistent(Rcat,Ran,[],[],Retcat,Retan),
                output_status(T,Retcat,Retan),
                search(T,Retcat,Retan,RetP,RetQ).

search([],M,N,M,N).
search(_,[X],[Y],[X],[Y]).
search(Q,M,N,RetP,RetQ):- eliminate(Q,M,N,[Highest|T]),
                evaluate([Highest|T],Highest,0,M,N,Rethigh),
                del(Rethigh,[Highest|T],T3),
                justify(Rethigh,M,N,[Highest|T]),
                do(Rethigh,Rcat,Ran),
                check_consistent(Rcat,Ran,M,N,Retcat,Retan),
                output_status(T3,Retcat,Retan),



                search(T3,Retcat,Retan,RetP,RetQ).

check_consistent(Rc,Ra,Mc,Na,Rtcat,Rtan):-
        (    (Rc==[],Ra==[],Rtcat=Mc,Rtan=Na);
             (Rc==[],Ra\==[],Rtcat=Mc,consistent(Rtan,Ra,Na));
             (Rc\==[],Ra==[],consistent(Rtcat,Rc,Mc),Rtan=Na);
             (Rc\==[],Ra\==[],consistent(Rtcat,Rc,Mc),consistent(Rtan,Ra,Na))
        ).


output_status(T,Retcat,Retan):-
                nl,write('List of remaining preliminary Tests= '),write(T),
                nl,write('List of possible cations= '),
                        ((Retcat==[],write([all]));write(Retcat)),
                nl,write('List of possible anions= '),
                        ((Retan==[],write([all]));write(Retan)),
                pause.
```

```
consistent(Rt,Ra,Na):-repeat,
 ((subset(Ra,Na),Rt=Ra);
  (nl,write('INCONSISTENT DATA  !!'),nl,
   nl,write('Result of the previous test is  '),tab(2),
   write(Na),nl,write('Result of the present test is '),tab(2),
   write(Ra),nl,nl,write('What do you want me to do?'),
   nl,tab(5),write('1. retain the result of the previous test'),
   nl,tab(5),write('2. retain the result of the present test'),
   nl,tab(5),write('3. take union of both the present and previous test result
'),
   nl,tab(5),write('4. take intersection of the present and previous test resu
ts'),
   nl,tab(5),write('5. you want to perform the last test once again'),
   see(user),nl,prompt(In,>),read(Z),decision(Z,Rt,Ra,Na)
  )
 ).

justify(H,M,N,Q):-nl,write('I want you to perform  '),write(H),
                  nl,nl,write('Do you want me to justify?'),
                  nl,prompt(In,>),read(Z),
((Z==yes,
  ((M==[],N==[],
   nl,write('Total number of anions and cations which can be identified by'),
   nl,print_status(Q),pause,!);
   (nl,write('Out of the above possible cations and anions'),
    pr_status(Q,M,N),pause,!)
  )
 );
 (Z==no,!)
).

decision(1,Nai,Ram,Nai).
decision(2,Rai,Rai,Nam).
decision(3,Rtm,Ram,Nam):-append(Ram,Nam,Rtm).
decision(4,Rtm,Ram,Nam):-intersection(Ram,Nam,Rtm).
decision(5,Rtm,Ram,Nam):-result(Last_test,_,_),do(Last_test,Rm,Nm),
                         (((Last_test==dil_sulphuric_acid_test;
                            Last_test==conc_sulphuric_acid_test),
                            Rtm=Nm
                          );
                          ((Last_test==flame_test;
                            Last_test==borax_bead_test;
                            Last_test==charcoal_test),
                            Rtm=Rm
                          )



                         ).

eliminate(Que,[X],_,Retque):-actualdel(Que,Retque).
eliminate(Que,_,[Y],Retque):-delan(Que,Retque).
eliminate(Que,P,[],Retque):-Retque=Que.
eliminate(Que,[],P,Retque):-Retque=Que.
eliminate(Que,_,_,Que).

actualdel([],[]).
actualdel([H|T],T1):-Y=..[H,_,0,_,_,_],call(Y),actualdel(T,T1).
actualdel([H|T],[H|T1]):- actualdel(T,T1).

delan([],[]).
delan([H|T],T1):-Y=..[H,0,_,_,_,_],call(Y),delan(T,T1).
delan([H|T],[H|T1]):-delan(T,T1).
```

```
print_status([]):-!.
print_status([H|T]):-P=..[H,_,_,N,_,_],call(P),
                          tab(15),displist(['*  ',H,is,N]),nl,
                          print_status(T).

pr_status([],_,_):-!.
pr_status([H|T],M,N):-P=..[H,_,_,_,C,A],call(P),
                intersection(M,C,T1),intersection(N,A,T2),
                append(T1,T2,T3),nl,((T3==[],T4=[none]);T4=T3),
                displist([H,can,identify,the,radicals,T4]),
                pr_status(T,M,N).

evaluate([],Big,Bigvalue,M,N,Retbig):- Retbig=Big.
evaluate([H|T],Big,Bigvalue,M,N,Retbig):-
                Y=..[H,_,_,_,P,Q],call(Y),
                sef(P,Q,M,N,Total),
   (  (Total=<Bigvalue,evaluate(T,Big,Bigvalue,M,N,Retbig));
      (Rbig=H,Rbigvalue=Total,
                evaluate(T,Rbig,Rbigvalue,M,N,Retbig))
   ).

del(Rhigh,[Rhigh|L1],L1).
del(Rhigh,[H|L1],[H|L2]):-del(Rhigh,L1,L2).

do(Rethigh,Retcat,Retan):-Y=..[Rethigh,Retcat,Retan,Tr],call(Y),
    asserta(result(Rethigh,Retcat,Retan)),assertz(how(['In',Rethigh,&,Tr])).

sef(P,Q,M,N,Tot):- intersection(P,M,Y),intersection(Q,N,Z),
                sizeof(Y,0,N1),sizeof(Z,0,N2),
                Tot is (N1+N2).

sizeof([],Count,Counter):- Counter=Count.
sizeof([H|T],Count,Counter):- Count1 is Count+1, sizeof(T,Count1,Counter).

subset([H|T],[H|T]).
subset([],_).
subset(_,[]).
subset([H|T],Super):-member(H,Super),subset(T,Super).

intersection(_,[],[]).
intersection([],_,[]).
intersection([X|R],Y,[X|Z]):-member(X,Y),!,intersection(R,Y,Z).
intersection([X|R],Y,Z):-intersection(R,Y,Z).



                search(T3,Retcat,Retan,RetP,RetQ).

check_consistent(Rc,Ra,Mc,Na,Rtcat,Rtan):-
        (   (Rc==[],Ra==[],Rtcat=Mc,Rtan=Na);
            (Rc==[],Ra\==[],Rtcat=Mc,consistent(Rtan,Ra,Na));
            (Rc\==[],Ra==[],consistent(Rtcat,Rc,Mc),Rtan=Na);
            (Rc\==[],Ra\==[],consistent(Rtcat,Rc,Mc),consistent(Rtan,Ra,Na))
        ).


output_status(T,Retcat,Retan):-
                nl,write('List of remaining preliminary Tests= '),write(T),
                nl,write('List of possible cations= '),
                      ((Retcat==[],write([all]));write(Retcat)),
                nl,write('List of possible anions= '),
                      ((Retan==[],write([all]));write(Retan)),
                pause.
```

```
        nl,write('Tell me whether an yellowish white ppt is also formed?').
ask_next_q(_).

/* these rules are invoked by the interpreter which performs the test */
charcoal_test(P,Q,[if,the,colour,of,the,residue,when,hot,is,X1,and,&,when,cold,
is,X2,and,&,metallic,bead,is,X3,then,infer,X,Z,&,Tr1]):-
                    repeat,io_module(1,Y),
                    charcoal_cavity(Y,X1,X2,X3,X,Z),
                    cobalt_nitrate_cc(Y,X,Z,P,Q,Tr1).

cobalt_nitrate_test(P,Q):-cobalt_nitrate_cc(1,[],[],P,Q).

cobalt_nitrate_cc(M,A,B,C,D,[if,the,colour,of,the,residue,is,X1,then,infer,P,Q]
):-
        (M==1;M==8),
        repeat,io_module(2,Y),
        cobalt_nitrate(Y,X1,P,Q),
         (
         ((Y==1;Y==2;Y==3;Y==4),C=P,D=Q);
          (Y==5,C=A,D=B)
         ).
cobalt_nitrate_cc(M,A,B,A,B,'.').

flame_test(P,Q,[if,the,colour,of,the,flame,is,X1,with,naked,eye,and,&,X2,with,b
lue,glass,then,infer,P,Q]):- repeat,io_module(3,Y),
                    colour_flame(Y,X1,X2,P,Q).

identify_gas(X,Tr) :- repeat,
        io_module(4,Y),identify_class(Y,X,Tr).

identify_class(1,X,[if,the,gas,is,X1,and,X2,and,X3,then,infer,X]):-
        io_module(5,Z),
        Z>0,Z<3,gas_evolved(Y,X,X1,X2,X3,_,_).

identify_class(2,X,[if,the,gas,is,X1,and,X2,and,X3,then,infer,X]):-
        io_module(6,Y),
        Y>0,Y<7,Z is Y+2,gas_evolved(Z,X,X1,X2,X3,_,_).

identify_class(3,X,[if,the,gas,is,X1,and,X2,and,X3,then,infer,X]):-
        io_module(7,Y),
        Y>0,Y<5,Z is Y+8,
        gas_evolved(Z,X,X1,X2,X3,_,_).

dry_heating_test(M,N,Tr) :- repeat,readfile(dryheat_proc),
                            nl,prompt(In,>),reading(S),get0(C),
                            rm_duplicates(S,L),
                            perform(L,[],[],X,Y,Tr),
                            rm_duplicates(X,M),rm_duplicates(Y,N).

reading([H|T]):-get0(X),name(Y,[X]),read_check(Y,H,T),!.

read_check(P,Q,[]):-P=='.',Q=P,!.
read_check(P,Q,R):-P==',',reading([Q|R]).
read_check(P,Q,R):-integer(P),P>0,P<7,Q=P,reading(R).
read_check(P,Q,R):-atom(P),Q=P,reading(R).

perform([],P,Q,P,Q,'.').
perform([H|T],P,Q,C,D,[Tr,&,Tr1]) :-
                            do_head(H,A,B,Tr),append(A,P,M),append(B,Q,N),
                            perform(T,M,N,C,D,Tr1),!.

do_head('.',A,B,'.'):-!.
```

```
do_head(1,A,B,Tr):- id_gas_evolving(A,B,Tr).
do_head(2,A,B,Tr):- id_chn_residue(A,B,Tr).
do_head(3,A,B,Tr):- id_col_sublimate(A,B,Tr).
do_head(4,A,B,Tr):-id_noise(A,B,Tr).
do_head(5,A,B,Tr):- id_swelling(A,B,Tr).
do_head(6,A,B,Tr):- id_crystallization(A,B,Tr).
do_head(7,[],[],[]).
do_head(w,_,_,_):- write('no explanation !!'),nl,nl,fail.

append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

id_gas_evolving(M,N,[Tr,if,the,gas,evolved,is,X,then,infer,M,N]):-
        identify_gas(X,Tr),gas_evolved(_,X,_,_,_,M,N).

rm_duplicates([H|T],[H|N]) :- rm_dup_head(H,T,S),
                              rm_duplicates(S,N),!.
rm_duplicates([],[]).

rm_dup_head(X,Y,Z) :- (member(X,Y),delete(X,Y,Z));same(Y,Z).
same(A,A).

delete(_,[],[]).
delete(Y,[Y|L1],M) :- !,delete(Y,L1,M).
delete(Y,[X|L1],[X|L2]) :- delete(Y,L1,L2).

member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).

id_chn_residue(P,Q,[if,the,colour,of,the,residue,when,hot,is,X1,and,&,X2,when,c
old,then,infer,P,Q]) :-
                repeat,
                io_module(9,Y),
                chn_colour_residue(Y,X1,X2,P,Q).

id_col_sublimate(P,Q,[if,the,colour,of,the,sublimate,is,X1,then,infer,P,Q]):-
                repeat,
                io_module(10,Y),
                colour_sublimate(Y,X1,P,Q).

id_swelling(P,Q,Tr):- repeat,
                io_module(11,Y),dec(Y,P,Q,Tr).

dec(yes,['Na+','K+','Mg+2'],[],[if,there,is,fusion,then,infer,&,sodium,potassiu
m,and,magnesium,cations,&]).
dec(no,P,Q,[if,there,is,swelling,then,infer,P,Q]):-
        repeat,io_module(12,Z),dec1(Z,P,Q).

dec1(yes,[],['BO3-3','PO4-3']).
dec1(no,[],[]).

id_noise(P,Q,[if,there,is,crackling,noise,then,infer,P,Q]):-
        repeat,io_module(13,Y),deci(Y,P,Q).

deci(yes,['Na+','K+','Pb+2'],['NO3-','I-','Cl-']).
deci(no,[],[]).

id_crystallization(P,Q,[if,there,is,water,of,crystallization,then,infer,P,Q]):
        repeat,io_module(14,Y),decis(Y,P,Q).

decis(yes,[],['Cl-','NO3-','SO4-2']).
decis(no,[],[]).
```

```
syn('CO3-2',carbonate).
syn('HCO3-',bicarbonate).
syn('SO4-2',sulphate).
syn('S2O3-2',thiosulphate).
syn('CH3COO-',acetate).
syn('NO2-',nitrite).
syn('NO3-',nitrate).
syn('SO3-2',sulphite).
syn('S-2',sulphide).
syn('C2O4-2',oxalate).
syn('Cl-',chloride).
syn('Br-',bromide).
syn('I-',iodide).
syn('PO4-3',phosphate).

syn('Ag+',silver).
syn('Hg+2',mercury).
syn('Pb+2',lead).
syn('Bi+3',bismuth).
syn('Cu+2',copper).
syn('Cd+2',cadmium).
syn('As+3',arsenic).
syn('Sb+3',antimony).
syn('Sn+2',tin).
syn('Fe+3',iron).
syn('Cr+3',chromium).
syn('Al+3',aluminium).
syn('Co+2',cobalt).
syn('Ni+2',nickel).
syn('Mn+2',manganese).
syn('Zn+2',zinc).
syn('Ba+2',barium).
syn('Sr+2',strontium).
syn('Ca+2',calcium).
syn('Mg+2',magnesium).
syn('NH4+',ammonium).
syn('Na+',sodium).
syn('K+',potassium).


dil_sulphuric_acid_test(P,Q,Tr):- repeat,io_module(15,Y),decisi(Y,P,Q,Tr).

decisi(yes,P,Q,[on,adding,dilute,sulphuric,acid,&,Tr1,&,then,infer,P,Q]):-
        identify_gas(Gas,Tr1),dilh2so4(Gas,P,N),
        ((Gas==SO2,repeat,io_module(19,Ans),decisi1(Ans,Q));(Q=N)).

decisi(no,[],[]).
decisi1(yes,['S2O3-2']).
decisi1(no,['SO3-2']).

conc_sulphuric_acid_test(P,Q,[on,adding,conc,sulphuric,acid,&,Tr1,&,then,infer
P,Q]):- repeat,io_module(17,Y),decisio(Y,P,Q,Tr1).

decisio(yes,P,Q,Tr1):-identify_gas(Gas,Tr1),conch2so4(Gas,P,Q).
decisio(no,[],[]).

borax_bead_test(P,Q,[if,the,colour,of,the,bead,is,&,X1,when,hot,and,X2,when,co
d,in,the,oxidising,flame,and,X3,in,reducing,flame,then,infer,P,Q]):-
        repeat,io_module(18,Y),
        colour_bead(Y,X1,X2,X3,P,Q).
an(['CO3-2','SO4-2','S2O3-2','CH3COO-','NO2-','NO3-','SO3-2','S-2','C2O4-2','C
-','Br-','I-','PO4-3','BO3-3']).
```

```
lead([cooling,potassium_chromate]).
silver([nitric_acid,potassium_iodide,potassium_chromate]).
mercury([stannous_chloride,sodium_carbonate]).
bismuth([dilution,sodium_stannite,thiourea]).
copper([potassium_ferrocyanide]).
cadmium([potassium_ferrocyanide]).
arsenic([ammonium_molybdate,magnesia_mixture]).
atimony([dilution,tin_metal]).
tin([stannous_chloride,ammonium_molybdate]).
iron([potassium_ferrocyanide,potassium_sulphocyanide]).
chromium([lead_acetate,hydrogen_peroxide]).
aluminium([lake]).
cobalt([cobaltinitrite]).
nickel([dimethyl_glyoxime]).
manganese([bromine_water,pink_colour]).
zinc([sodium_hydroxide,potassium_ferrocyanide]).
barium([potassium_chromate]).
strontium([ammonium_sulphate]).
calcium([ammonium_oxalate]).
magnesium([ammonium_phosphate]).
sodium([potassium_pyroantimonate]).
potassium([sodium_cobaltinitrite,picric_acid]).
ammonium([sodium_hydroxide,nesslers_reagent]).



carbonate([magnesium_sulphate]).
bicarbonate([magnesium_sulphate]).
sulphate([barium_chloride]).
thiosulphate([silver_nitrate,ferric_chloride]).
acetate([ethyl_alcohol,ferric_chloride]).
nitrite([ferrous_sulphate,potassium_permanganate,potassium_iodide]).
nitrate([copper_turnings,ferrous_sulphate]).
sulphite([barium_chloride,ferric_chloride,potassium_permanganate]).
sulphide([cadmium_carbonate,lead_acetate,sodium_nitroprusside]).
oxalate([calcium_chloride,barium_chloride]).
chloride([silver_nitrate,manganese_dioxide,chromyl_chloride]).
bromide([silver_nitrate,manganese_dioxide,carbon_disulphide]).
iodide([silver_nitrate,manganese_dioxide,carbon_disulphide]).
phosphate([ammonium_molybdate,magnesia_mixture]).
borate([green_edged_flame,turmeric_paper]).

anions([carbonate,bicarbonate,sulphate,thiosulphate,acetate,nitrite,sulphite,s
lphide,oxalate,chloride,bromide,iodide]).

cations([silver,mercurous,lead,mercuric,bismuth,copper,arseneic,antimony,tin,i
on,chromium,aluminium,cobalt,nickel,manganese,zinc,barium,strontium,calcium,ma
nesium,ammonium,sodium,potassium]).

member(X,[X|_]):-!.
member(X,[_|Y]):-member(X,Y).

        E=..[Rad,Tests],E,asserta(rad(anion)),
```

```
/*rules for identifying a gas in inferencing*/
gas_evolved(1,'CO2',colourless,odourless,'turns lime water milky',[],['CO3-2','C2O4-2']).
gas_evolved(2,'CO',colourless,odourless,'burns with a blue flame',[],['C2O4-2']).
gas_evolved(3,'H2S',colurless,'smells like rotten eggs','turns lead acetate paper black',[],['S-2','S2O3-2']).
gas_evolved(4,'SO2',colourless,'characteristic suffocating smell','turns,acidified K2Cr2O7 paper green',[],['SO3-2','S2O3-2']).
gas_evolved(5,'HCl',colourless,'pungent smell','white fumes with ammonia white ppt with silver nitrate solution',[],['Cl-']).
gas_evolved(6,'CH3COOH',colourless,'characteristic','vinegar like smell',[],['CH3COO-']).
gas_evolved(7,'CH3COCH3',colourless,'sweet smell','vapours catch fire',[],['CH3COO-']).
gas_evolved(8,'NH3',colourless,'characteristic smell','turns moist turmeric paper brown',['NH4+'],[]).
gas_evolved(9,'NO2','reddish brown','pungent smell','turns FeSO4 solution black',[],['NO2-','NO3-']).
gas_evolved(10,'Cl2','greenish yellow','pungent smell','turns starch iodide paper blue',[],['Cl-']).
gas_evolved(11,'Br2','reddish brown','pungent smell','turns starch paper orange yellow',[],['Br-']).
gas_evolved(12,'I2','dark violet','pungent smell','turns starch paper orange yellow',[],['I-']).

/* rules for dry heat test */
chn_colour_residue(1,yellow,white,['Zn+2','Sn+2'],[]).
chn_colour_residue(2,brown,brown,['Cd+2'],[]).
chn_colour_residue(3,brown,yellow,['Pb+2','Bi+3'],[]).
chn_colour_residue(4,blue,white,['Cu+2'],['SO4-2']).
chn_colour_residue(5,violet,green,['Cr+3'],[]).
chn_colour_residue(6,pink,blue,['Co+2'],[]).
chn_colour_residue(7,green,yellow,['Ni+2'],[]).
chn_colour_residue(8,'pale brown',black,['Fe+3'],[]).
chn_colour_residue(9,'light green','reddish brown',['Fe+2'],[]).
chn_colour_residue(10,'coloured salt','black residue',['Co+2','Mn+2','Fe+2','Cu+2','Ni+2'],[]).

colour_sublimate(1,white,['NH4+','Hg+2','As+3'],[]).
colour_sublimate(2,yellow,['As+3'],['S-2']).
colour_sublimate(3,'grey with metal globules',['Hg+2'],[]).
colour_sublimate(4,black,['Hg+2'],['S-2','I-']).

/* rules for charcoal cavity test */
charcoal_cavity(1,yellow,white,none,['Zn+2','Sn+2'],[]).
charcoal_cavity(2,brown,brown,none,['Cd+2'],[]).
charcoal_cavity(3,brown,yellow,'greyish bead which marks paper',['Pb+2'],[]).
charcoal_cavity(4,orange,yellow,'pinkish brittle bead',['Bi+3'],[]).
charcoal_cavity(5,none,none,'read beads or scales',['Cu+2'],[]).
charcoal_cavity(6,none,'white but does not mark paper',none,['Ag+2'],[]).
charcoal_cavity(7,white,greyish,white,none,['Sb+3'],[]).
charcoal_cavity(8,'glowing white residue',none,none,['Ba+2','Ca+2','Mg+2'],[])
charcoal_cavity(9,black,none,none,[],[]).

/* rules for cobalt nitrate test */
cobalt_nitrate(1,green,['Zn+2'],[]).
cobalt_nitrate(2,'dirty green',[Sn+2],[]).
cobalt_nitrate(3,pink,['Mg+2'],[]).
cobalt_nitrate(4,blue,['Al+3'],['PO4-3','BO3-3']).
```

```
cobalt_nitrate(5,black,[],[]).

/* rules for flame test */
colour_flame(1,'gloden yellow',invisible,['Na+'],[]).
colour_flame(2,'pale violet',pinkish,['K+'],[]).
colour_flame(3,'bluish green or blue',visible,['Cu+2'],[]).
colour_flame(4,crimson,crimson,['Sr+2'],[]).
colour_flame(5,'brick red','light yellow',['Ca+2'],[]).
colour_flame(6,'grassy green',green,['Ba+2'],[]).
colour_flame(7,'bluish white',none,['As+3','Sb+3','Pb+2'],[]).
colour_flame(8,'no colour',none,[],[]).

/* rules for borax bead test */
colour_bead(1,'deep blue','deep blue','deep blue',['Co+2'],[]).
colour_bead(2,green,green,green,['Cr+3'],[]).
colour_bead(3,green,blue,'reddish opaque or colourless',['Cu+2'],[]).
colour_bead(4,'reddish yellow','pale yellow',green,['Fe+2','Fe+3'],[]).
colour_bead(5,'light pinkish','light pinkish',none,['Mn+2'],[]).
colour_bead(6,'reddish brown','reddish brown',none,['Ni+2'],[]).
colour_bead(7,none,none,none,[],[]).

/* rules for dilute sulphuric acid test */
dilh2so4('CO2',[],['CO3-2']).
dilh2so4('SO2',[],['SO3-2']).
dilh2so4('SO2',[],['S2O3-2']).
dilh2so4('H2S',[],['S-2']).
dilh2so4('NO2',[],['NO2-']).

/* rules for concentrate sulphuric acid test */
conch2so4('CO',[],['C2O4-2']).
conch2so4('HCl',[],['Cl-']).
conch2so4('Br2',[],['Br-']).
conch2so4('I2',[],['I-']).
conch2so4('CH3COOH',[],['CH3COO-']).
conch2so4('NO2',[],['NO3-']).

dry_heating_test(13,12,25,['Zn+2','Sn+2','Cd+2','Pb+2','Bi+3','Co+2','Cu+2','N
+2','Fe+3','Fe+2','NH4+','Hg+2','As+3'],['CO3-2','C2O4-2','S-2','S2O3-2','SO3-
','SO4-2','CH3COO-','NO2-','NO3-','Cl-','Br-','I-']).
charcoal_test(12,2,14,['Zn+2','Sn+2','Cd+2','Pb+2','Bi+3','Cu+2','Ag+2','Sb+3'
'Ba+2','Al+3','Ca+2','Mg+2'],['PO4-3','BO3-3']).
flame_test(9,0,9,['Na+','K+','Cu+2','Sr+2','Ca+2','Ba+2','As+3','Sb+3','Pb+2']
[]).
borax_bead_test(7,0,7,['Co+2','Cr+3','Cu+2','Fe+2','Fe+3','Mn+2','Ni+2'],[]).
dil_sulphuric_acid_test(0,6,6,[],['CO3-2','SO3-2','S2O3-2','S-2','NO2-']).
conc_sulphuric_acid_test(0,6,6,[],['C2O4-2','Cl-','Br-','I-','CH3COO-','NO3-']
.
```

```
/* input output module which asks a question and accepts an input */
pause_fail:-nl,write(ok),skip(10),fail.
pause:-nl,write(ok),skip(10).

io_module(Ctxt,Y):-      ask(Ctxt),
                         nl,prompt(In,>),read(Y),
                         checkingY(Ctxt,Y).

checkingY(_,how):-ans_how.
checkingY(_,help):-write('no help !'),fail.
checkingY(_,Y):-integer(Y),!.
checkingY(Ct,why):-map_w(Ct),pause_fail.
checkingY(_,yes).
checkingY(_,no).

ans_how:-((how(X),displist(X),pause,fail);true),!,fail.
how:-ans_how.

displist([[[H|T]|T1]|T2]):-     write(H),tab(1),displist(T),nl,
                                displist(T1),nl,displist(T2),!.
displist([[H|T]|T1]):-nl,write(H),tab(1),displist(T),nl,displist(T1),!.
displist([&|T]):-nl,displist(T),!.
displist([]):-!.
displist([H|T]):-write(H),tab(1),displist(T),!.

/* a why qustion by the user maps against one of the contexts */
map_w(1):-readfile(charcoal_why).
map_w(2):-readfile(cobalt_why).
map_w(3):-readfile(flame_why).
map_w(4):-readfile(gas_why3).
map_w(5):- write(' A COLOURLESS and ODOURLESS gas which'),nl,
           tab(5),write('* turns lime water milky is CO2'),nl,
           tab(5),write('* burns with a blue flame is CO').
map_w(6):-readfile(gas_why1).
map_w(7):-read_file(gas_why2).
map_w(9):-readfile(residue_why).
map_w(10):-readfile(sublimate_why).
map_w(11):-nl,
          write('Many alkali metal salts contain a large quantity of '),
          nl,write('water as water of crystallization. On heating the'),
          nl,write('water separates and dissolves the salt.').
map_w(12):-nl,




      write('Normally Phosphates,Borates and alums swell on heating').
map_w(13):-nl,
  write('Some anhydrous salts like lead nitrate,potassium iodide,sodium'),nl,
  write('chloride etc contain some mother liquor entrapped in their'),nl,
  write('crystals. On heating the mother liquor escapes by breaking the'),nl,
  write('crystals. This results in crackling noise.').
map_w(14):-nl,
      write('Hydrated salts on heating lose water of crystallization'),nl,
      write('which condenses on the cooler parts of the test tube '),nl,
      write('Most of the hydrated salts contain chloride or nitrate'),nl,
      write('or sulphate as an acid radical').
map_w(15):-readfile(dil_why).
map_w(16):-nl,write('Formation of white ppt with the liberation '),
                    write('of SO2  indicates S2O3-2 otherwise SO3-2').
map_w(17):-readfile(conc_why).
map_w(18):-readfile(borax_why).
map_w(19):-write('If the gas evolved is SO2 and no ppt is formed then'),nl,
      write('SO3-2 is to be inferred. If an yellowish white ppt is'),nl,
      write('also formed S2O3-2 is to be inferred').
map_w(20):-write('I thought you wouuld be interested in that').
```

```
ask_next_q(1):-readfile(charcoal_proc).
ask_next_q(2):-readfile(cobalt_proc).
ask_next_q(3):-readfile(flame_proc).
ask_next_q(4):- nl,write('Tell me whether the evolving gas is a '),
        nl,tab(5),write('1.  colourless and odourless gas'),
        nl,tab(5),write('2.  colourless gas with odour'),
        nl,tab(5),write('3.  coloured gas with pungent smell').
ask_next_q(5):-
        nl,write('Tell me whether the gas '),
        nl,tab(10),write('1.  turns lime water milky'),
        nl,tab(10),write('2.  burns with a blue flame').
ask_next_q(6):-nl,readfile(gas_proc1).
ask_next_q(7):-readfile(gas_proc2).
ask_next_q(9):-readfile(chn_residue).
ask_next_q(10):-nl,tab(5),write('Colour of the Sublimate'),
                nl,tab(10),write('1. white'),
                nl,tab(10),write('2. Yellow'),
                nl,tab(10),write('3. Grey with'),
                nl,tab(16),write('metal globules'),
                nl,tab(10),write('4. Black'),nl.
ask_next_q(11):-nl,write('Is there a fusion (or melting) of the salt?').
ask_next_q(12):-nl,write('Is there a swelling of the salt?').
ask_next_q(13):-nl,write('Is there a crackling noise on heating the salt?').
ask_next_q(14):-
     nl,write('Tell me whether there is condensation of H2O on the cooler'),
     nl,write('walls of the test tube?').
ask_next_q(15):- nl,
    write('Take a little of the salt in a clean test tube.Treat it'),nl,
    write('with a few ml of dilute sulphuric acid.Warm if no gas is evolved'),
    nl,write('Tell me whether a gas is being evolved?').
ask_next_q(17):-nl,
     write('Take a little of the salt in a test tube and treat it with'),nl,
     write('a few ml of concentrated sulphuric acid. Heat the contents '),nl,
     write('if no gas is evolved'),nl,
     write('Tell me whether a gas is being evolved?'),!.
ask_next_q(18):-readfile(borax_proc).
ask_next_q(19):-



        nl,write('Tell me whether an yellowish white ppt is also formed?').
ask_next_q(_).
```

```
dil_sulphuric_acid_test(P,Q,Tr):- repeat,io_module(15,Y),decisi(Y,P,Q,Tr).

decisi(yes,P,Q,[on,adding,dilute,sulphuric,acid,&,Tr1,&,then,infer,P,Q]):-
        identify_gas(Gas,Tr1),dilh2so4(Gas,P,N),
        ((Gas==SO2,repeat,io_module(19,Ans),decisi1(Ans,Q));(Q=N)).

decisi(no,[],[]).
decisi1(yes,['S2O3-2']).
decisi1(no,['SO3-2']).

conc_sulphuric_acid_test(P,Q,[on,adding,conc,sulphuric,acid,&,Tr1,&,then,infe
P,Q]):- repeat,io_module(17,Y),decisio(Y,P,Q,Tr1).

decisio(yes,P,Q,Tr1):-identify_gas(Gas,Tr1),conch2so4(Gas,P,Q).
decisio(no,[],[]).

borax_bead_test(P,Q,[if,the,colour,of,the,bead,is,&,X1,when,hot,and,X2,when,c
d,in,the,oxidising,flame,and,X3,in,reducing,flame,then,infer,P,Q]):-
        repeat,io_module(18,Y),
        colour_bead(Y,X1,X2,X3,P,Q).
an(['CO3-2','SO4-2','S2O3-2','CH3COO-','NO2-','NO3-','SO3-2','S-2','C2O4-2','
-','Br-','I-','PO4-3','BO3-3']).
```

```
/* reads a file */
readfile(X):-seeing(Old),see(X),readline(0),see(Old),!.

readline(Cr):-read_in(S,C),C1 is Cr+1,
        (((C==26;C==4),seen,!);
         (C1==23,nl,write('You want more?'),tab(2),
                seeing(Old),see(user),read(Ans),see(Old),
                ((Ans==yes,((nonvar(S),write(S));true),readline(0));
                 (seen,!)
                )
         );
         (var(S),nl,readline(C1));
         (write(S),nl,readline(C1))
        ).

read_in(W,C2):- get0(C),readword(C,W,C2).

readword(C,W,C2):- inword(C,NewC),!,get0(C1),
                   restword(C1,Cs,C2),name(W,[NewC|Cs]).
readword(C,W,C).

restword(C,[NewC|Cs],C2):- inword(C,NewC),!,get0(C1),
                           restword(C1,Cs,C2).
restword(C,[],C).

inword(C,C):- C>31,C<127.
inword(C,C):-C==9.

/* a small interface to access the knowledge in tests */
charcoal_cavity_test:-charcoal_test.

charcoal_test:-charcoal_test(P,Q,T),disp(P,Q).

disp(P,Q):-displist(['List',of,possible,'cation(s) =',P]),nl,
           displist(['List',of,possible,'anions(s) =',Q]).

cobalt_nitrate_test:-cobalt_nitrate_test(P,Q),disp(P,Q).

flame_test:-flame_test(P,Q,T),disp(P,Q).

identify_gas:-identify_gas(X,T),
        displist(['List',of,possible,'anions(s)',=,X]).


dry_heating_test:-dry_heat_test.

dry_heat_test:-dry_heating_test(P,Q,T),disp(P,Q).

dry_heat_test(M,N):-dry_heating_test(M,N,T).

dilute_sulphuric_acid_test:-dil_sulphuric_acid_test.
dil_H2SO4_test:-dil_sulphuric_acid_test.
dilute_H2SO4_test:-dil_sulphuric_acid_test.
dil_sulphuric_acid_test:-dil_sulphuric_acid_test(P,Q,T),disp(P,Q).

concentrated_H2SO4_test:-conc_sulphuric_acid_test.
conc_H2SO4_test:-conc_sulphuric_acid_test.
concentrated_sulphric_acid_test:-conc_sulphuric_acid_test.
conc_sulphuric_acid_test:-conc_sulphuric_acid_test(M,N,T),disp(P,Q).

borax_test:-borax_bead_test.
borax_bead_test:-borax_bead_test(P,Q,T),disp(P,Q).
```

```
confirm_cation(X):-
                synonym(X,Rad),asserta(rad(cation)),
                find_group(Rad,Gr),!,salt_sol,
                Ex=..[Gr,Ret],Ex,test(T),retract(rad(cation)),
                ((var(Ret),assertz(result(confirm,T,no)),!,fail),
                 (Ret\==no,assertz(result(confirm,T,Ret)));
                 (assertz(result(confirm,T,no)),true)
                ),displist([Ret,is,confirmed]).

confirm_anion(X):-
                synonym(X,Rad),
                get_tests(Rad,Tests),
                display_select(Tests,1,Inp,Z),
                asserta(test(Z)),salt_solution(Rad),
                io_module(Rad,Z,Ret),retract(rad(anion)),
                (   (     (Ret==yes,assertz(result(confirm,Z,X)),
                           displist([X,is,confirmed]));
                          (X==carbonate;X==bicarbonate,
                           assertz(result(confirm,Z,Ret)),
                           displist([Z,is,confirmed]))
                   );(assertz(result(confirmed,Z,no)),!,fail)).

conf(X):-       E=..[X,Tests],E,
                (X==copper;X==cadmium;X==aluminium;
                 X==cobalt;X==nickel;X==sodium;
                 X==barium;X==calcium;X==magnesium;
                 X==strontium;write_any(X)
                ),display_select(Tests,1,Inp,Z),
                asserta(test(Z)),
                io_module(X,Z,Ret),
                ((Ret==yes,!);(!,fail)).

synonym(X,Rad):-        ((syn(X,Y),asserta(radical(Y)));
                         (syn(Y,X),asserta(radical(X)))
                        ),radical(Rad).
get_tests(Rad,Tests):-
        E=..[Rad,Tests],E,asserta(rad(anion)),
        (Rad==carbonate;Rad==bicarbonate;Rad==sulphate;write_any(Rad)).

display_select([X],1,1,X).
display_select([],Ctr,In,Y):-repeat,nl,prompt(Ini,>),read(In),
                        ((integer(In),In>0,In<Ctr);
                         (write('*** BAD INPUT ***'),fail)).
display_select([H|T],Ctr,In,Y):-
                        displist([Ctr,H,test]),nl,
                        Ct is Ctr+1,
                        display_select(T,Ct,In,Y),
                        ((Ctr==In,Y=H);!).

salt_sol:-readfile(salt_proc),pause.

find_group(H,Gr):-t(P,Q,R),lookup(H,t(P,Q,R),Gr).

lookup(H,t(Gr,G,_),Gr):-member(H,G),!,asserta(group(Gr)).
lookup(H,t(_,_,Next),R):-nonvar(Next),lookup(H,Next,R).

get_procedure(Radic,X1,X2,List):-
                        get_two_lists(Radic,List,[X1|[X2]]).
```

```
get_two_lists(Radi,[[Radi|T]|T1],T).
get_two_lists(Radi,[[H|T]|T1],T2):-get_two_lists(Radi,T1,T2).

t(groupI,[silver,mercury,lead],
   t(groupIIA,[mercury,lead,bismuth,copper,cadmium],
      t(groupIIB,[arsenic,antimony,tin],
         t(groupIII,[iron,chromium,aluminium],
            t(groupIV,[cobalt,nickel,manganese,zinc],
               t(groupV,[barium,strontium,calcium],
                  t(groupVI,[magnesium,ammonium,sodium,potassium],_)
                 )
              )
           )
        )
     )
 ).

execute([]):-!.
execute([H|T]):-H,execute(T).
salt_solution(Rad):-     salt(Rad);aqueous;sce.

salt(Rad):-              ((Rad==carbonate;Rad==bicarbonate;Rad==borate;
                          (test(T),radical(An),!,
                               ((An==acetate,T==ethyl_alcohol);
                                (T==manganese_dioxide);
                                (T==chromyl_chloride);
                                (An==nitrate,T==copper_turnings)
                               )
                           )),asserta(soluble_in(salt))
                         ).

aqueous:-repeat,
        nl,write('Mix a little of the salt in water and'),nl,
        write_tell('dissolution,of,the,salt,in,water'),!,
        nl,prompt(INi,>),read(Ans),
        ((Ans==yes,asserta(soluble_in('SALT SOLUTION')));(!,fail)).

sce:-readfile(extract),pause,asserta(soluble_in('SODIUM CARBONATE EXTRACT')).
```

```
potassium_chromate([     [lead,[above,solution],[yellow,ppt]],
                         [silver,[original,solution],[brick,red,ppt]],
                         [barium,[above,solution],[yellow,ppt]]
                   ]).

potassium_iodide([[lead,[above,solution],[yellow,ppt]],[silver,[original,solut


on],[yellow,ppt]],[nitrite,[asserta(acid('H2SO4')),write_ex,write('after addin
 a drop of starch solution'),nl],['BLUE COLOURATION']]]).

nitric_acid([   [silver,[above,solution],[white,ppt]]]).

stannous_chloride([[mercury,[original,solution],[white,ppt,turning,grey]],
                [tin,[solution,obtained,by,dissolving,the,above,ppt,in,'
                      conc HCl',add,a,few,pieces,of,zinc,metal,and,then],
                        [white,ppt,turning,grey]]]).

copper_turnings([[mercury,[original,solution],[silvery,deposit,on,cu,chips]],
[nitrate,[nl,write('Heat 0.5 gms of the salt with 2ml of conc H2SO4 and '),nl,w
rite('add a few Cu chips.'),nl],['DENSE REDDISH FUMES',of,nitrogen,peroxide]]]
.

sodium_stannite([[bismuth,[above,ppt],[black,ppt]]]).

thiourea([[bismuth,[original.solution,in,dilute,'HCl',add,2,drops,of,dilute,'
HNO3',and,then],[yellow,colouration]]]).

potassium_ferrocyanide([[copper,[above,solution,add,a,little,'CH3COOH',and,then
],[chocolate,ppt]], [cadmium,[above,solution],[bluish,white,ppt]], [iron,[solut
ion,obtained,by,dissolving,the,above,ppt,in,dilute,'HCl'],[prussian,blue]], [zi
nc,[original,solution],[bluish,white,ppt]]]).

ammonium_molybdate([[arsenic,[solution,obtained,by,dissolving,the,above,ppt,in,
conc,'HNO3'],[yellow,ppt]],[tin,[solution,obtained,by,dissolving,the,above,ppt,
in,dilute,'HCl',add,a,few,pieces,of,zinc,metal,and,then],[deep,blue,colouration
]],[phosphate,[asserta(acid('concentrated HNO3')),write_extract],['YELLOW PPT',
or,'COLOURATION']]]).

magnesia_mixture([[arsenic,[solution,obtained,by,dissolving,the,above,ppt,in,co
nc,'HNO3',add,'NH4OH',and,a,pinch,of,'NH4Cl',and,'MgSO4',solution],[white,ppt]]
,[phosphate,[asserta(acid('dilute CH3COOH')),write_extract,nl,write('Also add e
xcess of NH4OH')],['WHITE PPT']]]).

dilution([[bismuth,[solution,obtained,by,dissolving,the,above,ppt,add,excess,of
,water],[milkiness]],[antimony,[solution,obtained,by,dissolving,the,above,ppt,i
n,conc,'HCl',add,excess,of,water],[milkiness]]]).

tin_metal([[antimony,[solution,obtained,dissolving,the,above,ppt],[black,deposi
t,on,this,metal]]]).

potassium_sulphocyanide([[iron,[solution,obtained,by,dissolving,the,above,ppt,i
n,dilute,'HCl'],[blood,red,colouration]]]).

lead_acetate([[chromium,[solution,obtained,by,extracting,the,above,ppt,and,'NaO
H',and,'NaNO3',with,water],[yellow,ppt,soluble,in,'NaOH']]]).

hydrogen_peroxide([[chromium,[solution,obtained,by,extracting,the,above,ppt,and
,'NaOH',and,'NaNO3',with,water,and,add,2,drops,of,'H2SO4',and,1,ml,of,ether,and
],[blue,colour,in,the,ether,layer]]]).

ammonium_hydroxide([[aluminium,[solution,obtained,by,dissolving,the,above,ppt,i
n,dilute,'HCl',add,a,few,drops,of,blue,litmus,and],[blue,ppt,floating,in,the,co
lourless,solution]]]).
```

```
cobaltinitrite([[cobalt,[original,solution,add,a,pinch,of,'NH4Cl',and,'NH4OH',a
nd,add,a,pinch,of,'KNO2',and,a,few,drops,of,acetic,acid,and,shake,well],[yelow,
ppt]]]).

dimethyl_glyoxime([[nickel,[original,solution,add,a,pinch,of,'NH4Cl',and,a,few,

drops,of,'NH4OH'],[bright,red,ppt]]]).

bromine_water([[manganese,[original,solution,add,'NaOH',solution,till,a,ppt,res
ults,and,shake,after],[brown,ppt]]]).

sodium_hydroxide([[zinc,[original,solution],[white,ppt,soluble,in,excess,of,'Na
OH']]]).

ammonium_sulphate([[strontium,[solution,obtained,by,dissolving,the,above,ppt,in
,hot,dilute,acetic,acid],[white,ppt]]]).

ammonium_oxalate([[calcium,[solution,obtained,by,dissolving,the,above,ppt,in,hot
,dilute,acetic,acid],[on,scratching,the,sides,of,the,tests,tube,after,adding,'N
H4OH',a,white,ppt]]]).

ammonium_phosphate([[magnesium,[original,solution,add,a,pinch,of,'NH4Cl',and,ex
cess,of,'NH4OH',and,then],[white,ppt]]]).

potassium_pyroantimonate([[sodium,[original,solution],[white,ppt,or,milkiness,o
n,scratching,the,sides,of,the,test,tube]]]).

sodium_cobaltinitrite([[potassium,[original,solution],[yellow,ppt]]]).

picric_acid([[potassium,[original,solution],[yellow,ppt]]]).

tartaric_acid([[potassium,[original,solution],[white,ppt,on,scratching,the,side
s]]]).
```

```
silver_nitrate([[thiosulphate,[write_sol,nl],[white,ppt,which,changed,to,yello
,orange,brown,and,finally,black]],[chloride,[asserta(acid('dilute HNO3')),writ
```



```
_extract],['CURDY WHITE PPT',soluble,in,'NH4OH']],[bromide,[asserta(acid('dilu
e HNO3')),write_extract],['PALE YELLOW PPT',partially,soluble,in,'NH4OH']],[io
ide,[asserta(acid('dilute HNO3')),write_extract],['YELLOW PPT',insoluble,in,'N
4OH']]]).

ferric_chloride([[thiosulphate,[write_sol],['PURPLE or VIOLET',colour,which,fa
es,away,on,standing]],[acetate,[write_sol,write('Red colour results'),nl,write
'Dilute it with 2ml of water and boil'),nl],['BROWN PPT']],[sulphite,[write_so
],[dark,'RED COLOURED',solution]]]).

ethyl_alcohol([[acetate,[write_mno],['FRUITY SMELL']]]).

ferrous_sulphate([[nitrite,[asserta(acid('CH3COOH')),write_ex],['DARK BROWN',o
,'BLACK',solution]],[nitrate,[write_sol,write('Now add conc. H2SO4 by the side
 of the test tube'),nl],['DARK BROWN RING',at,the,junction,of,the,two,layers]]
).

potassium_permanganate([[nitrite,[write_sol],[disappearing,of,the,'PINK', colo
r,of,'KMnO4']],[sulphite,[write_sol],[disappearing,of,the,'PINK',colour,of,'KM
O4']]]).

barium_chloride([[sulphate,[write_sol],['WHITE PPT',insoluble,in,dilute,'HCl']
,[sulphite,[write_sol,write('Filter the ppt and treat the residue with dilute 
Cl.'),nl],['DISSOLUTION',of,the,ppt,with,the,evolution,of,'SO2']],[oxalate,[wr
te_extract],['WHITE PPT']]]).

sodium_nitroprusside([[sulphide,[write_s],['PURPLE or VIOLET',colour]]]).

cadmium_carbonate([[sulphide,[retrieve(S,T),displist(['To','3ml',of,S,add,a,li
tle,solid,T]),nl],['YELLOW PPT']]]).
lead_acetate([[sulphide,[asserta(acid('dilute CH3COOH')),write_extract],['BLAC
 PPT']]]).

calcium_chloride([[oxalate,[asserta(acid('dilute CH3COOH')),write_extract],['W
ITE PPT']]]).

manganese_dioxide([[chloride,[write_mno],['Cl2',gas,which,turns,starch,iodide,
aper,blue]],[bromide,[write_mno],['Br2',gas,which,turns,starch,paper,yellow]],
iodide,[write_mno],['I2',vapour,which,turns,starch,paper,blue]]]).

chromyl_chloride([[chloride,[readfile(cct)],[yellow,ppt]]]).

carbon_disulphide([[bromide,[asserta(acid('dilute HCl')),write_extract,write('
lso add a few drops of Cl2 water and shake well.'),nl],['ORANGE',colour,in, 'C
2',layer]],[iodide,[asserta(acid('dilute HCl')),write_extract,write('Also add 
 few drops of Cl2 water and shake well.'),nl],['VIOLET',colour,in,'CS2',layer]
]).

green_edged_flame([[borate,[asserta(test('2-3 ml of ethyl alcohol')),write_mno
,[vapours,burning,with,'GREEN EDGED FLAME']]]).

turmeric_paper([[borate,[write('Take 0.5 gms of the salt.Make its solution wit
 dilute HCl'),nl,write('and soak a turmeric paper in it.On drying the paper be
omes'),nl,write('brown in colour.Now touch the paper with a drop of NaOH')],[a
'DIRTY BLUE',or,'GREENISH SPOT']]]).

magnesium_sulphate([[X,[write('Shake 0.5 gms of the salt with distilled water.
```

```
io_module(Radical,Test,Y):-     repeat,
                                ((rad(anion),ask_n(Radical,Test));
                                 (rad(cation),ask_c(Radical,Test))
                                ),nl,prompt(In,>),read(Y),




                                chek(Radical,Test,Y).

chek(Rad,Ct,why):-((rad(anion),!,map_Y(Rad));
                    (rad(cation),!,ans_why(Rad,Ct))
                   ),pause_fail.
chek(_,_,how):-ans_how,!,fail.
chek(_,_,Int):-integer(Int),!.
chek(_,_,yes).
chek(_,_,no).

ask_n(Rad,Tes):-          E=..[Tes,List],E,
                          get_procedure(Rad,X1,X2,List),
                          execute(X1),
                          write('Tell me whether'),nl,displist(X2),nl,
                          write('is observed?').


ask_c(Rad,Test):-         (Test==magnesia_mixture;Test==dilution),
                          E=..[Test,List],E,
                          get_procedure(Rad,X1,X2,List),
                          displist(X1),nl,write_tell(X2),!.

ask_c(Rad,Test):-         E=..[Test,List],E,
                          get_procedure(Rad,X1,X2,List),
                          write('To 2 ml of the  '),
                          displist(X1),nl,write('add a little of  '),
                          write(Test),tab(1),nl,
                          write_tell(X2),!.

ask_c(groupI,1):-
        nl,write('To the SALT SOLUTION add a few ml of dilute HCl'),nl,nl,
        write_tell('WHITE PPT').
ask_c(groupI,2):-
        nl,write('Boil a part of the white residue with a little of water'),
        nl,nl,write_tell('DISSOLUTION of the ppt').
ask_c(groupI,3):-
        nl,write('Filter and treat the residue with NH4OH and shake'),nl,
        write_tell('dissolution of the ppt').

ask_c(groupIIA,1):-readfile(groupII_proc).
ask_c(groupIIA,2):-
        write('Treat a pinch of groupII ppt with yellow ammonium sulphide'),
        nl,nl,write('Tell me whether the ppt remained INSOLUBLE?').
ask_c(groupIIA,3):-
        nl,write('Tell me the colour of the groupII ppt'),
        nl,tab(5),write('1. Black'),
        nl,tab(5),write('2. yellow'),
        nl,tab(5),write('3. none').
ask_c(groupIIA,4):-
        write('Boil the black ppt with 3-4ml of dil HNO3(50%)'),nl,
        write('Tell me whether the ppt remained INSOLUBLE?').
ask_c(groupIIA,5):-
        nl,write('To 1ml of the ABOVE SOLUTION add 2 drops of dil H2SO4'),
        nl,write_tell('WHITE PPT').
ask_c(groupIIA,6):-
        write('To the rest of the solution in dilute HNO3 add excess'),nl,
        write('of ammonium hydroxide'),nl,
        write('What is the colour of the ppt formed?'),
        nl,tab(5),write('1. white ppt').
```

```
ask_c(groupIIB,1):-readfile(groupII_proc).
ask_c(groupIIB,2):-
        write('Treat a pinch of groupII ppt with yellow ammonium sulphide'),
        nl,write('Tell me whether the ppt remained INSOLUBLE?').
ask_c(groupIIB,3):-     nl,tab(5),write('Tell me the colour of the ppt'),
                        nl,tab(5),write('1. yellow'),
                        nl,tab(5),write('2. orange'),
                        nl,tab(5),write('3. brown or dirty yellow ppt').

ask_c(groupIII,1):-readfile(groupIII_proc).
ask_c(groupIV,1):-readfile(groupIV_proc).
ask_c(groupIV,2):-
        nl,write('What is the colour of the original salt?'),nl,
        write('Indicate 1 for Pink and 2 for green or bluish green').

ask_c(groupV,1):-readfile(groupV_proc).
ans_why(groupV,1):-readfile(groupV_why).
ans_why(groupI,1):-readfile(groupI_why).
ans_why(groupI,2):-nl,
        write('If the ppt dissolves the cation may be lead or'),nl,
        write('else it can be either Ag+ or Hg2+2').
ans_why(groupI,3):-nl,
        write('If the ppt dissolves the cation may be Ag+ or'),nl,
        write('else it can be Hg2+2 if the ppt turns black').
ans_why(groupIIA,1):-readfile(groupII_why).
ans_why(groupIIA,2):-
   nl,write('If the ppt dissolves the cation belongs to groupIIB'),
   nl,tab(5),write('otherwise it belongs to groupIIA').
ans_why(groupIIA,3):-
   nl,write('If the colour of the ppt is'),nl,tab(5),
   write('black then  [Hg+2,Pb+2,Bi+2,Cu+2] may be present'),nl,tab(5),
   write('yellow then [Cd+2] may be present').
ans_why(groupIIA,4):-fail.
ans_why(groupIIA,5):-fail.
ans_why(groupIIA,6):-fail.

ans_why(groupIIB,3):-readfile(groupIIB_why).
ans-why(groupIII,1):-readfile(groupIII_why).
ans_why(groupIV,1):-readfile(groupIV_why).
ans_why(groupIV,2):-nl,
                    write('If the original colour is pink it indicates Co+2')
                    tab(26),write('green or bluish green it indicates Ni+2').
ans_why(Rad,_):-readfile(Rad).

map_Y(io2):- write('If the salt is insoluble, sodium carbonate extract has
e prepared'),!.
map_Y(io):-write('This menu selects the test requested by the user'),!.
    ''Y):-readfile(X).

              ') !.decy(X,Z.Y).
```

```
pause_fail:-nl,write(ok),skip(10),fail.
pause:-nl,write(ok),skip(10).
write_dis(X):-displist(['Tell',me,whether,the,X,dissolved,'?']).
write_disappear(X):-displist(['Tell',me,whether,X,'disappeared?']).
write_s:-retrieve(Z,X),displist(['To','2-3ml',of,Z,add,'1-2ml',of,X,'.']),nl.
write_sol:-
  retrieve(Z,X),displist(['To','2-3 ml',of,Z,add,a,few,drops,of,X,'.']),nl.
retrieve(S,T):-test(T),soluble_in(S).
write_extract:-retrieve(X,Z),acid(Y),
  displist(['Acidify','2 ml',of,X,with,Y,and,&,boil,off,'CO2',completely,&]),
  displist(['Add',to,it,1,ml,of,Z,solution]),nl.
write_any(X):-
 displist(['Any',of,the,following,tests,confirm,the,presence,of,X,radical]),
 nl,write('Indicate your choice by keying the number against the test'),nl.
 write_tell(X):-displist(['Tell',me,whether,'a(an)',X,is,'observed?']),nl.
write_mno:-test(T),
 displist(['Heat','0.5 gms',of,the,salt,with,'2ml',of,conc,'H2SO4',and,T,'.']
nl.
write_ex:-retrieve(X,Y),acid(Z),
displist(['Acidify','2ml',of,X,with,2,drops,of,dilute,Z,&,add,2,ml,of,Y]),nl.
```

```
confirm_cation:-confirm_groupwise.
confirm_groupwise:-confirm_groupwise(Y),write('CATION = '),write(Y).
confirm_groupwise(Y):-repeat,nl,
   write('Do you want to detect the cations groupwise?'),
   nl,prompt(In,>),read(Ans),
   readfile(salt_proc),pause,
((Ans==yes,groupI(Y1),
       ((Y1==no,write_no(groupI),groupIIA(Y3),
               ((Y3==no,write_no(groupII),groupIII(Y4),
                       ((Y4==no,write_no(groupIII),groupIV(Y5),
                               ((Y5==no,write_no(groupIV),groupV(Y6),
                                      ((Y6==no,write_no(groupV),groupV
                                               ((Y7==no,rem,!,fail)
                                               )
                                        );(Y=Y6)
                                       )
                                );(Y=Y5)
                               )
                        );(Y=Y4)
                       )
                );(Y=Y3)
               )
        );(Y=Y1)
       )
 );(Ans==no,fail)
).

rem:-displist(['The',cation,does,not,belong,to,groupVI,'.']),nl,nl,
 write('The cation does not belong to the list of cations detectable by

write_no(X):-displist(['The',cation,does,not,belong,to,X,'.']),nl.

groupI:-groupI(Y),nl,write('CATION = '),write(Y).

groupI(Y):-io_module(groupI,1,Z),anscheck(Z,Y).

anscheck(yes,Y):-io_module(groupI,2,Z),anscheck1(Z,Y).
anscheck(no,no).

anscheck1(yes,lead):-conf(lead).
anscheck1(no,Y):- io_module(groupI,3,Z),anscheck2(Z,Y).

anscheck2(yes,silver):-conf(silver).
anscheck2(no,mercury):-conf(mercury).


groupII:-groupIIA(Y),nl,write('CATION = '),write(Y).

groupIIA:-groupIIA(Y),nl,write('CATION = '),write(Y).

groupIIA(Y):-io_module(groupIIA,1,Z),ansck0(Z,Y).

ansck0(yes,Y):-io_module(groupIIA,2,Z),io_module(groupIIA,2,Z),ansck(Z,Y
ansck0(no,no).

ansck(no,Y):-write('groupIIA is absent.'),nl,nl,
  write('groupIIB is analysed by confirming As+3,Sb+3,Sn+2 individually'
  ask(no,Y).
ansck(yes,Y):-io_module(groupIIA,3,Z),ansck1(Z,Y).

ansck1(1,Y):-io_module(groupIIA,4,Z),ansck2(Z,Y).
ansck1(2,cadmium):-conf(cadmium).
ansck1(3,no).

ansck2(yes,mercury):-conf(mercury).
ansck2(no,Y):-  io_module(groupIIA,5,Z)...
```

```
ansck4(1,bismuth):-conf(bismuth).
ansck4(2,copper):-conf(copper).
ansck4(3,no).

groupIIB:-groupIIB(Y),nl,write('CATION = '),write(Y).

groupIIB(Y):-io_module(groupIIB,1,Z),ask0(Z,Y).

ask0(yes,Y):-io_module(groupIIB,2,Z),ask(Z,Y).
ask0(no,no).

ask(yes,Y):-ansck(yes,Y).
ask(no,Y):-io_module(groupIIB,3,Z),as(Z,Y).

as(1,Y):-((conf(arsenic),Y=arsenic);Y=no).
as(2,Y):-((conf(antimony),Y=antimony);Y=no).
as(3,Y):-((conf(tin),Y=tin);Y=no).
as(no,no).

groupIII:-groupIII(Y),nl,write('CATION = '),write(Y).

groupIII(Y):- io_module(groupIII,1,Z),verify(Z,Y).

verify(1,Y):-((conf(iron),Y=iron);Y=no).
verify(2,Y):-((conf(chromium),Y=chromium);Y=no).
verify(3,Y):-((conf(aluminium),Y=aluminium);Y=no).
verify(no,no).

groupIV:-groupIV(Y),nl,write('CATION = '),write(Y).




groupIV(Y):-io_module(groupIV,1,Z),veri(Z,Y).
veri(1,Y):-io_module(groupIV,2,Z),verif(Z,Y).
veri(2,Y):-((conf(manganese),Y=manganese);Y=no).
veri(3,Y):-((conf(zinc),Y=zinc);Y=no).
veri(no,no).

verif(1,Y):-((conf(cobalt),Y=cobalt);Y=no).
verif(2,Y):-((conf(nickel),Y=nickel);Y=no).

groupV:-groupV(Y),nl,write('CATION = '),write(Y).

groupV(Y):-io_module(groupV,1,Z),
        ((Z\==no,callconfirm(Y),Y\==no);
         (Y=no)
        ).

callconfirm(Y):-((conf(barium),Y=barium);
                 (conf(strontium),Y=strontium);
                 (conf(calcium),Y=calcium)
                ).

groupVI:-groupVI(Y),nl,write('CATION = '),write(Y).

groupVI(Y):-    repeat,groupVI_proc,
                ((callcon(Y),Y\==no);(Y=no)).

callcon(Y):-    ((conf(magnesium),Y=magnesium);
                 (conf(sodium),Y=sodium);
                 (conf(potassium),Y=potassium);
                 (conf(ammonium),Y=ammonium)
                ).
```

APPENDIX A

CATION	FORMULA	ANION	FORMULA
silver	Ag+	carbonate	CO3-2
mercurous	Hg2+2	bicarbonate	HCO3-
lead	Pb+2	sulphate	SO4-2
mercuric	Hg+2	thiosulphate	S2O3-2
bismuth	Bi+3	acetate	CH3COO-
copper	Cu+2	nitrite	NO2-
cadmium	Cd+2	nitrate	NO3-
arsenic	As+3	sulphite	SO3-2
antimony	Sb+3	sulphide	S-2
tin	Sn+2	oxalate	C2O4-2
iron	Fe+3	chloride	Cl-
chromium	Cr+3	bromide	Br-
aluminium	Al+3	iodide	I-
cobalt	Co+2	phosphate	PO4-3
nickel	Ni+2		
manganese	Mn+2		
zinc	Zn+2		
barium	Ba+2		
strontium	Sr+2		
calcium	Ca+2		
magnesium	Mg+2		
ammonium	NH4+		
sodium	Na+		
potassium	K+		

APPENDIX B

NAME OF THE TEST	DETECTABLE CATIONS	DETECTABLE ANIONS	TOTAL NO
dry_heat_test	Zn+2,Sn+2,Cd+2,Pb+2 Bi+3,Co+2,Cu+2,Ni+2 Fe+3,Ni+3,NH4+,Hg+2 As+3	CO3-2,HCO3-,C2O4-2,S-2 CH3COO-,SO4-2,S2O3-2 SO3-2,NO2-,NO3-,Cl- Br-,I-	26
charcoal_test	Zn+2,Sn+2,Cd+2,Pb+2 Bi+3,Cu+2,Ag+2,Sb+3 Ba+2,Al+3,Ca+2,Mg+2	PO4-3,BO3-3	14
flame_test	Na+,K+,Cu+2,Sr+2 Ca+2,Ba+2,As+3,Sb+3 Pb+2		9
borax_bead_test	Co+2,Cr+3,Cu+2,Fe+2 Fe+3,Mn+2,Ni+2	-	7
dil_h2so4_test	-	CO3-,HCO3-,SO3-2 S2O3-2,S-2,NO2-	6
conc_h2so4_test	-	C2O4-2,Cl-,Br- I-,CH3COO-,NO3-	6

APPENDIX C

GROUP NO	CATION	GROUP REAGENT	GROUP PRECIPITATE
I	Ag+,Hg2+2,Pb+2	Dilute HCl	Metal chloride
IIA	Hg+2,Pb+2,Bi+3 Cu+2,Cd+2	H2S gas in solution made acidic with HCl	Metal sulphides
IIB	As+3,Sb+3,Sn+2 Sn+4	-DO-	As Sulphides
III	Fe+3,Cr+3,Al+3	NH4OH in presence of NH4Cl	Metal Hydroxides
IV	Co+2,Ni+2,Mn+2	H2S in solution made ammonical with NH4OH	Metal Sulphides
V	Ba+2,Sr+2,Ca+2	(NH4)2CO3 in presence of NH4Cl & NH4OH	Metal carbonate
VI	Mg+2,NH4+,Na+	No particular reagent	

Appendix D - Summary of Evaluable Predicates

abolish(F,N)	Abolish the procedure named F arity N.
abort	Abort execution of the current directive.
arg(N,T,A)	The Nth argument of term T is A.
assert(C)	Assert clause C.
assert(C,R)	Assert clause C, ref. R.
asserta(C)	Assert C as first clause.
asserta(C,R)	Assert C as first clause, ref. R.
assertz(C)	Assert C as last clause.
assertz(C,R)	Assert C as last clause, ref. R.
atom(T)	Term T is an atom.
atomic(T)	Term T is an atom or integer.
bagof(X,P,B)	The bag of Xs such that P is provable is B.
break	Break at the next procedure call.
call(P)	Execute the procedure call P.
clause(P,Q)	There is a clause, head P, body Q.
clause(P,Q,R)	There is an clause, head P, body Q, ref R.
close(F)	Close file F.
compare(C,X,Y)	C is the result of comparing terms X and Y.
consult(F)	Extend the program with clauses from file F.
current_atom(A)	One of the currently defined atoms is A.
current_functor(A,T)	A current functor is named A, m.g. term T.
current_predicate(A,P)	A current predicate is named A, m.g. goal P.
db_reference(T)	T is a database reference.
debug	Switch on debugging.
debugging	Output debugging status information.
display(T)	Display term T on the terminal.
erase(R)	Erase the clause or record, ref. R.
erased(R)	The object with ref. R has been erased.
expand_term(T,X)	Term T is a shorthand which expands to term X.
exists(F)	The file F exists.
fail	Backtrack immediately.
fileerrors	Enable reporting of file errors.
functor(T,F,N)	The top functor of term T has name F, arity N.
get(C)	The next non-blank character input is C.
get0(C)	The next character input is C.
halt	Halt Prolog, exit to the monitor.
instance(R,T)	A m.g. instance of the record ref. R is T.
integer(T)	Term T is an integer.
Y is X	Y is the value of arithmetic expression X.
keysort(L,S)	The list L sorted by key yields S.
leash(M)	Set leashing mode to M.
listing	List the current program.
listing(P)	List the procedure(s) P.
name(A,L)	The name of atom or number A is string L.
nl	Output a new line.
nodebug	Switch off debugging.
nofileerrors	Disable reporting of file errors.
nonvar(T)	Term T is a non-variable.
nospy P	Remove spy-points from the procedure(s) P.
number(T)	Term T is a number.
op(P,T,A)	Make atom A an operator of type T precedence P.
primitive(T)	T is a number or a database reference

print(T)	Portray or else write the term T.
prompt(A,B)	Change the prompt from A to B.
put(C)	The next character output is C.
read(T)	Read term T.
reconsult(F)	Update the program with procedures from file F.
recorda(K,T,R)	Make term T the first record under key K, ref. R.
recorded(K,T,R)	Term T is recorded under key K, ref. R.
recordz(K,T,R)	Make term T the last record under key K, ref. R.
rename(F,G)	Rename file F to G.
repeat	Succeed repeatedly.
retract(C)	Erase the first clause of form C.
save(F)	Save the current state of Prolog in file F.
see(F)	Make file F the current input stream.
seeing(F)	The current input stream is named F.
seen	Close the current input stream.
setof(X,P,B)	The set of Xs such that P is provable is B.
sh	Start a recursive shell
skip(C)	Skip input characters until after character C.
sort(L,S)	The list L sorted into order yields S.
spy P	Set spy-points on the procedure(s) P.
system(S)	Execute command S.
tab(N)	Output N spaces.
tell(F)	Make file F the current output stream.
telling(F)	The current output stream is named F.
told	Close the current output stream.
trace	Switch on debugging and start tracing.
true	Succeed.
var(T)	Term T is a variable. var(x), succeeds if X is an uninstantiated variable.
write(T)	Write the term T.
writeq(T)	Write the term T, quoting names if necessary.
'LC'	The following Prolog text uses lower case.
'NOLC'	The following Prolog text uses upper case only.
!	Cut any choices taken in the current procedure.
\+ P	Goal P is not provable.
X<Y	As numbers, X is less than Y.
X=<Y	As numbers, X is less than or equal to Y.
X>Y	As numbers, X is greater than Y.
X>=Y	As numbers, X is greater than or equal to Y.
X=Y	Terms X and Y are equal (i.e. unified).
T=..L	The functor and args. of term T comprise the list L.
X==Y	Terms X and Y are strictly identical.
X\==Y	Terms X and Y are not strictly identical.
X@<Y	Term X precedes term Y.
X@=<Y	Term X precedes or is identical Y.
X@>Y	Term X follows term Y.
X@>=Y	Term X follows or is identical to term Y.
[F|R]	Perform the (re)consult(s) specified by [F|R].

REFERENCES

Stefik 82] Stefik M et al, The organization of Expert Systems Tutorial, AI 18,March 1982.

Davis 81] Davis R, Expert Systems, where are we? And where do we go from here?, 7th ICaI 1981.

Sangal 85] Sangal R, Expert Systems, CSI Communications, December 1985.

Davis 82] Davis R and D Lenat, Knowledge Based Systems in AI,Mc Graw Hill, 1982.

Hayes Roth 83] Hayes-Roth, F D A Waterman, D B Lenat, Building Expert Systems,Addison-wesley,1983.

Clark 82] K L Clark and F G Mc Cabe,PROLOG:a language for implementing expert systems,MI 10.

Negoita 85] Constantin Virgil Negoita,Expert Systems and Fuzzy Systems, The Benjamin Cummings Publishing Co, 1985.

Rich 83] Elaine Rich, Artificial Intelligence, Mc Graw Hill 1983.

Togai 86] Masaka Togai and Miroyuki Togai, A VLSI implementation of a fuzzy inference engine towards an Expert System on a chip, Information Sciences, Vol 38, No2, April 86.

Winston 75] Winston, Artificial Intelligence, Addison-wesley,1975.

Feigenbaum 82] Edward A Feigenbaum, The Handbook of AI, Vol 2,Willman Kaufmann 1981-82.

Clocksin 81] W F Clocksin, C S Mellish, Programming in PROLOG , Springer-verlag, 1981.

Myers 86] W Myers, Introduction to Expert Systems, IEEE Expert,Spring '86.