Abstract

The purpose of this report is to provide a detailed description of a database management system that operates on an IBM PC, and was designed to help modelers choose appropriate smog chamber runs for mechanism testings. In addition, this report desctribes the outlines for a prospective expert system.

This report covers:

- a brief discussion of model developing and testing;
- reasons for choosing Turbo Prolog as the implementing language for the system;
- a detailed description of the database fields and underlying organizing principles;
- an overview of the data set;
- a brief introduction to the structure of Turbo Prolog;
- an overall structure of the database management system;
- an introduction to the hierarchical structure of predicates in the system;
- detailed explanations of selected predicates defined in the source program.
- a discussion of the long-term goal of this project involving the incorporation of the database management system into an expert system capable of:
  a) choosing appropriate runs to test the mechanisms;
  b) carrying out automatic model evaluations by making inferences from the sets of testing results;
  c) explaining the rules it used in the process of model evaluations at different levels of details;
  d) acquiring new knowledge from the domain expert through an intelligent editor.

A user's guide to the database management system, a hierarchy of predicates defined in the system, the source codes, and a sample output will also be provided in the appendices.

Acknowledgements

I wish to acknowledge and thank my advisor, Dr. Harvey Jeffries, for his support and guidance throughout my studies at the UNC. Although the work described in this report is my own, the inspirations which started it all came from Dr. Jeffries.

I also wish to thank Dr. Miller and Dr. Flynn for their help reading the report and their comments and criticisms.

Some of the tables and plots were provided by Dr. Kenneth Sexton. The original experimental conditions database were prepared by Dr. Sexton and Mr. Jeffry Arnold. Thanks to both of them. I would also like to thank Ms. Terry Kale for helps she offered to me.

Finally, a special thank should go to my wife, Lai-Choi, for her encouragement.

Contents

Tables

Figures

1

Introduction

This report provides a detailed description of a database management system called OSCECIS (Outdoor Smog Chamber Experimental Conditions Information System). The system was written in Turbo Prolog, a fifth generation computer language based on a subset of predicate logic known as Horn clauses.[1,2,3] For the time being, OSCECIS stands alone as a helping tool whose main function is to provide easy access to the experimental conditions database for the smog chamber experiments conducted in the Outdoor Smog Chamber of the University of North Carolina at Chapel Hill over the past ten years. In the long run, OSCECIS will be incorporated into an expert system capable of assisting the photochemical kinetics model developers in testing their models, as well as carrying out automatic model evaluations.

Model Development and Testing

One of the objectives of conducting smog chamber experiments is to collect reliable data to test photochemical kinetics models which simulate the photochemical reactions taking place in the lower atmosphere. Photochemical kinetics models are the major component in methods for computing control requirements for organic emissions needed to meet Federal Air Quality Standard for ozone.[4]

Model Development

Although the past fifteen years of model development saw many breakthroughs in photochemistry, the photochemical processes of the lower atmosphere are still not completely known. Today, photochemical reactions taking place in the troposphere fall into three categories according to the degree to which they are understood:

1) well-known accepted reactions;
2) reasonably understood reactions; and

3) partially understood reactions.

The incomplete knowledge of photochemical reactions makes it necessary for modelers to speculate about choices in a range of uncertainty.

What complicates the model development even more is that photochemistry is not the only factor affecting the smog formation. Meteorological conditions also play a crucial role in the process. Because multi-cell simulations of smog formation with reasonable meteorological information still demand too much computer memory and time, most models today are restricted to single-cell simulations with minimum meteorological input.

The complexity of photochemistry and meteorology of the lower atmosphere requires that mathematical models be simpler than the actual atmospheric chemistry. This simplification is achieved by three processes: generalization, deletion, and distortion.

The generalization process involves using a single representation (called "lumping") for similar aspects of the process being modeled to reduce the number of items that must be included in the model. The deletion process involves omission of some aspects of the process judged to be unimportant. The distortion process involves changing from exact model representation in some way to an inexact or incorrect representation, for example, treating two consecutive processes as a single step. Each of these simplification processes may be used intentionally or unintentionally by the modelers.[5]

The three simplification processes involve subjective judgements of individual model development groups as to what reaction is more important than the other, what species need to be combined or deleted, what reaction rates are to be used etc. As such, different modelers may produce different descriptions even though the real world situation they are describing is identical and they are using the same kinetics database.

Model Testing

To test the "appropriateness" of the descriptions, smog chamber experiments which vary in concentrations of nitrogen oxides (NO_x) and hydrocarbons (HCs) as well as other initial conditions were conducted and data were collected. These data provide the ground for comparisons between the experimental outcomes and model predictions as well as for comparisons between different models.

Because the condition of a given chamber experiment represents only a tiny fraction of the countless combinations of conditions in the real world, a good fit between even a large set of experimental data and model predictions does not guarantee a good fit between model predictions and data collected from the real world. On the other hand, it is quite safe to say that a model showing poor fit to chamber data is not likely to be an accurate representation of the real world.

But why don't we use the data collected from the real world to test the model instead? Jeffries[6] argued against this approach by considering five factors:

- The meteorological factors are so influential in atmospheric concentration that it is hard to isolate the chemical factors.
- The atmospheric hydrocarbon mixture is too complex and various components of the chemical mechanism can only be tested in isolation.
- The large number of possible combinations of conditions in the urban-like simulations suggests that extremes in conditions may be the most feasible method of covering the range of conditions needed.
- When mechanisms fail in the comparison, "debugging" experiments are needed to assist in understanding why they failed; these types of experiments remove some of the complexity of the urban-like simulation to make the cause and effect relation clearer.
- Whitten's strategy of "bottom-up" construction (to be discussed very soon) and testing of complex mechanisms requires a series of conditions to avoid the ambiguity of offsetting errors and to increase the degree of understanding of the workings of the complex description being constructed.

Figure 1 shows a schematic diagram illustrating the processes of model development and model testing. The arrows marked by "G,D,D" represent steps in which generalization, deletion, and distortion are likely to take place.

Explanations vs Predictions

Besides giving an explanation of the process of smog formation, a good model should also be able to provide predictions. While a great amount of knowledge has accumulated during the past fifteen years of model development, the knowledge of photochemical reactions in troposphere is still far from complete. Every now and then a new reaction rate would be determined or a new species found to have played a role in a previously obscure reaction. The business of model development thus involves not only the task of striking for a balance between model explanation and model prediction but also the task of incorporating into the model the newly

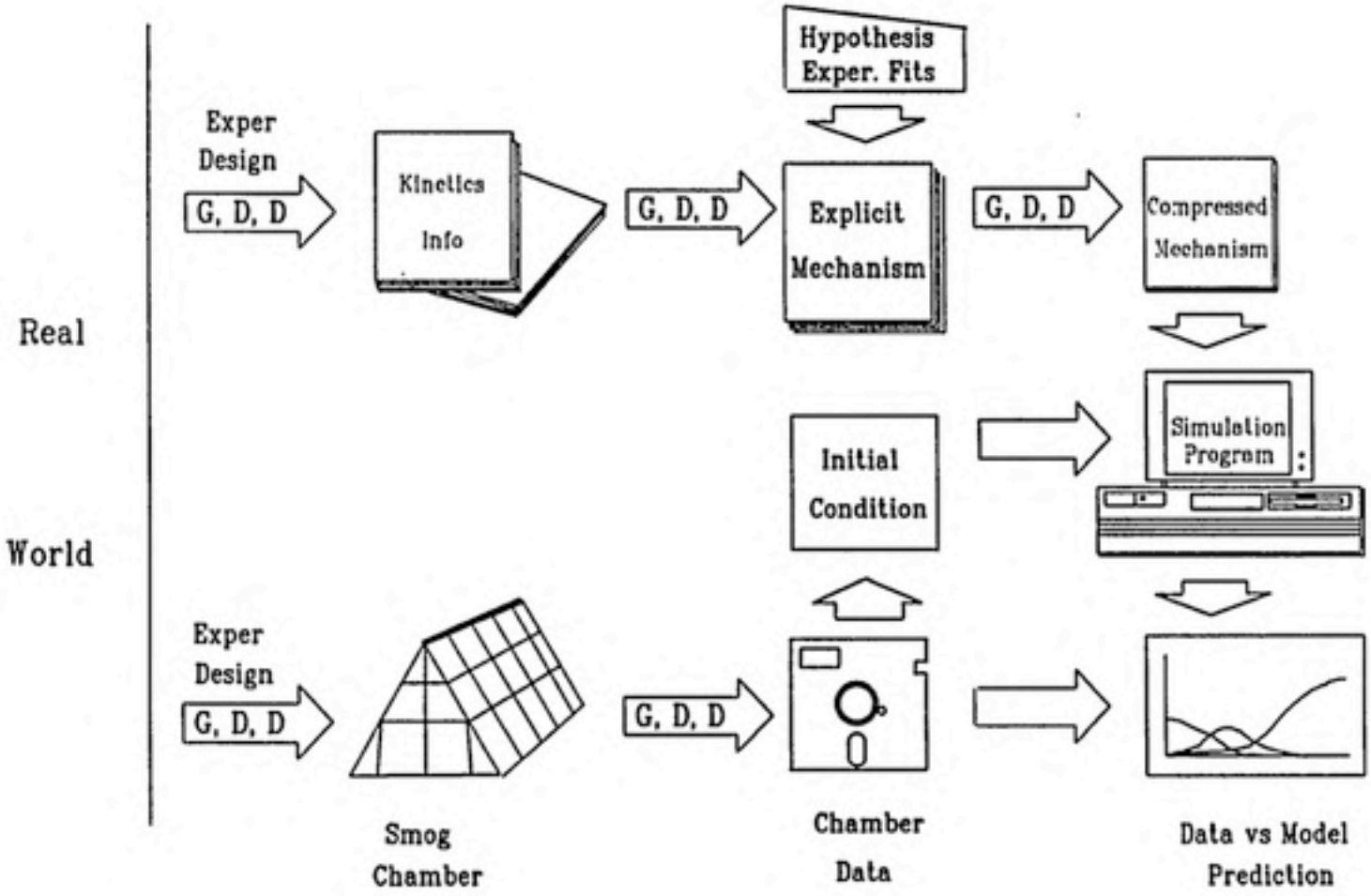

Figure 1. Schematic Diagram Illustrating the Process of Model Development and Testing.

discovered facts which may readily tilt the balance painstakingly maintained in an earlier time. This is one reason why despite fifteen years of continuous efforts, no model developed so far has been found to be totally satisfactory.

A Hierarchical Approach

The formidable task of model development can be made easier if the modelers follow a concept of hierarchy of chemical species to test their models. The concept of hierarchy of chemical species was first described by Whitten.[6] It is based on the number of HC/NO_x system in which the species occur, with the most ubiquitous species occupying the lowest levels. Figure 2 is a modified version of Whitten's schematic diagram by Jeffries.[5] According to this concept, the models should be tested in a "bottom-up"fashion beginning from the bottom box in the diagram.

The hierarchical approach is very important in the sense that it not only helps establish the cause-and-effect in the mechanisms of a model but also suggests that a complex mechanism should not be constructed in one operation but in a stepwise fashion.[7].

The "bottom-up"approach is also applicable to the design of chamber experiments. That is, the experimenter begins with the simplest set of conditions that the mechanism is designed to simulate and increase the complexity of the experimental system one factor at a time. To produce data which simulate the urban environment more closely, Jeffries *et al.* further extend the concept of hierarchy to experimental conditions with different combinations of dilution and injection processes. The hierarchy begins with No Dilution / Initial Injection as the simplest condition and ends in Large Dilution / Continuous Injection as the most complicated condition. Figure 3 is a detailed diagram of this hierarchy produced by Jeffries *et al.* Figure 4 is an illustration of the concept using three chamber experiments.

Need for Experimental Conditions Database

As the knowledge of photochemistry grows, so do the models and the database for testing the models. A direct result of the growth of database is the increase in its physical size and complexity of experimental conditions. The database of the UNC smog chamber experiments has grown to such a point that a separate database containing merely experimental conditions need to be created for easy access to individual experiment or groups of experiment sharing certain combination of experimental conditions. This report deals primarily with this *experimental conditions* database, so called to separate it from the "main"database containing the *experimental outcomes.*

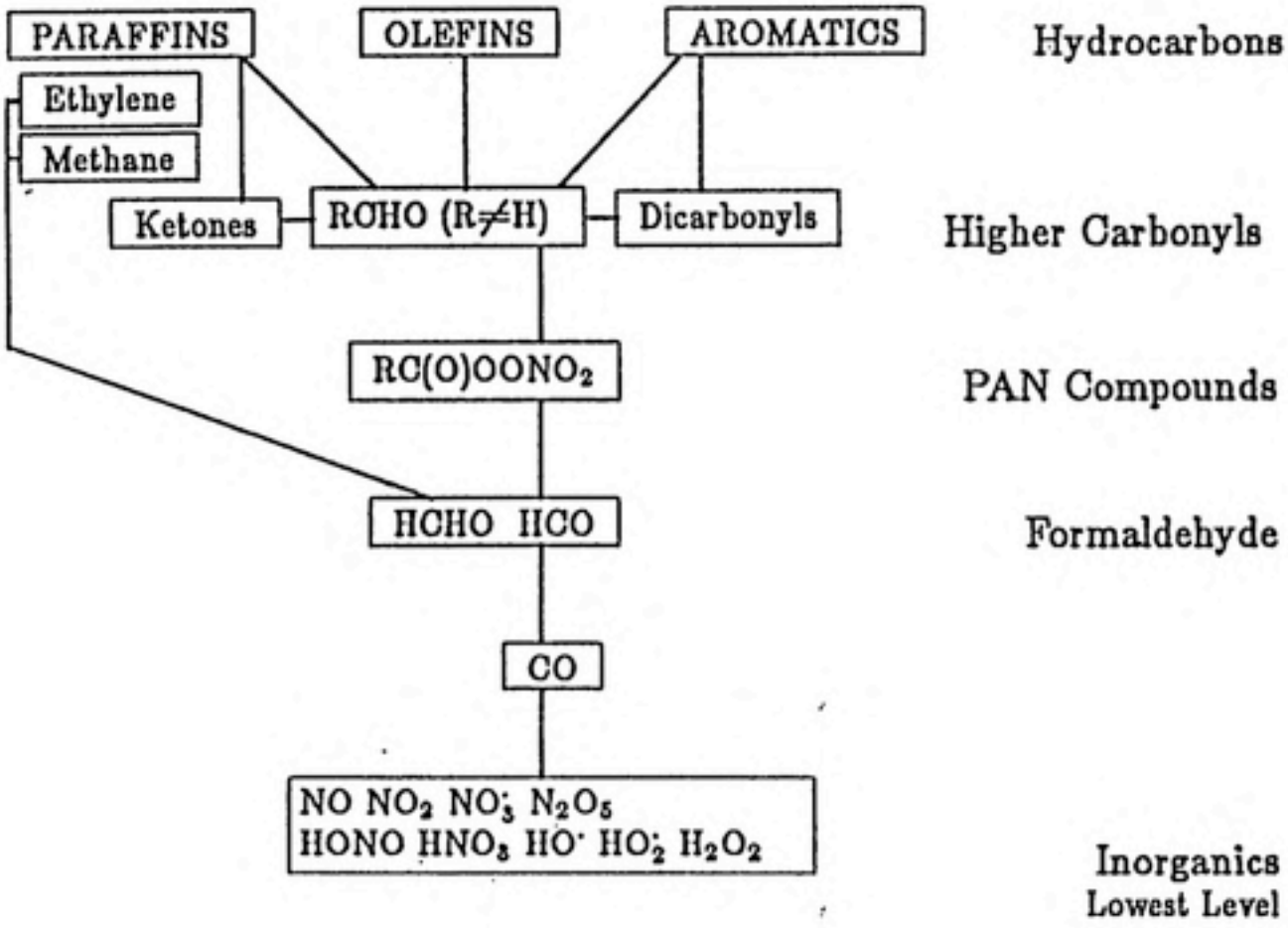

Figure 2. Hierarchy of Species in Photochemical Mechanisms

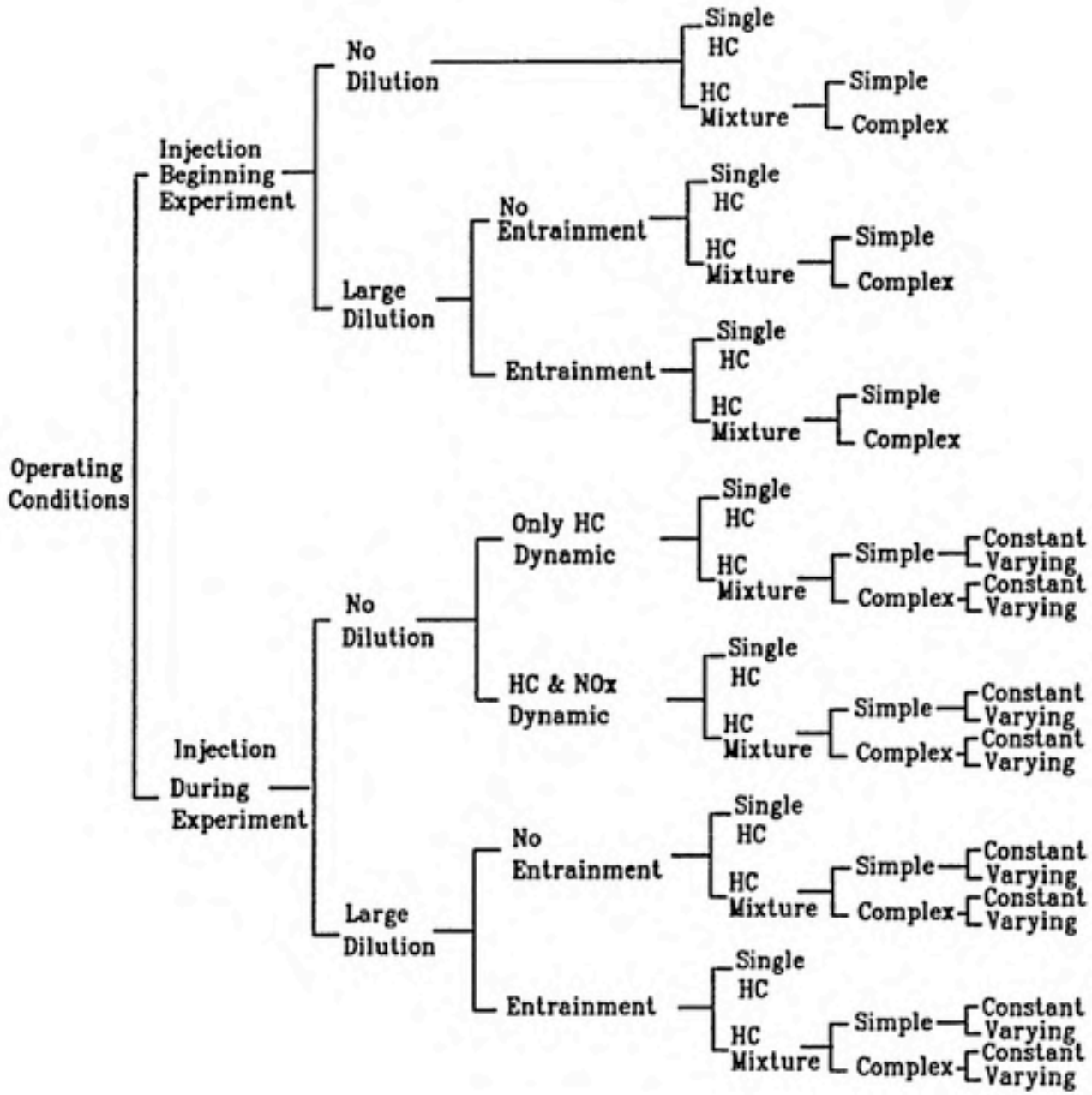

Figure 3. Hierarchy of Experimental Conditions to Simulate the Urban Environment

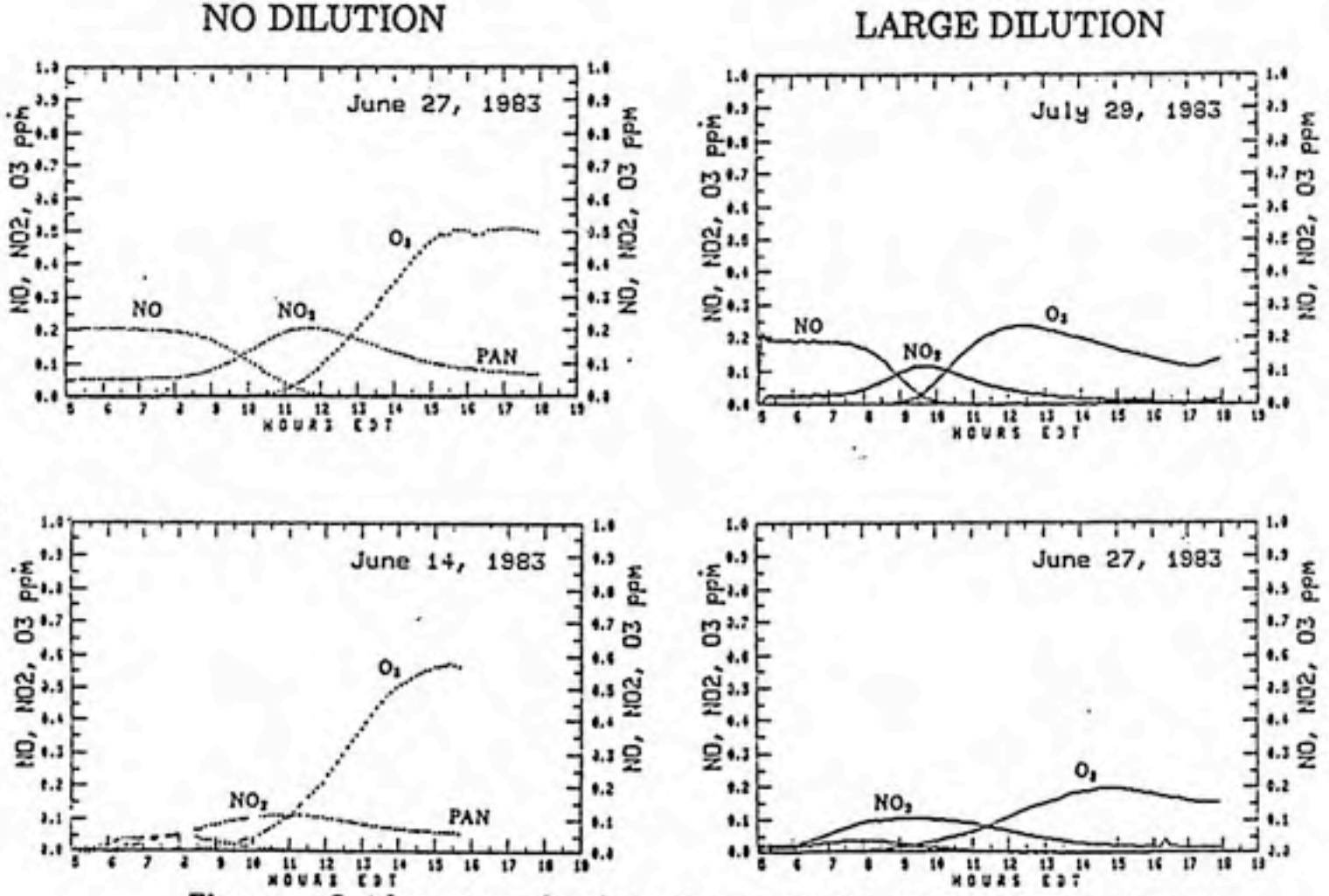

Figure 4. Outdoor smog chamber runs as examples of hierarchial experimental Conditions. NO_x concentration approximately 0.25 ppm, HC concentration approximately 2.6 ppmC of propylene/n-butane/toluene mixture. Large dilution means that 20% of initial mass is left after 10 hours of dilution (equilivalent to mixing height rise from 250-m to 1250-m).

Computer Language: Turbo Prolog

Turbo Prolog, a new version of Prolog programming language, was chosen as the implementing language for OSCECIS after careful considerations of its several distinctive features. The following sections will cover some features of Prolog language in general, followed by specific features of Turbo Prolog.

Areas of Application

Prolog is a relatively new programming language, which has gained worldwide popularity since its first release in 1972 by Alain Colmerauer at the University of Marseilles, France. In 1983, Prolog was chosen as the fundamental system language for the ambitious Japanese national project involving the design and production of fifth generation computers. Today, Prolog is a major contender to the older language Lisp for applications of symbolic computation ranging from relational database, mathematical logic and abstract problem solving to many areas of artificial intelligence such as natural language processing, robotics and expert systems. Although both Lisp and Prolog have efficient way of handling recursive structures, the powerful pattern-recognition facility of Prolog dwarfs the relatively primitive and limited pattern-matching capability of Lisp, not to mention the much cleaner syntax of Prolog. On the other hand, early versions of Prolog tended to make much severe demands on the computer memory than Lisp did, though we will see how Turbo Prolog solves this problem to a great extent in a latter section.

Declarative vs Procedural

One major difference between Prolog and conventional high level languages such as Pascal and PL/1 is that the former allows declarative descriptions of problems whereas the latter are strictly procedural. In a declarative language, the programmer supplies to the computer a set of assertions about entities and their relations without giving execution details. The Prolog system derives procedural meaning automatically from the syntax of the declarative language and carries out required executions. In a procedural language, the programmer supplies stepwise execution details to the computer which, in turn, follows the pre-programmed steps closely to obtain desired answers. In brief, a declarative language tells *what*, whereas a procedural language tells *how*.

By virtue of the declarative nature of the Prolog language, a program for a given application typically requires ten times fewer program lines with Prolog than with Pascal or PL/1. This helps decrease the time and cost of software development tremendously.

Another advantage associated with the declarative nature of Prolog is that it can solve that class of problems having an undeterministic nature; that is, the class of problems which can not be pre-programmed stepwise because the problem space is too large to be handled by a computer or involves combinatorial explosion.

Logical Database

As mentioned at the beginning of this report, Prolog is founded on a subset of predicate logic called Horn clauses. Predicate logic is a formalism that is a natural and powerful representation language marred only by its perceived computational inefficiency.[8] By restricting its foundations to Horn clauses, Prolog loses some degree of expressiveness, but gains a great deal in computational efficiency.[9,10] More details about Horn clauses will be given in Chapter 4.

The structure of Prolog and its strong pattern-recognition facility make it especially suitable for implementing database systems. Database systems implemented in other languages usually require different representations for the database, the query language and the host unit where searching and other data manipulation mechanisms reside. For example, a typical database management system may have the following characteristics: its database may be in the form of a text file, its host unit may be written in a high level language such as Pascal, and its query language may take the form of certain linguistic structures limited only by the imagination of the designer. In Prolog, however, the database, the query language and the host unit all share the same structure. Moreover, the expressiveness inherited from the predicate logic by Prolog provides for free a query language which is natural and very close to human language.

Knowledge Base and Expert System

One of the goals of building a database is to collect facts against which hypotheses, theories or models can be matched. The process of these testing activities involves complex mental exercises which call upon great perseverances and ingenuities on the part of the researchers. While the human intelligence is indispensable in this process, artificial intelligence represented in the computer may be a highly desirable helping tool which can take over some of the chores of reasoning details and leave the more subtle aspects of the problems to the researchers.

The structure of Prolog makes it especially suitable for knowledge representations upon which a knowledge base can be built. The major difference between a database and a knowledge base is that the objects of the former are data, whereas the objects of the latter are entities, their relations, and rules describing these relations. The knowledge base, together with a knowledge acquisition facility, an

inference mechanism, and an explanatory interface, constitutes an expert system which, if successfully implemented, can perform the tasks of a given domain at an expert's level.

The Prolog language provides a bridge between a database and a knowledge base by virtue of its unique language structure. The knowledge base can be built in an incremental fashion without changing the structure of the original database system (there is only one language structure anyway). This will result in a great saving of efforts and time. Unless the database designer is very sure that his or her database is not going to involve in any form of complex reasoning or that there is never a need to develop the database into a knowledge base, Prolog should be seriously considered as a candidate for the implementing language.

Why Turbo Prolog?

As a latest implementation of Prolog language, Turbo Prolog retains all the major advantages of Prolog language discussed in previous sections while at the same time adding to it many desirable new features. Some of them are summarized below.

1. Turbo Prolog is a full-fledged compiler which can produce stand-alone programs for the IBM PC and compatibles. Older versions of Prolog are mostly interpreters. The differences between an interpreter and a compiler are: Interpreter accepts a source program, translates it into some intermediate data structure, and then executes the algorithm by carrying out each operation given in the intermediate structure. An interpreter is considerably less efficient than a compiler because it has to carry out the translation "ritual" every time it executes the program. A compiler performs translation step once and the output (an execution file) can be executed many times without overhead. It demands fairly heavy computer resources while compiling, but when executing, only those resources needed by the executing program are required.[11] A compiled Turbo Prolog program is more conservative in its memory requirements than an equivalent interpreted program of any earlier version of Prolog while, at the same time, it executes at a much faster speed.
2. Turbo Prolog has a typed structure in much the same way that Pascal does. The typed structure not only makes debugging a much easier job but also reduces the space requirements of the Prolog language.
3. Unlike earlier implementations of Prolog which allow only integer in mathematical operations, Turbo Prolog allows both integer and real operations and functions as well as bit-wise operations for control and robotics applications.
4. Turbo Prolog provides a comprehensive, fully modular program development environment. A program can be broken up and compiled in separate modules

which are linked together in a later stage. This feature is highly desirable for developing huge programs. Moreover, the Turbo Prolog development group is currently putting their efforts in developing an interface between Turbo Prolog and Turbo Pascal, an exciting feature which will enhance the powers of the two already powerful languages even more.

5. Turbo Prolog allows a full control of window facility which contain mixed text and graphics. It also allows an easy access to the memory and I/O ports. These tools will be especially useful for creating user-friendly programs.

The features discussed in this and earlier sections suit our purposes of developing the OSCECIS well. As mentioned earlier, OSCECIS is a first step towards an ultimate goal of developing an expert system capable of assisting the modelers in testing their models as well as carrying out automatic model evaluations based on the given sets of testing results. The suitability of Prolog as an expert system developing language has been proven by many successful applications.[12] The additional features of Turbo Prolog can only enhance this suitability.

Previous Work

The experimental conditions database is extracted from the main database containing the experimental outcomes and complete documentation of the UNC smog chamber experiments. Table 1 shows an example Segmented Data File (SegFile) for the run conducted in October 4, 1983. A SegFile contains complete information of a fully processed run in the main database. The data in a given SegFile can be used to generate plots such as those shown in Figure 5.

The organizing principles of the experimental conditions database has been discussed in the 1985 UNC report entitled "Outdoor Smog Chamber Experiments to Test Photochemical Models: Phase II"[5] which also contains a complete index to all pre-1984 chamber experiments and suggestions on run selection for model testing. The OSCECIS follows these organizing principles closely with only a slight change in run series and rankings and an addition of a new field containing information of percentage of sun light.

Structure of This Report

In Chapter 2, we will outline the organizing principles of the experimental condition database while describing each of its fields. An overview of the data set will be given in Chapter 3. In Chapter 4, a brief introduction of the structure of Turbo Prolog will be given, followed by the discussion of the overall structure of OSCECIS. Chapter 5

Table 1. Example Segmented Data File

```
                    GENERAL DOCUMENTATION

RUNDATE: OCTOBER 04, 1983

RUNTYPE: AUTO

RUN DESCRIPTION: COMPARISON OF REACTIVITY OF EXHAUST
FROM DIRECT INJECTION FROM DODGE CHARGER IN HIGH IDLE
WITH SYNTHETIC AUTOEXHAUST.

RESULTS: TWO SYSTEMS RESULTED IN SIMILAR REACTIVITY

INITIAL CONDITIONS:          BLUE          RED
DODGE CHARGER                0.0           2.587
SYNTHETIC EXHAUST NMHC       2.190         0.0
NO                           0.214         0.215
NO2                          0.037         0.039
.
.
.
88888
04-OCT-83                                  GENERATED ON 19-MAR-84
SY:0C043K.C1G        .C1G
PICKED DATA ENTERED BY JEFFREY HOFFNER
CALIBRATION FACTORS APPLIED BY CHARLES
TOLUENE                          2.912000E-01    PPMC/IN      2     KGS
ETHYLENE                         9.020000E-02    PPMC.IN      2     KGS
.
.
NAME ABBREVIATIONS - SAME ORDER AS IN DATA
TOLUENE   IS TOLUENE                          MAX A:   1.5610 MAX CON:   0.4546
ETHYLENE  IS ETHYLENE                         MAX A:   4.9164 MAX CON:   0.4435
.
```

Table 1. cont. Example Segmented Data File

```
88888
04-OCT-83                                   GENERATED ON 13-MAR-84
SY:OC043K.C2G        .C2G
PICKED DATA ENTERED BY JEFFREY HOFFNER
CALIBRATION FACTORS APPLIED BY JEFFREY HOFFNER
ETHANE                              1.498000E-01    PPMC/IN       2     KGS
PROPYLENE                           7.307000E-01    PPMC/IN       2     KGS
.
.
NAME ABBREVIATIONS - SAME ORDER AS IN DATA
ETHANE    IS ETHANE                                 MAX A:   0.2014 MAX CON:   0.0302
PROPYLENE IS PROPYLENE                              MAX A:   0.1311 MAX CON:   0.0958
.
.
88888
.
.
.
88888
USER DOCUMENTATION FOR RUN 831004
.
.
CALIBRATION FACTORS USED:
SPECIE
 TIME      INTERVAL  BEGINNING     GAIN      ENDING  BEGINNING     OFFS      ENDING
                       GAIN        SLOPE      GAIN     OFFSET      SLOPE     OFFSET
  HR          HR                   HR-1                            HR-1
O3G
 2.500     15.500       1.01380   0.00000   1.01380   0.00000   0.00000   0.00000
18.000      6.000       0.00000   0.00000   0.00000  -9.99999   0.00000  -9.99999
.
.
END OF PROGRAM DOCUMENTATION
99999
YYMMDDHHMM    U    S     O3G       NOG       NOXG      NO2G      DPG       CTMP
    LTMP      TSR         UV
8310040500    1   7R     0.0008    0.0864    0.1138    0.0237   50.0450   56.8300
   58.6810   -0.0046    -0.2439
8310040504    1   3B     0.0196   -0.0022   -0.0007    0.0001   56.6496  -60.5560
   21.6880   -0.0062    -0.2439
```

Table 1. cont. Example Segmented Data File

```
8310041704    1   3B    0.5787   -0.0012    0.0513    0.0507   72.7266  -60.5120
   20.5120    0.1592   11.7073
8310041708    1   7R    0.5508   -0.0013    0.0578    0.0573   64.7658   83.3620
   81.3040    0.1376   10.2439
99999
YYMMDDHHMM    USER SIDE      GENERATED ON 19-MAR-84
2,4,4 TRI TOLUENE   ETHYLENE  N-BUTANE  TRANS-2-B ISOPENTAN N-PENTANE ACETYLENE
8310040625    1    B
   0.2266    0.4546    0.4435    0.0877    0.0366    0.1339    0.0000    0.0905
8310040655    1    R
   0.1868    0.2580    0.2248 -999.0000    0.0459    0.1195    0.0634    0.2332
.
.
8310041455    1    R
   0.1439    0.1573    0.0890 -999.0000    0.0000    0.0828    0.0340    0.167
8310041525    1    B
   0.1821    0.2514    0.1300    0.0688    0.0000    0.0921    0.0000    0.000
99999
YYMMDDHHMM    USER SIDE      GENERATED ON 13-MAR-84
ETHYLENE  ETHANE    PROPYLENE
8310040625    1    B
   0.4522    0.0093    0.0952
8310040655    1    R
   0.4097    0.0238    0.0958
.
.
8310041525    1    B
   0.1603    0.0086    0.0000
8310041555    1    R
   0.2675    0.0260    0.0000
99999
```

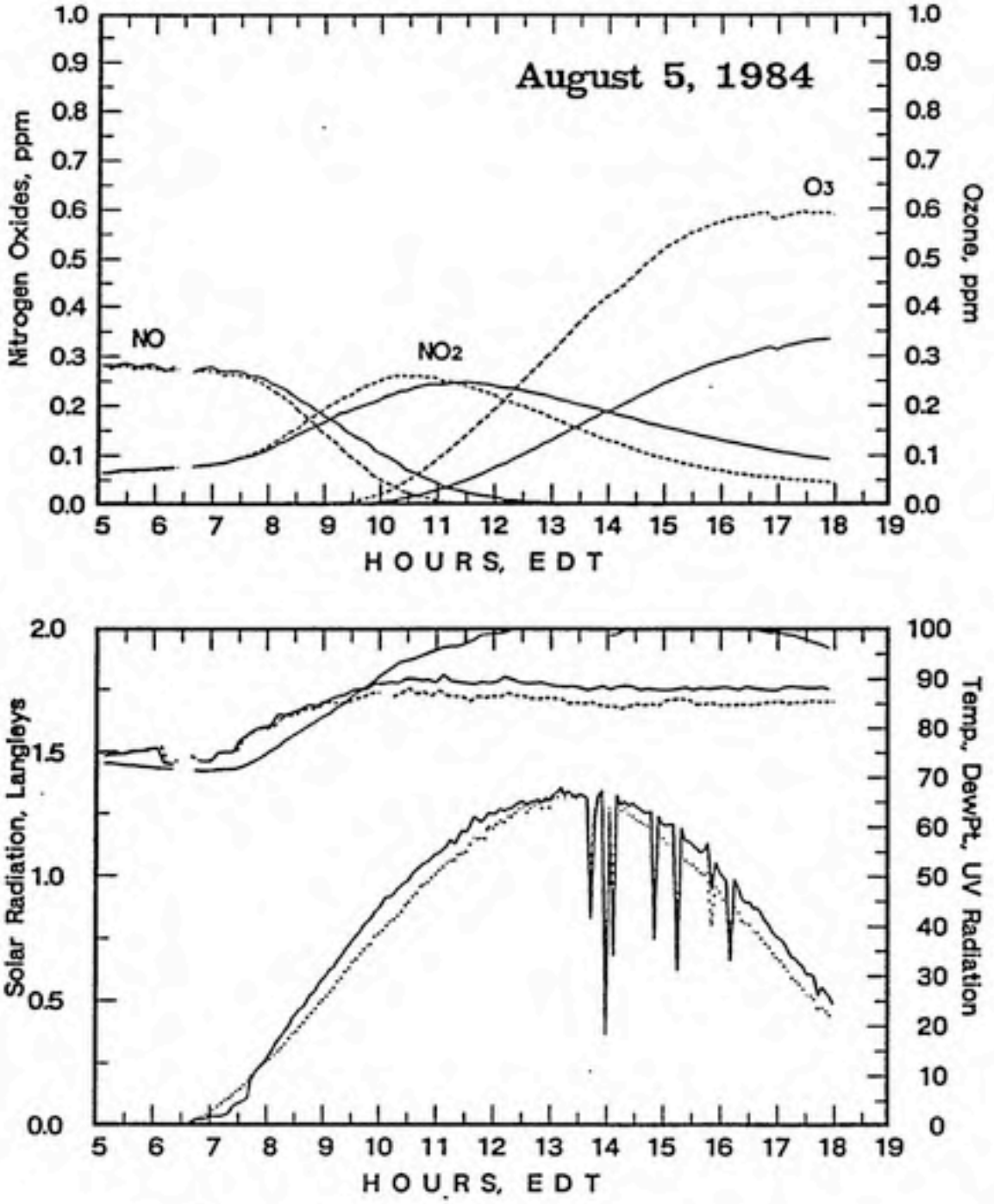

Figure 5. An Example Plot

Top: (Solid) 0.91 ppmC SYNAUTO, no MeOH, 0.01 ppm HCHO;
(Dashed) 1.32 ppmC SYNAUTO, no MeOH, 0.02 ppm HCHO;
Bottom: RED chamber air temperature (top solid line, °F);
RED (solid line) and BLUE (dashed line) chamber dewpoint (°F);
ambient total solar radaiation (solid line, cal-cm^{-2}-sec^{-1});
ambient ultraviolet radiation (dashed line,mcal-cm^{-2}-sec^{-1}).

gives detailed explanations of selected predicates defined in OSCECIS. Chapter 6 discusses the long-term goal of this project: the development of an expert system into which the OSCECIS will ultimately be incorporated. Appendix A contains the user's guide to the system. Appendix B contains the hierarchical structure of predicates defined in the system. Appendix C lists the source program. Appendix D shows a sample output of the system.

2

Database Records

This chapter covers a detailed description of the fields of the database records. The organizing principles of the database have been described in the 1985 UNC report "Outdoor Smog Experiments to Test Photochemical Models: Phase II".[5] In the process of explaining the organizations of the database fields, some organizing principles will be reiterated but not covered in full details.

Each record in the database is composed of 27 fields. For reasons which will be made clear in chapter 4, each record is physically broken up into two portions. The first portion contains mainly codes of experimental conditions, whereas the second portion contains mainly concentrations and identifications of nitrogen oxides (NO_x) and hydrocarbons (HCs) for a given experiment in the two sides of the smog chamber. Figure 6 is a schematic diagram of the structure of a single record in the database. Each box in the record encloses the name of the corresponding field. Detailed description of each field follows.

Date of Run

This is the principal key for each record. It consists of a character string of length six in the form "yymmdd", where y's are the last two digits of the year in which the run was conducted, mm is the month and dd is the day. For example, the run date "770518" uniquely identifies the run conducted in May 18, 1977.

Class of Run

This field classifies the experiments conducted in the chambers into two basic types: Characterization runs and Organic/NO_xruns. Physically, this field is composed of a character string of length one.

First Portion: Experimental Conditions

Run Date	Class of Run	Dilu-tion	Injec-tion	H C type	Proj type	P c status	Series	Rank

Second Portion: Concentrations and Identifications of HCs

	Red Side Info							Blue Side Info									
Run Date	Red NOx Con	1 st H C Con	1 st H C I D	2nd H C Con	2nd H C I D	3rd H C Con	3rd H C I D	Blue NOx Con	1 st H C Con	1 st H C I D	2nd H C Con	2nd H C I D	3rd H C Con	3rd H C I D	H C Instru-ment	Quality of Run	Sun light

Figure 6. Schematic Diagram of the Structure of a Single Record

Characterization runs address unique aspects of the UNC chamber performance. They are designed to provide chamber-specific data such as the magnitude of O3 produced by rural background air, magnitude of NOx emissions from the chamber walls, photolysis rate of aldehydes etc. The following codes designate the Characterization runs:

- C – Characterization experiment.
- D – Characterization experiment, second day.

Organic / NOx runs are designed to test the chemical aspect of a mechanism. For instance, a run of this type may contain propylene in one side of the chamber and ethylene / acetaldehyde in the other side of the chamber in order to test whether a mechanism can adequately describe the chemical transformations of both sides. The Organic / NOx runs and their corresponding codes are shown below:

- O – One day organic NOx experiment.
- T – Two day organic NOx experiment, first day.
- S – Two day organic NOx experiment, second day.

Dilution

This field identifies two types of chamber experiments: runs with dilution rates controlled at about 1 dilution rates which are used to approximate various patterns of mixing height rises. The following codes designate the two types of dilution:

- N – Normal dilution (about 1 % / hour).
- L – Large dilution (about 5-fold dilution in 10 hours).

Injection

This field is similar to the previous field. It classifies runs into various injection levels of reactants. The following codes designate the five injection levels:

- I – All reactants injected before sunrise.
- H – HC injected during experiment, NOx injected before sunrise.
- N – NOx injected during experiment, HC injected before sunrise.
- B – Both HC and NOx injected during experiment.
- C – Combination of injection different from above.

By varying the two attributes of Dilution and Injection, a hierarchy of experimental conditions can be developed to simulate the urban environment in a progressive manner. The hierarchy begins with No Dilution / Initial Injection combination at the lowest level and ends with Large Dilution / Continuous Injection combination at the highest level. Table 2 shows the four basic combinations of Dilution and Injection.

Hydrocarbon Type

The field is designed to give a summary of the relationship of the hydrocarbon species or mixtures in both sides of a given side-by-side chamber experiment. It consists of a character string of length two. There are two basic types associated with this attribute of the database. The first type consists of runs with a single species on each side of the chamber. The following codes designate HC class of species of this type:

- 1A – aldehyde on each side;
- 1O – olefin on each side;
- 1P – paraffin on each side;
- 1R – aromatic on each side;
- 1U – olefin one side, aldehyde other side;
- 1V – olefin one side, paraffin other side;
- 1W – olefin one side, aromatic other side;
- 1X – some other combination.

The second type consists of runs with mixture on at least one side of the chamber. The following codes designate HC class of species of this type:

- MB – simple mixture of two or three species;
- MC – named mixture or mixture of named mixtures;
- MV – mixture with time varying composition.

A simple mixture has two or three species. A named mixture or mixture of named mixtures has at least four species and can have several hundred (*e.g.* auto exhaust). Table 2 and Table 3 give the named mixtures used in the UNC experiments.

Table 2. Composition of Hydrocarbon Mixtures used in UNC Smog Chamber Experiments.

Compound	SIMMIX1	SIMMIX2	UNCMIX	SIMARO	COMARO	AUTO-0	AUTO-1A	AUTO-1B	P/B	P/B/T	BASMIX
butane	0.4002	0.4539					0.0391	0.0391	0.7663	0.5352	0.3160
pentane	0.3183	0.3610	0.3531							0.1268	
isopentane			0.1484			0.0550	0.0519	0.0519			
2-methylpentane			0.0996								
2,4-dimethylpentane			0.0864								
2,2,4-trimethylpentane			0.1202				0.1121	0.1121			
ethylene	0.1631	0.1850	0.1167			0.4000	0.2391	0.2391			0.0650
propylene	0.1184		0.0524			0.2200	0.0416	0.0416	0.2337	0.1648	0.0943
1-butene			0.0254				0.0196	0.0196			
trans-2-butene							0.0196	0.0196			
cis-2-butene			0.0313								
2-methyl-1-butene			0.0347								
2-methyl-2-butene			0.0317								
benzene							0.0538	0.0538			
toluene				0.4846	0.2482	0.3000	0.2116	0.2116		0.3000	0.2000
m-xylene				0.3850	0.1882					0.2000	
o-xylene							0.0481	0.0481			
1,2,4-trimethylbenzene				0.1334	0.2803	0.1250	0.0564	0.0564			
n-propylbenzene					0.1371						
sec-butylbenzene					0.1522						
formaldehyde							0.0200	0.0200			
acetaldehyde											
CO								AUTO[1]			
total paraffin	0.7185	0.8150	0.7077	0.0000	0.0000	0.0550	0.2031	0.2031	0.7663	0.5352	0.4608
total olefin	0.2815	0.1850	0.2923	0.0000	0.0000	0.6200	0.3199	0.3199	0.2337	0.1648	0.1393
total aromatic	0.0000	0.0000	0.0000	1.0000	1.0000	0.3250	0.4724	0.4724	0.0000	0.3000	0.4000

[1] $[CO]_{propene}$ matched to $[CO]_{ethene}$

Table 3. Composition of Hydrocarbon Mixtures used in UNC Smog Chamber Experiments.

Compound	TOL-XYL-MIX1	TOL-XYL-MIX2	TOL-XYL-MIX3	TOL-TMB-XYL3	ISO-/2,2,4-TM-PENTANES	DODGEMIX II
butane						0.4449
pentane						
isopentane					0.5000	
2-methylpentane						
2,4-dimethylpentane						
2,2,4-trimethylpentane					0.5000	
ethylene						0.0615
propylene						0.0810
1-butene						
trans-2-butene						0.0498
cis-2-butene						
2-methyl-1-butene						
2-methyl-2-butene						
toluene	0.5000	0.6666	0.8666	0.4213		
m-xylene	0.5000	0.3334	0.1667	0.3572		
o-xylene			0.1667	0.1984		
1,2,4-trimethylbenzene				0.1231		
n-propylbenzene						
sec-butylbenzene						
formaldehyde						0.0148
acetaldehyde						0.0098
CO						
total paraffin	0.0000	0.0000	0.0000	0.0000	1.0000	0.4449
total olefin	0.0000	0.0000	0.0000	0.0000	0.0000	0.1953
total aromatic	1.0000	1.0000	1.0000	1.0000	0.0000	0.1386

Processing Status

This field is composed of a character string of length one representing the processing status of a run. The suitability of a run for mechanism testing depends on factors such as sunlight or whether there are instrument problems associated with the particular run. As such, not all runs are worth processing. In addition, runs are assigned a priority for processing based on factors such as the experimental conditions or the extent to which the run conditions provided unique information. Five types of existing processing status and their corresponding codes follow:

- S – A fully processed and documented run, available for distribution on magnetic media.
- P – A partially processed run, currently being processed, available in graphical form with estimated initial conditions of ca. ±20 %.
- L – A partially processed run of low priority, available in graphical form for raw data with target initial conditions of ca. ±20 % for NOx and HCs. This type of run may never be fully processed.
- N – Not to be processed, not available.
- U – Unprocessed run; a processible run but no processing has occurred.

Project

Four projects using the UNC Outdoor Smog Chamber have contributed data for model testing as well as for the experimental conditions database. The codes identifying the project types were declared as character strings of length two. They are:

- ET – Experiments to test explicit mechanisms for aldehydes, olefins, paraffins, and simple two-component of mixtures.
- RE – The reactivity grant project, primarily dealing with mixtures and changes in mixture composition.
- AU – The automobile exhaust smog chamber project, dealing with the relative reactivity of exhaust from two vehicle, with two gasolines.
- ME – Methanol exhaust project, dealing with the relative reactivity of exhaust from vehicle with methanol additive fuel.

Series

Series type was designed to put individual runs into meaningful groups or sequences

so as to aid modelers in choosing appropriate runs for model testings. Table 4 contains the forty series types set up for this purpose. Note that the leading zeros for those series codes smaller than 10 can not be omitted since the database field containing the series type is a character string of length two instead of an integer.

Table 4. Series Codes Used in the Experimental Conditions Database

Code	Series Name
01	Characterization; Background
02	Characterization; NOx oxid. NO only
03	Characterization; NOx oxid. CO added
04	Characterization; O3 decay
05	Characterization; NO2/O3,dark
06	Characterization; Ald. added
07	Matched propylene
08	Matched mixture
09	Matched other
10	Replicate propylene
11	Replicate other
12	Aldehyde development
13	Olefin development
14	Paraffin development
15	Aromatic development
16	HC type transition
17	Terpene development
18	Reactivity comparison
19	Single to Simple
20	Simple Comp. change
21	Complex Comp. change
22	Simple to Complex
23	Substitution, UNCMIX
24	Substitution, P/B
25	Substitution, SIMMIX
26	Substitution, Other
27	Addition, one species
28	Addition, CO
29	Addition, Aldehyde
30	Addition, Other

31	AutoExhaust
32	EKMA test
33	Delta HC
34	Delta NOx
35	Static to dynamic
36	Temperature effects
37	Water effects
38	Solar effects
39	Ketone development
40	All NO2

Rank

Unlike all the fields mentioned earlier, this field is composed of an integer instead of a character string. The purpose of ranking runs is to aid in selecting runs from the database for model testing. Runs are ranked into various categories as shown in Table 5.

Table 5. General Run Rankings

Rank Range	Category
1	Best in HC type
2-10	Best of kind
11-50	Satisfactory
51-100	Supporting
101-500	Some problems
501-	Not for modeling

The assignment of the rank number to a run is a result of a subjective judgement based on overall quality of the run, weather conditions, and HC instruments used.

Concentrations and HC Species/Mixture Identifications

This section discusses a collection of fields which contain information on NOx and HC concentrations, and HC species/mixture identifications. The fields are marked by "Red Side Info" and "Blue Side Info" in Figure 6. A quick examination of Figure

6 reveals that the "Red Side Info" and "Blue Side Info" share a common structure. For simplicity, the structure of the "Red Side Info" will only be discussed since the discussion applies also to the "Blue Side Info".

The first field of the "Red Side Info" contains the NOx concentration in ppm. The second, fourth and sixth fields of the "Red Side Info" contain concentrations, in ppmC, of the first, second and third HC species/mixture respectively. The third, fifth and seventh fields of the "Red Side Info" contain identifications of the first, second and third HC species/mixture respectively. When there is less than three HC species/mixtures , the field corresponding to the empty HC species/mixture will be given a value zero. As a rule, these fields are filled up in left to right order.

All fields containing concentrations are of real type whereas all fields containing HC identifications are of integer type.

HC Instrument

This field contains the number of HC instrument used in a particular experiment. The information in this field will be used to judge the quality of the run and to assign a rank to the particular run. This field is of integer type.

Quality of Run

The major determining factors in the quality of a run are weather conditions such as sunlight, cloudiness and haze condition as well as the number and performance of the analytical instruments. Details of grading techniques has been covered in the 1982 final report "Outdoor Smog Chamber Experiments to Test Photochemical Models".[13] Experiments are rated on a scale of 0 to 9 (highest). This field is of integer type.

Sunlight

This field contains information on the percent of possible solar radiation received for the day. The percent solar radiation is a function of cloud cover of the day. This field is of integer type.

3

Overview of Data Set

This chapter provides a general view of the data set by presenting the distributions of chamber runs according to different database attributes discussed in the previous chapter. A chapter with similar goal has been given in the 1985 UNC report.[5] Since the publication of the report, however, eighty seven new runs have been added to the database. Not only these new runs brought the total number of runs in the database from 345 to 432, but they also made the information contained in the old report obsolete. This chapter updates the distributions information of the database and, at the same time, serves as a supplement to the discussions in the previous chapter.

Period Covered, Distribution

Table 6 shows the distribution of experiments by year. Runs conducted in 1977-80 period were discussed in an earlier report.[14] The number of runs listed for these years only included those experiments that have been fully processed. Runs conducted in 1984-86 period have not been covered in any earlier report.

Table 6. Distribution of Experiments by Year

Year	Runs
'77	18
'78	45
'79	28
'80	23
'81	67

'82	90
'83	74
'84	52
'85	19
'86	16

Table 7 shows the distribution of experiments by month. It can be seen that experiments were performed mainly in the summer months when sunlight is abundant. Experiments conducted in non-summer months were primarily designed to provide data on temperature and solar effects.

Table 7. Distribution of Experiments by Month

Month	Runs
Jan	3
Feb	1
Mar	2
Apr	1
May	4
Jun	65
Jul	91
Aug	97
Sep	73
Oct	69
Nov	23
Dec	3

Table 8 gives the distribution of runs by project. Experiments conducted in 1984-86 period contributed 40 runs to the "ET" type project and 47 runs to the Methanol Exhaust project.

Table 8. Distribution of Experiments by Project

Code	Project	Runs
ET	Exp. Test Photo. Mech. 1978-80	114
ET	Exp. Test Photo. Mech. 1981-83	128
ET	Exp. Test Photo. Mech. 1984-86	40
RE	HC Reactivity	69
AU	Automobile Exhaust	34
ME	Methanol Exhaust	47

The distribution of experiments by class is shown in Table 9. As can be seen, the majority of the runs are of "One day, regular" type.

Table 9. Distribution of Experiments by Class

Code	Class	Runs
C	Characterization	48
D	Second day, char	12
O	One day, regular	364
T	Two day, regular	5
S	Second day, reg.	3

Table 10 presents the breakdown of the runs by dilution and injection conditions. The characters enclosed in parenthesis are codes for the corresponding types.

Table 10. Distribution of Experiments by Dilution and Injection Conditions

Dilution	Injection	Runs
Normal (N)	Initial Only (I)	414
	HC Continuous (H)	5
	NOx Continuous (N)	0
	HC & NOx Continuous (B)	7
	Other Combination (C)	1
Large (D)	Initial Only (I)	2
	HC Continuous (H)	0
	NOx Continuous (N)	0
	HC & NOx Continuous (B)	0
	Other Combination (C)	3

Compounds and Mixtures Used

A summary of experiments by the HC type is shown in Table 11 . "Single, olefins" and "Mixture, complex" are by far the two predominant types of run.

Table 11. Distribution of Experiments by HC Type

Code	HC Type	Runs
1A	Single, aldehyde	49
1O	Single, olefins	103
1P	Single, paraffins	13
1R	Single, aromatics	11
1U	Single, ole vs ald	16
1V	Single, ole vs par	2
1W	Single, ole vs aro	3
1X	Single, other	14
MB	Mixture, simple	37
MC	Mixture, complex	131
MV	Mixture, varying	3
B	Background air	41
S	One side	4

Table 12 Number of Chamber Sides Containing Species lists all the species and the mixtures used in the experiments. The number given in the table is the number of chamber sides in which the species or mixture appreared.

Table 12. Number of Chamber Sides Containing Species

Sides	Runs
54	ETHYLENE
204	PROPYLENE
2	PROPANE
9	1-BUTENE

40	N-BUTANE
7	TRANS-2-BUTENE
1	ISOPENTANE
1	N-PENTANE
3	2,3-DIMETHYLBUTANE
1	BENZENE
50	TOLUENE
2	N-OCTANE
5	ETHYLBENZENE
28	M-XYLENE
14	O-XYLENE
5	N-PROPYLBENZENE
6	1,2,4-TRIMETHYLBENZENE
1	2,2,4-TRIMETHYLPENTANE
1	M-ETHYLTOLUENE
22	ISOPRENE
9	A-PINENE
78	FORMALDEHYDE
4	METHACROLEIN
4	METHYLVINYLKETONE
62	ACETALDEHYDE
3	ACETONE
3	BENZALDEHYDE
5	METHYL ETHYL KETONE
2	PROPIONALDEHYDE
9	BIACETYL
8	METHYLGLYOXAL
1	METHYLBENZYLQUINONE
1	N-PROPYLNITRATE
4	N-BUTYLNITRATE
4	3-PENTANONE
1	GYLYCOALDEHYDE
1	ACETONITRILE
12	METHANOL
7	O3
41	CO
8	H2O2
5	N2O
24	SYNTHETIC AUTO EXHAUST
31	'72 DODGE AUTO EXHAUST DIRECT

7	'72 DODGE AUTO EXHAUST CRYO
18	'79 VOLARE AUTO EXHAUST DIRECT
2	SYNTHETIC AUTO EXHAUST 0
3	SYNTHETIC AUTO EXHAUST 1A
2	SYNTHETIC AUTO EXHAUST 1B
18	SYNURBAN
4	SYNAUTO84
2	HIMWAUTO
6	TOL/XYL/MIX1
2	TOL/XYL/MIX2
2	TOL/XYL/MIX3
2	TOL/TMB/XYLS
4	ISO-/2,2,4-TM-PENTANES
111	UNCMIX
9	SIMMIX1
11	COMARO
24	SIMARO
15	BUTANE/PROPYLENE
20	SIMMIX2
13	AROMATIC MIX
4	DODGEMIX II
28	TWO DAY CON'T
56	BACKGROUND

Experimental Series

The distribution of runs by series is given in Table 13 . The series are grouped according to properties designated by the first word of the name of each series.

Table 13. Distribution of Experiments by Series

Code	Name	Runs
01	Char; Background	12
02	Char; NO oxid. NO only	7
03	Char; NO oxid. CO added	11
04	Char; O3 decay	9
05	Char; NO2/O3. dark	4
06	Char; Ald. added	8

07	Matched propylene	39
08	Matched mixture	13
09	Matched other	25
18	Reactivity Comparison	149
23	Substitution, UNCMIX	12
24	Substitution, P/B	5
25	Substitution, SIMMIX	2
26	Substitution, Other	17
27	Addition, one species	3
28	Addition, CO	14
29	Addition, Ald	5
30	Addition, Other	6
33	Delta HC	60
34	Delta NOx	8
35	Static to dynamic	18

Quality, Rank, Processing Status, Sunlight

The distribution of runs by quality is shown in Table 14. It can be seen that the majority of the runs are of resonably good qualities.

Table 14. Distribution of Experiments by Quality

Quality	Runs
9	80
8	116
7	96
6	34
5	66
4	18
3	12
2	7
1	2
0	1

Table 15 gives the distribution of runs by rank. The distribution of runs by rank is pretty close in shape to that by quality. This is because quality of run is a major factor influencing the assignment of rank to a run.

Table 15. Distribution of Experiments by Rank

Category of Rank	Runs
Best in HC type	10
Best of kind	167
Satisfactory	173
Supporting	42
Some Problems	22
Not for modeling	18

The distribution of runs by processing status is given in Table 16. Note the distinction between "Not to be processed" runs and "Unprocessed" runs. The former are runs which have been judged to be unsuitable for model testing, whereas the latter are runs which have not been processed but are suitable for model testing.

Table 16. Distribution of Experiments by Processing Status

Code	Processing Status	Runs
S	Fully processed; segfile	201
P	Partially processed	56
L	Partially processed; low priority	45
N	Not to be processed	20
U	Unprocessed run	110

Table 17 shows the distribution of runs by percent sunlight. Since most of the runs were conducted during summer months when sunlight is abundant, the distribution of runs by sunlight skews towards the higher percentage.

Table 17. Distribution of Experiments by Sunlight

Sunlight	Runs
0 - 10	31
11 - 20	13
21 - 30	15
31 - 40	19
41 - 50	28
51 - 60	30
61 - 70	44
71 - 80	54
81 - 90	90
91 - 100	108

4

Overview of OSCECIS

We have discussed the features of the experimental conditions databases in Chapter 2 and 3. In this chapter and next, we will look at the Turbo Prolog program, OSCECIS, that handles the database. This chapter begins with an introduction to the three basic sections of a typical Turbo Prolog program. It then provides an overview of OSCECIS by presenting some of its special features and the principles underlying its design. The materials covered in this chapter lay the ground for the detailed discussions of the components of OSCECIS in next chapter.

Sample Turbo Prolog Program

To provide some common language for our discussions in this and latter chapters, it is desirable to look at a simple Turbo Prolog program at this point.

A sample Turbo Prolog program is shown below. The numbers preceding the lines are for reference purposes. They are not part of the program. The wordings enclosed between "/*"and "*/"are comments.

Lines 2, 7, and 12 in the example contain keywords which prefaced the three basic sections found in a typical Turbo Prolog program.

```
1          /* Sample Turbo Prolog Program */
2 DOMAINS
3   person, author = symbol
4   title = string
5   publication = book(author, title)
6
```

```
 7 PREDICATES
 8   father(person, person)
 9   reads(person, publication)
10   grandfather(person,person)
11
12 CLAUSES
13   father(tom,bob).
14   father(bob,george).
15   father(george,bill).
16   reads(tom, book(knuth,''The Art of Programming'')).
17   reads(bob, book(dickens, ''The Tale of Two Cities'')).
18   reads(george, book(tolstoy, ''War and Peace'')).
19
20   grandfather(X,Z) :- /* The grandfather of X is Z if */
21      father(X,Y),     /* the father of X is Y and     */
22      father(Y,Z).     /* the father of Y is Z         */
```

The DOMAINS section is a mirror image of the TYPE section in a Pascal program. In the example, line 3 declares two domains, person and author, which consists of elements from the standard domain type *symbol*. Line 4 declares a domain, title, which consists of elements from the standard domain type *string*. Other standard domain types available are *integer*, *real*, and*char*. Line 5 declares a compound domain, publication, which consists of elements from the structured domain type, book(author, publication), defined by the user.

The PREDICATES section corresponds to the VAR section in a Pascal program but with a major distinction: It declares the relations known to exist between the objects in the program through the predicates which take the forms:

```
    predname
or
    predname(domain1,domain2,...,domainN)
```

where predname stands for predicate name and domain1,...,domainN stand for standard type or user-defined domains. Lines 8, 9, and 10 contain examples of predicate declarations.

The CLAUSES section corresponds to the program body in a Pascal program. The section contains facts and rules describing the relations declared in the PREDICATES section. In the example, line 13 through line 18 contain 6 assertions which

are called facts. Line 20 through line 22 contain a different type of assertion called a rule.

In this report, we will use the word predicate (and avoid using the word clause) to refer collectively to both facts and rules in a Turbo Prolog program. When a distinction between facts and rules is necessary, we will use the words "facts" and "rules" directly.

The only rule in the sample program is an example of Horn clauses which take the form:

```
A  if  B1 and B2 and ... Bn.
```

where A is the *head* of the rule and B1 and B2 and ... Bn is the *tail.*

In contrast to a Horn clause, a non-Horn clause takes the form:

```
A1 or A2 or ... Am  if  B1 and B2 ... Bn.
```

Note that a non-Horn clause contains at least two *heads* separated by a logical *or.* Although a non-Horn clause can always be converted to a logically equivalent Horn clause by moving the extra *heads* to the *tail* while negating them, the conversion is not unique in the sense that there is more than one way to choose the it heads to be moved over. This is the major reason why a language based on non-Horn clauses is inefficient in terms of computation time, although this type of language is more expressive than Prolog. As an example, the following non-Horn clause:

```
A1 or A2  if  B1 and B2 and ... Bn.
```

can be converted into either of the following two Horn clauses:

```
A1  if  not(A2) and B1 and B2 and ... Bn.
```

or

```
A2  if  not(A1) and B1 and B2 and ... Bn.
```

as a result of two possible ways to bring the *heads* to the *tail* portion.

Overall Organization of OSCECIS

The main objective of developing OSCECIS is to make the information contained in the database easily accessible. To achieve this end, the window facility of Turbo Prolog was used extensively to produce a series of menus. An interactive session with OSCECIS is almost equivalent to choosing appropriate panels from a series of pop-up menus. Whenever the user makes a mistake, he or she can easily back up by striking the *Esc* key as many times as necessary without causing any adverse side effect or even a failure to the system. Other handy facilities that OSCECIS provides are: access to the DOS commands, access to an editor, an online tutorial containing instructions on how to use the system, database updating facilities, a database diagnosing tool, and access to the eight types of codes use in the database. These features, together with the ease with which the users can get information out of the database, make OSCECIS a user-friendly system.

Figure 7 shows the main menu of OSCECIS. The panels in the box contain the ten options available in the system. In this section, we will discuss in detail the options which are comparatively simpler than the two particular options,"Query the database"and"Update the database". The last two options constitute the core of OSCECIS and are fairly complicated. They will be discussed separately in the next two sections.

Tutorial

The online tutorial is stored on a text file. It is invoked by choosing the"Tutorial" panel in the main menu of OSCECIS. The content of the tutorial can be inspected (but not altered) using cursor control keys (*e.g.* arrow keys to move one position at a time, *Home* to move to the first position of a line *etc.*), screen control keys (*e.g. PgUp* to go up one page of screen), or function keys (*e.g. F2* to go to a line, *F3* to search for a character string). In addition, the window which displays the tutorial can be resized by following the instructions shown in the highlighted area below the window. These nice facilities are provided for free with a simple call to a built-in predicate, display, of Turbo Prolog.

Access to DOS Commands

This option is activated by choosing the"DOS commands" panel in the main menu. The OSCECIS will be suspended under this option to allow access to the system commands. To go back to OSCECIS, simply type EXIT to the DOS prompt.

Figure 7. Main Menu of OSCECIS

Editor

The editor is accessible by choosing the"Editor"panel in the main menu. Besides text entering, the editor allows uses of cursor control keys to move around the window while editing. Many function keys can also be used to achieve certain functions such as copying, searching, or moving text strings. A second editor can also be brought to the currently active editor window to achieve viewing or copying of text from other files. An online help on the use of the editor is also available by choosing the *F1* key from within the editor. A summary of the functions of the editor is given in Appendix A.

Loading and Saving Database

Before querying can take place, the user should load the database to OSCECIS by choosing the"Load the database"panel in the main menu. When the option is taken, OSCECIS will call a built-in Prolog predicate, consult, which will load three files, *ec.cod*, *ec.db1*, and *ec.db2*, to the system. The first file contains mainly codes and corresponding attribute names used in the database. The second file

contains the experimental conditions of each run in the database. The third file contains mainly concentrations of NO_x, and identifications and concentrations of HC species/mixtures used in the two sides of the chamber for each run in the database. A review of Figure 6 is helpful at this point. The reasons for breaking up the run records into two files are primarily due to the considerations of execution efficiency and of the limitation imposed by the Turbo Prolog system on the size of a file to 64 kilobytes. We will come back to these in next section.

The"Save the database" option should be invoked whenever the user wants the changes made to the database to be saved. When the option is selected, the updated copy of the database will be saved to files *ec.db1* and *ec.db2*, and the old database will be moved to files *ecdb.db1* and *ecdb.db2* respectively.

The users are allowed to choose the"Load the database" panel only once in the entire interactive session. The"Save the database on file" option, on the other hand, may be selected as many times as necessary in a session.

Viewing the Database Codes

The codes and corresponding attribute names in the file *ec.cod* mentioned previously can be viewed by choosing the"View the database codes"panel in the main menu.

When the panel is selected, a menu containing eight options for viewing the database codes as shown in Figure 8 will appear. The first two types of codes, species and series codes, are especially useful in the process of querying the database. Since the first two types of codes can not fit into a screen, two separate text files, *ec.spe* and *ec.ser*, were created to contain the respective codes and names. The species codes are arranged in increasing order of species names whereas the series codes are arranged in increasing order of the codes. When either type of codes is chosen for viewing, OSCECIS will display the contents of the chosen type of codes in the same way it does to display the text file containing the tutorial. The users can inspect the codes and corresponding names the way they inspect the tutorial file by using either cursor or screen control keys, or other function keys.

The remaining six types of codes are all relatively short. Upon request, the codes and names of each type will be extracted from the *ec.cod* file and printed on the currently active window. Since the codes can be inspected by just a glance, neither cursor control key nor screen control key is needed.

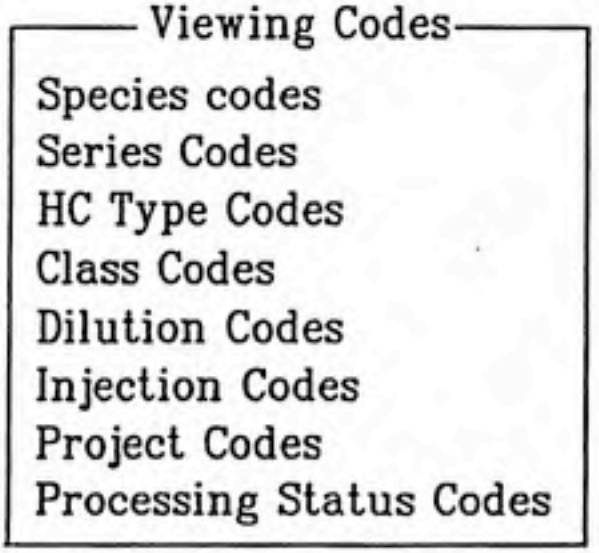

Figure 8. Menu for Viewing the Database Codes

Diagnosing the Database

To check the possible errors in the database, the"Diagnose the database" panel should be chosen. Upon being activated, OSCECIS will collect the eight types of codes from the file *ec.cod* and put them into eight different lists. It then picks up each record (recall that each record is divided into two parts) and examines the fields against the lists. Whenever a certain field contains a code which can not be found in the corresponding list, a message will be generated, telling which field of which record is possibly in error. The word"possibly"was used in the last sentence because the error may be due to the codes in the *ec.cod* file instead. This happens when new species or new conditions was used in the new experiments but their corresponding codes have not been added to the *ec.cod* file by the time when the interactive session is taking place. This option can, thus, help maintain the consistency among various database files.

Miscellaneous

The miscellaneous option gives the users a chance to look at the distributions of

species or series types used in the database. Although the distribution of individual species or series types can be obtained through querying the database, the numbers of species (over 200) and series types (40) available in the database make individual queries of each type a tedious task. This option provides an easy way to look at the two distributions. The information obtained through this option is especially useful for the experimenters in planning new chamber runs. The results obtained through this option can be either sent to screen, line printer, or text file. Table 12 shows the number of chamber sides containing species obtained by this option.

Querying the Database

The"Query the database"option constitutes the core of OSCECIS. Figure 9 shows the four basic query types available.

The first type, query by species, allows a quick way to query the database. The users are only asked to provide one to three species codes for the species of interest.

The second type, query by series, is similar to the first type except that the users are asked to provide a single series code instead.

The third type, query by dates, allows the users to search for any number of records by the keys (run dates). Alternatively, the users can search for runs within any period of time by supplying two dates, in chronological order, to the system. Records containing experiments in a given month of different years can also be retrieved easily using this option.

The last type of query, detailed query, is quite involved. Under this option, a series of menus corresponding to relevant fields in the database will be displayed. The users response by choosing appropriate panel from each menu until they run through the whole series of menus. Their responses to the menus determine what records will be retrieved from the database.

Despite all the differences among the query types, all queries undergo three basic steps illustrated in the following sections.

Step 1: Query Posing

In this step, the users are expected to response to each pop-up menu by choosing appropriate panel using arrow and *return* keys, or they will be asked to enter codes or numbers depending on what types of query they chose and what choices they made in earlier menus. The responses from the users will be entered to the database

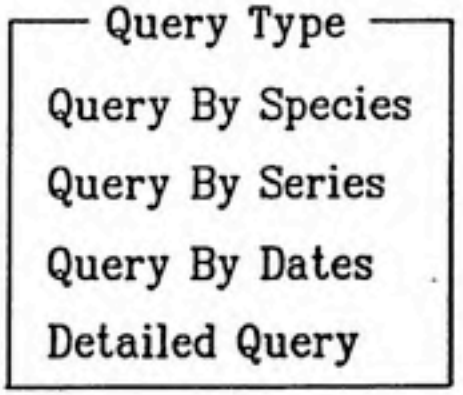

Figure 9. Menu for Four Query Types

and remain there for the duration of each query. These responses are organized into three types of templates.

The first type of templates, which we will call w-templates, contain the species IDs associated with the HC species/mixtures ID's in the second part of the record. The w-templates take the form:

```
w(IDTYPE)
```

where IDTYPE is a compound domain which consists of elements from the structured domain type, specID(STRING,INTEGER).

The second type of templates, which we will call x-templates, contain responses associated with the fields in the first part (except run date and rank) of each record in the database (see 6). The x-templates take the form:

```
x(EXPCOND)
```

where EXPCOND is a compound domain which consists of elements from the structured domain type, class(STRING), dilute(STRING), inject(STRING), hc-type(STRING), proj(STRING), proces(STRING), and series(STRING).

The third type of templates, which we will call y-templates, contain responses associated with the fields in the second part of each database record as well as responses associated with the last field, rank, from the first part of the record. The y-templates take the form:

```
y(RELATION)
```

where RELATION is a compound domain which consists of elements from the structured domain type, eq(STRING,TYPE), lt(STRING,TYPE), gt(STRING,TYPE), and range(STRING,TYPE,TYPE). The TYPE is itself a compound domain which consists of elements from the structured domain type, r(REAL) and i(INTEGER).

In brief, the w-templates have to do with the ID fields in the records. The x-templates have to do with the database fields declared as character strings. The y-templates have to do with the database fields declared as either real or integer types.

Step 2: Templates Collecting and Records Searching

The appearance of the message "Searching... please wait" on the screen marks the beginning of this step. Different searching strategies are used for different types of query.

For the first query type, query by species, only the y-templates containing the species codes are collected and used to match the fields containing the HC ID's of each record in the database.

The second query type, query by series, activates only the collection of the single x-template containing the requested series code which is, subsequently, used to match the series field of each record in the database.

For the third query type, no template collection activity takes place since the requested dates are passed directly to the searching mechanism which matches the key (run date) of each record against the dates passed to it.

The last query type, detailed query, activates, in this step, the searching mechanism to carry out a series of actions according to the algorithm outlined below:

a) collect all the templates available;

b) if species IDs are specified by the user, go and search the second part of each database record for the species with the same ID's and their corresponding concentrations (if the users also place a restriction on it);

c) if restrictions are also placed on other fields in the second part of the records (*i.e.* NOx, number of HC instruments used, quality of run, and sunlight), the runs found in b above should also satisfy these restrictions; otherwise reject the runs;

d) if the users place further restrictions on the first part of the database records (*i.e.* class, dilution, injection, HC type, processing status, series, and rank), the runs found in c above should also satisfy these restrictions; otherwise, reject the runs.

After the searching step, the successfully matched records will be sent to solution templates, called z-templates, for outputting purposes. The time taken to search for matched records will also be shown on the screen. We will explore the searching mechanism in greater depth in next chapter.

Step 3: Printing Results

The appearance of the "Output device" menu on the screen signals the readiness of OSCECIS to output the results. The users can choose to send the outputs to either screen, line printer or file. When the output device is chosen, another menu will appear which shows the eight choices of arranging the outputs for the given query. In brief, for each query, there are three choices of output device; for each output device, there are eight ways to arrange the output information. The users are free to choose any combination of output device and form. Multiple copies of output to any device are also allowed.

The eight output choices available are shown in Figure 10. "Simple output (date only)"will display the run dates grouped by years. The total counts of runs for the given query is also shown. Other output choices display the contents, according to a fixed format, of each run grouped by the particular choice. Subtotal of runs is also shown for each grouping of runs. Appendix D contains the sample outputs of the ethylene runs grouped by series and by HC type.

The solution templates, z-templates, contain the first portion of each successfully matched run record. Each z-template was declared as a compound structure which takes the form:

```
z(ec(Date,Class,Dilute,Inject,HCtype,Proj,PcSt,Ser,Rank))
```

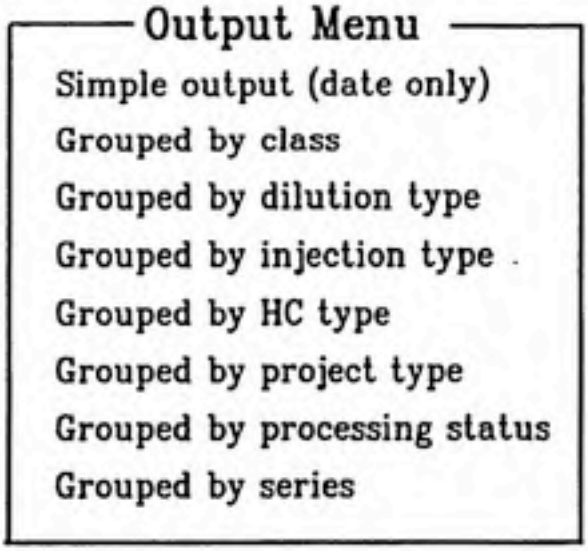

Figure 10. Menu for Eight Output Choices

Note that the inner structure is essentially the first portion of a database record (see Figure 6).

The attributes of the fields in the inner structure of the z-templates are used to group the outputs according to the following steps:

a) The attributes of the given field corresponding to the output choice (*e.g.* the Class field corresponds to the "Grouped by Class" option, the Dilute field corresponds to the "Grouped by dilution type" option and so on) are collected in a list.

b) The duplicated elements in the list are trimmed so that the new list contains essentially a set of the unique attributes of the given field in the z-templates.

c) The elements in the trimmed list are then used to group the runs in the z-templates.

Figure 11 shows an example of the steps taken when the "Grouped by series" option was chosen. We will examine these steps in greater detail in next chapter.

Solution Templates

```
z ( ec ( "770603", "0", "N", "I", "MB", "ET", "S", "29", 10 ))
z ( ec ( "790731", "0", "N", "I", "1U" "ET", "S", "18", 51 ))
z ( ec ( "790802", "0", "N", "I", "1U" "ET", "S", "18", 6 ))
```

Step a. ["29", "18", "18"] ← Original list

Step b. ["29", "18"] ← Trimmed list

Step c. Formatted output ordered by series:

```
-------------------------------------------------
  Date     CDI HC Ser Proj Q I P Sun     Rank
-------------------------------------------------

>>>>> For the Series type of 29 Addition, Aldehyde <<<<<

03-Jun-77 ONI MB  29    ET  7 0 S  95   Best of kind
          Red  Side   0.351  NOx  1.68    PPROPYLENE
                                  0.50    FORMALDEHYDE
          Blue Side   0.345  NOx  1.69    PROPYLENE

Number of runs of this series is 1

>>>>> For the Series type of 18 Reactivity Comparison <<<<<

31-Jul-79 ONI 1U 18    ET  7 0 S  77   Supporting
          Red Side    0.513  NOx  1.69    FORMALDEHYDE
          Blue Side   0.513  NOx  5.13    PROPYLENE

02-Aug-79 ONI 1U 18    ET  7 0 S  51   Best of kind
          Red Side    0.205  NOx  1.49    PROPYLENE
          Blue Side   0.205  NOx  1.01    FORMALDEHYDE

Number of runs of this series is 2
```

Figure 11. Steps of Grouping of Runs in Solution Templates by Series

Overall Picture

Figure 12 contains the schematic diagram which summarizes the processes of the four types of query available in OSCECIS. The boxes enclosing"Enter"as the first word of their statements require the users to key in either codes, real numbers, or integers. The boxes enclosing"Assert"as the first word of their statements require the users to choose the panels of interest by using arrow and *return* keys.

Updating the Database

The"Update the database"option allows the users to manipulate the records in the database. Figure 13 shows the four basic options available in OSCECIS.

Display Record

The first option, display record, allows the display of the content of a database record identified by the run date provided by the user. The"content"refers to the actual attribute of each field in the record as it appears in the database. For example, the HC species formaldehyde is represented by its code 139 in the database. The"Display record"option will display the code 139 instead of the species name"FORMALDEHYDE". The meanings of the codes can be easily found out by returning to the main menu and choosing the"View the database codes" option. When the record with the date supplied by the user is not found, a warning message will be given.

Delete Record

Choosing this option will result in the deletion of the record with the given run date from the database. since this option can be destructive, the users are asked to confirm the deletion after they have typed in the run date. The absence of the to-be-deleted-record in the database will not cause any warning message to be displayed because, in this case, the option will not produce any destructive effect anyway.

Insert Record

The"Insert record"option will insert into the database a record with the run date given by the user. If a record identified by the given run date already existed in the database, a warning message will be issued and the insertion aborted. Otherwise, the user will be asked to enter the attributes of all the fields in the record through answering a series of questions and through choosing relevant panels in a series of pop-up menus. If the user makes a mistake in this process (*e.g.* enter an alphabetic

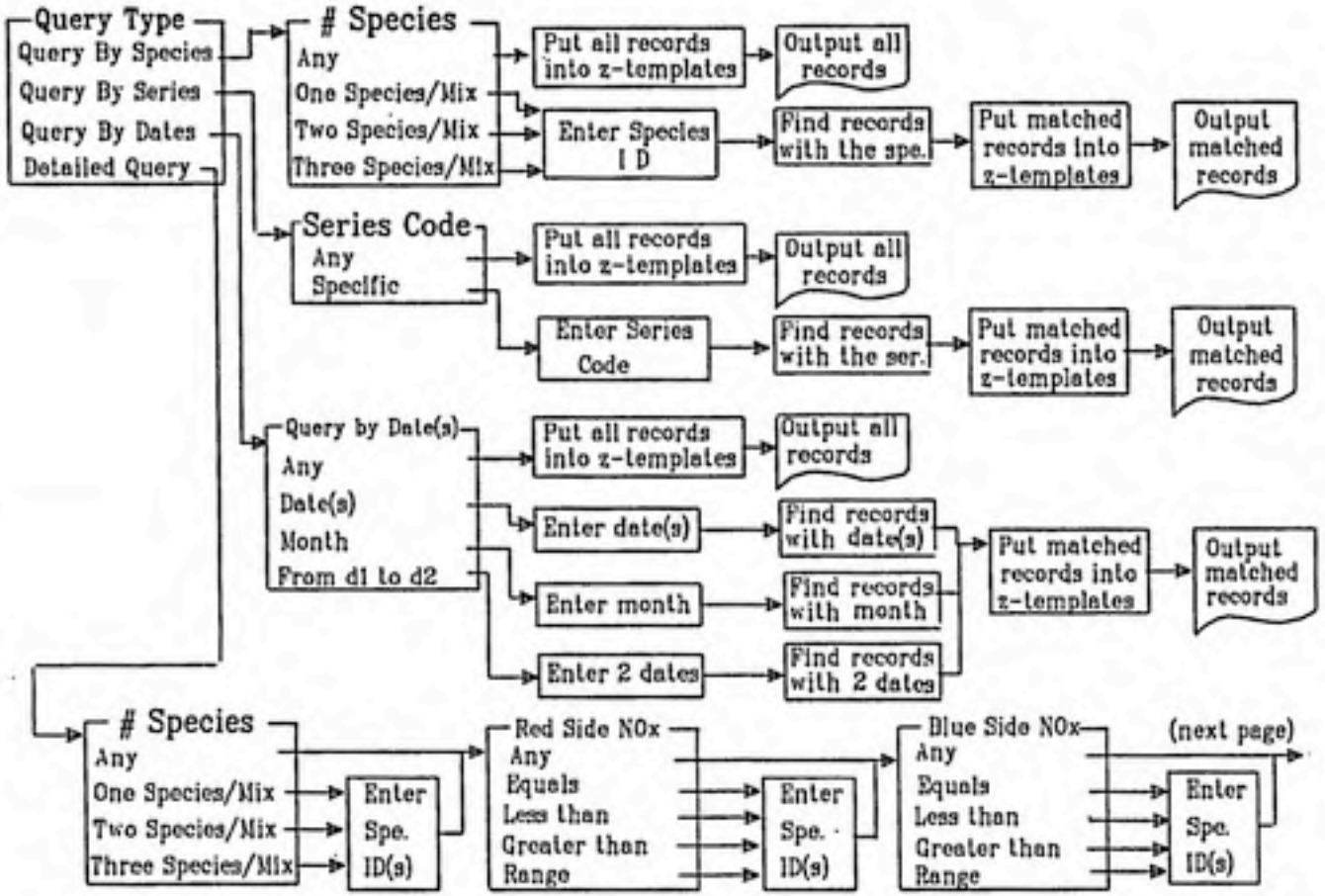

Figure 12. Flows of Four Query Types in OSCEIS

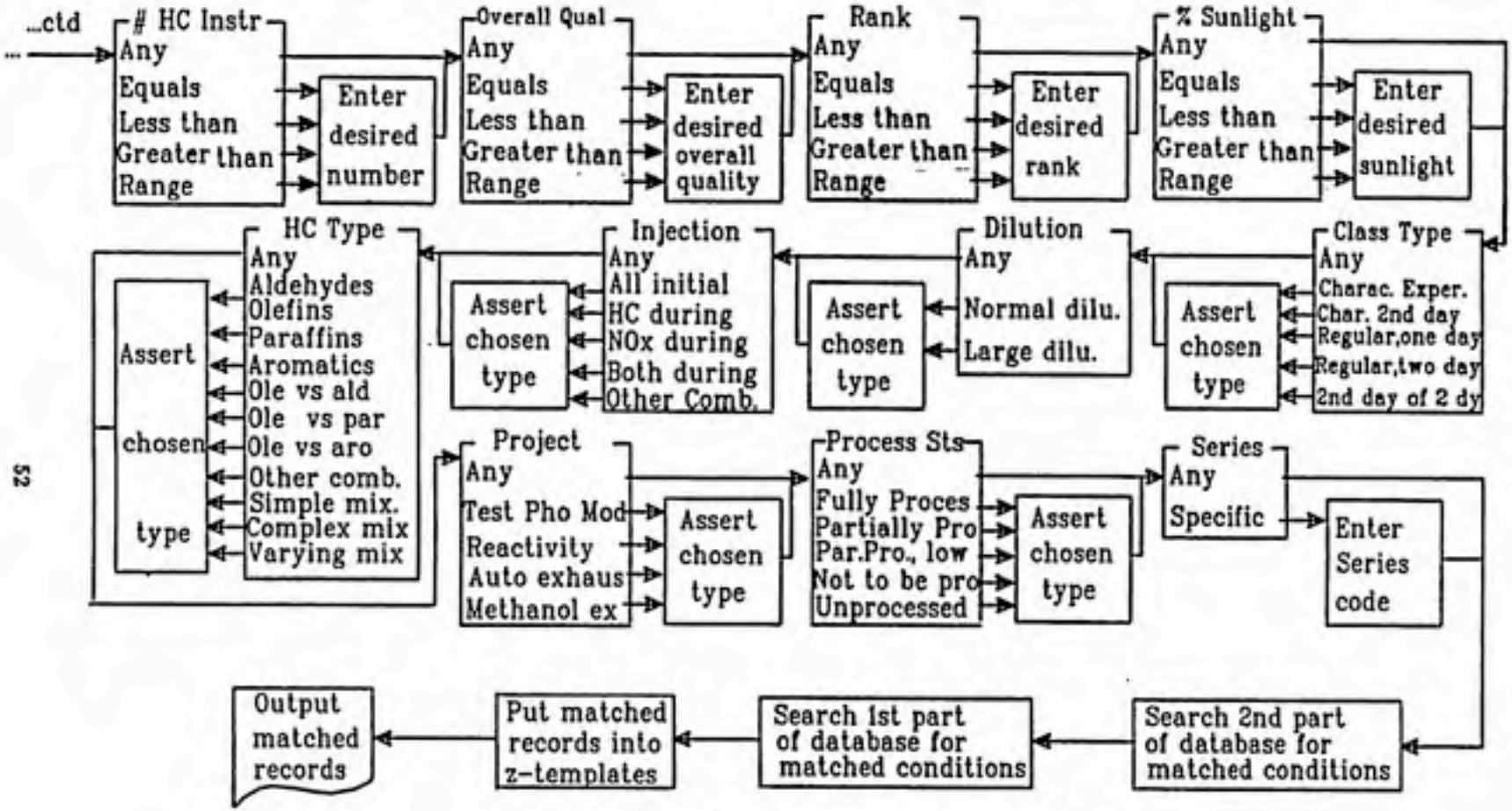

Figure 12. cont. Flows of Four Query Types in OSCEIS

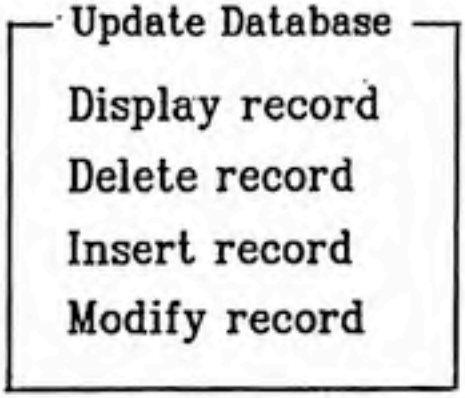

Figure 13. Menu for Updating Database

character to where a numeric value is expected), the insertion will be aborted. At the end of question answering and panel choosing, the user will be asked to confirm whether his or her inputs are correct. If the answer is positive, the new record will be inserted into the database.

Modify Record

The "Modify record" option allows the user to modify the attributes of any fields of any chosen database record. The user begins with typing in the run date. The system responses with checking the database to see whether a record with the given date already existed. If no such record exists in the database, a warning message will be issued and no modification will take place. Otherwise, the user will be prompted by a series of pop-up menus corresponding to each field of the record of interest. The first panel, "Not to be modified", in the menus should be chosen if the user does not want the attribute of the corresponding field to be changed. If the "To be modified" panel is chosen, a box will be displayed and the user will be asked to enter the new attribute for the corresponding field.

For those fields having only a few attributes (*i.e.* Class, Dilution, Injection, HC Type, Project Code, and Processing Status), the relevant attributes will be displayed in the panels of the respective menu and the user can choose the new attribute for the given field directly from the menu. Again, after the user has run

through the series of menus, he or she will be asked to confirm whether the inputs are correct. A positive response will cause the old record to be deleted from and the new record with modified fields to be inserted in the database.

Overall Picture

Figure 14 shows the schematic diagram summarizing the flows of the options of updating the database.

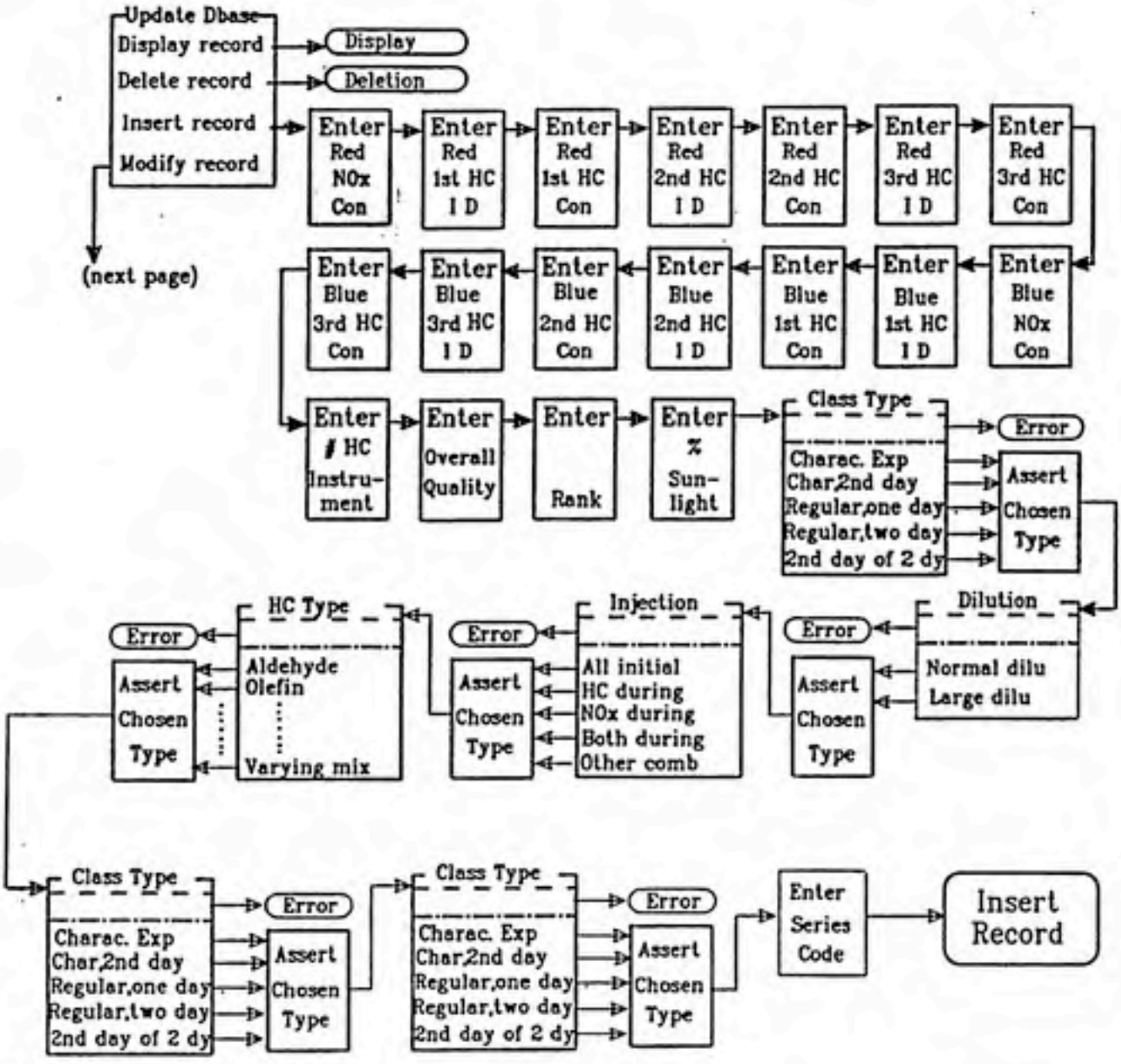

Figure 14. Flows of Four Updating Database Choices in OSCECIS

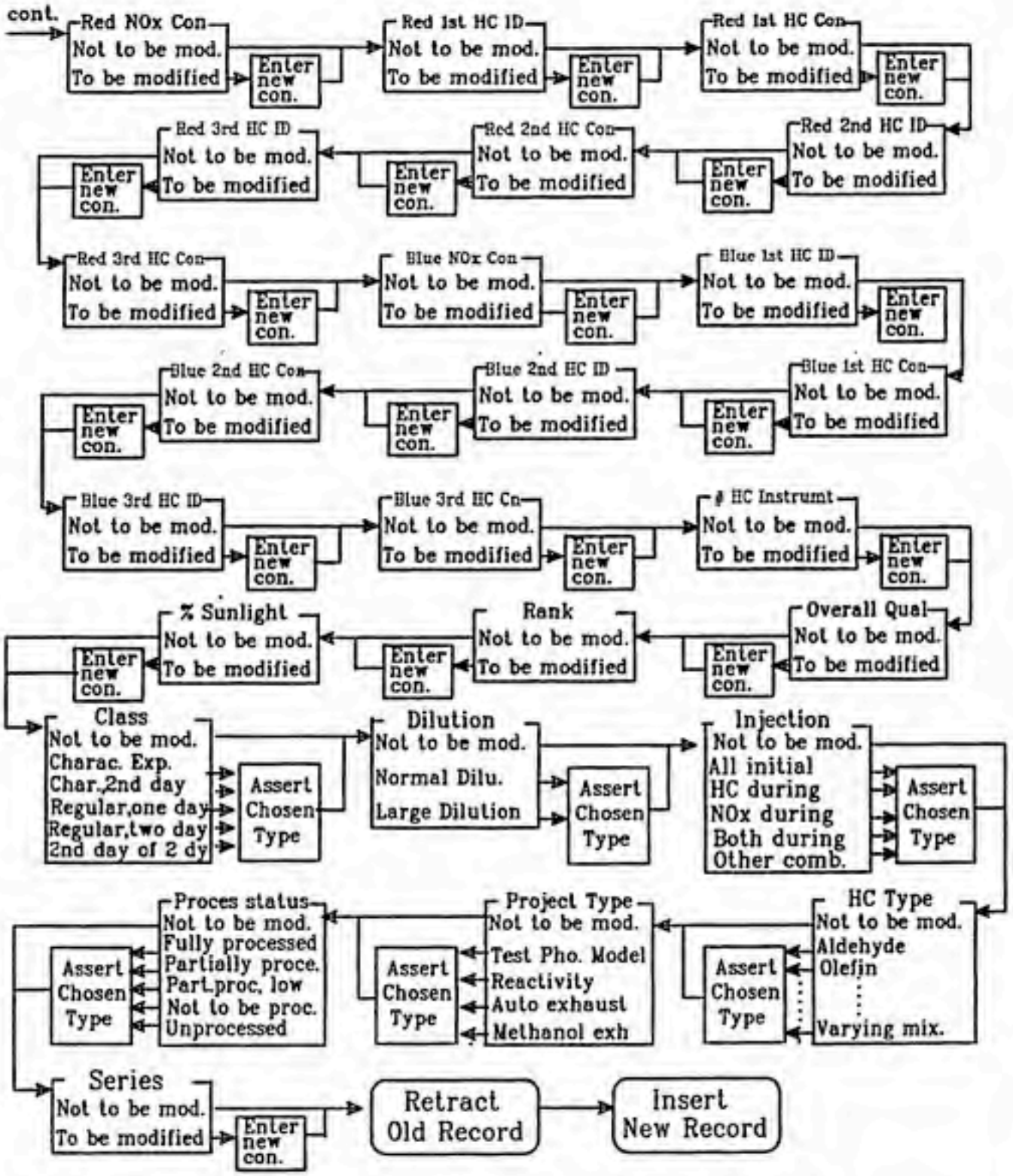

Figure 14. cont. Flows of Four Updating Database Choices in OSCECIS

5

Components of OSCECIS

This chapter begins with a brief introduction to the pattern-matching facilities of Prolog language by citing several examples of matched and unmatched patterns. It then proceeds to introduce the hierarchy of predicates in OSCECIS as a supplement to the discussions in previous chapter. Finally, a detailed description of selected components of OSCECIS will be presented with a goal to provide some insight on how the system works. A knowledge of Prolog is helpful but not required.

More About Turbo Prolog

To make our later discussions easier, we will pick up some terminologies while looking at some representations commonly found in a Turbo Prolog program.

As mentioned in Chapter 4, a predicate in Prolog language can take the form of either a predicate name or a predicate name followed by a pair of parentheses containing one or more arguments. Throughout this chapter, we will use the word "parameter(s)" to refer to the argument(s) in the parentheses. Each parameter can take the explicit form of any domain type (including standard domain types) as declared in the domain declaration section of the particular program or it can be a variable. Any terms starting with a capital letter in the CLAUSES section of a Turbo Prolog program is a variable. A variable matches a structure according to unification algorithm discussed in many textbooks on formal logic or logic programming.[15] The discussion of the unification algorithm is out of the scope of this report. However, to suit our needs, we will present some examples of matched and unmatched structures involving variables.

Table 18 shows six examples of matched structures.

Table 18. Examples of Matched Structures

	Structure 1	Structure 2	Variable Type
1.	X	any structure	X is of any type
2.	p(X)	p(a), p(b),...	X is of symbol type
3.	[a | _]	[a, b, c, d, e]	Both structures are lists of symbols
4.	List	[eq(L,r(B))|_]	List was declared as having structure 2
5.	eq(Labl,r(B))	eq(''HCcon1'',r(0.1))	Labl is of string type; B is of real type
6.	ec(cl(A),A)	ec(cl(''0''),''0'')	A is of string type

The underscores (_) in Table 18 represent anonymous variables. The structures which begin and end with brackets are called lists. The first structure on line 3 of 18is a short-hand representation of the second structure on the same line (they are both lists). The symbol "a" in the first structure on line 3 is called the *head* of the list and it matches the first element of the second structure. The underscore in the first structure of line 3 is called the *tail* of the list and it binds to the tail of the second structure (*i.e.* "_" binds to "[b, c, d, e]"). The list structure is one of the most powerful features of Prolog language.

Table 19 shows five examples of unmatched structures.

Table 19. Examples of Unmatched Structures

	Structure 1	Structure 2	Variable Type
1.	p(a)	p(b)	-
2.	[a | _]	[b, c, d, e]	-
3.	[lt(L,r(B))|_]	[eq(L,r(B))|_]	B's are of real type
4.	[eq(''C1'',i(5))|_]	[eq(''C2'',i(5))|_]	-
5.	ec(cl(A),A)	ec(cl(''0''),''T'')	A is of string type

The structures on line 1 do not match because the two symbols, a and b, do not. On line 2, the head of the first list does not match the head of the second list. The heads of the two lists on line 3 do not match because they are of different

types of predicates. On line 4, the string "C1" in the first list does not match the string "C2" in the second list although the two lists are of same structure. The structures on line 5 do not match because the two strings, "O" and "T", of the second structure are not the same.

Hierarchy of Predicates

All the predicates defined in OSCECIS can be arranged in a hierarchy according to what predicates they call and by what predicates they are called. Appendix B contains the hierarchy constructed according to this criterion. The hierarchy is presented in four blocks: main block, print block, menu block, and writeformat block.

The vertical lines represent the levels in the hierarchy. The predicates having fewer lines to the left of them are at a level higher than those having more lines. Accordingly, the predicate "mainmenu" occupies the top level of the hierarchy. Each pair of brackets encloses the name of the block referred to at that particular location. Each pair of parentheses encloses the parameter(s) of the predicate preceding it.

To explain the relationship of calling and called predicates, we define a **calling block** of a given predicate in the structure as the block whose height spans from the predicate of interest down until another predicate at the same or higher level is reached, and whose width encloses the whole structure to the right of the predicate of interest. For example, the calling block of *mainmenu* is the entire hierarchical structure in Appendix B. As another example, the calling block of the predicate *menu* is the entire menu block. As yet another example, the calling block of the predicate *ed* is the predicate itself.

In the hierarchical structure, a predicate calls only those predicate(s) at the next lower level in the calling block. For example, the predicate *view* calls only the predicates *repeat*, *menu*, and *view1*. As another example, the predicate *viewhc* calls the built-in predicate *fail*.

In Appendix B, only defined predicates are entered into the structure; database and built-in predicates (except the *fail* predicate) are not shown. In the structure, those predicates which call themselves are recursive in nature. Likewise, a predicate which calls another predicate which calls back the first predicate is also recursive. The iterative predicates are those which call the defined predicate *repeat* or the built-in predicate *fail*.

Although the hierarchy of predicates does not cover every aspect of the source program of OSCECIS, it holds the key to the understanding of the actual program. The reader is encouraged to at least take a glance at this construction in Appendix B before moving on to next section.

Description of Selected Predicates

A quick glance at the hierarchy of OSCECIS reveals that a detailed description of every predicate defined in the program will probably make the size of this report unbearable. As such, we will concentrate only on two groups of predicates which perform the tasks of searching and printing in the system.

Searching

Different query types adopt different searching strategies. We will investigate the searching strategy used in the most complicated query type, detailed query, by examining the predicates involved.

We assumed that the user has gone through the query posing step and the relevant w-, x- and y-templates have been inserted into the database (see section "Querying the Database" in Chapter 4). At this point, the predicate *detailfind* will be called. In the program, the only parameter of the predicate *detailfind* takes four values (0 to 3). The value passed to the parameter when *detailfind* is called depends on how many species ID's were specified by the user in the earlier step (remember that the user can specify 0 to 3 species when the "Detailed Query" option is chosen). For simplicity, we assume that the user has entered three species ID's in the query posing step and that the system has put the ID's in w-templates accordingly. As a result, the clause *detailfind(3)* as shown below is activated. Note that the numbers preceding the lines are for reference purposes. They are not part of the program.

```
1  detailfind(3):-
2     w(specID(''1st'',ID1)),
3     w(specID(''2nd'',ID2)),
4     w(specID(''3rd'',ID3)),
5     findall(Term,x(Term),Lis),
6     findall(Rel, y(Rel),List),
```

```
 7    rb(Date,RN,Rc1,R1,Rc2,R2,Rc3,R3,
             BN,Bc1,B1,Bc2,B2,Bc3,B3,IS,QA,SU),
 8    checkRB(List,''HCcon1'',ID1,Rc1,R1,Rc2,R2,Rc3,R3,
              Bc1,B1,Bc2,B2,Bc3,B3),
 9    checkRB(List,''HCcon2'',ID2,Rc1,R1,Rc2,R2,Rc3,R3,
              Bc1,B1,Bc2,B2,Bc3,B3),
10    checkRB(List,''HCcon3'',ID3,Rc1,R1,Rc2,R2,Rc3,R3,
              Bc1,B1,Bc2,B2,Bc3,B3),
11    rel_r(List,''redNOx'', RN),
12    rel_r(List,''blueNOx'',BN),
13    rel_i(List,''hcinst'', IS),
14    rel_i(List,''qual'',   QA),
15    rel_i(List,''sun'',    SU),
16    ec(Date,Cl,Di,In,HC,Pj,Pc,Se,Rk),
17    checkEC(Lis,Cl,Di,In,HC,Pj,Pc,Se),
18    rel_i(List,''rank'',  Rk),
19    assert(z(ec(Date,Cl,Di,In,HC,Pj,Pc,Se,Rk))),fail.
20  detailfind(_).
```

Lines 2 to 4 contain statements whose job is to match the w-templates containing the three species ID's asserted to the database in the earlier step. The ID's of the first, second, and third species are bound to ID1, ID2, and ID3 respectively. Line 5 contains a call to the built- in predicate *findall* which collects the inner structures of all the x-templates and put them into a list bound to the third parameter Lis. Similarly, line 6 contains a call to *findall* which collects the inner structures of all the y-templates and put them into a list bound to the third parameter List. Up to this point, all the templates have been collected and the system is ready to match the elements in the templates to the fields in the database records.

The statement on line 7 achieves the effect of fetching in, one in each cycle, the second part of database records, beginning from the first record in the database. The fields in the record being fetched in will bind to the corresponding variables on line 7. We will call this newly fetched record the current rb (for red and blue sides) record.

Lines 8 through 10 contain three calls to the predicate *checkRB* with different HC ID's (ID1, ID2, ID3), HC concentration labels ("HCcon1", "HCcon2", "HCcon3") and other 12 parameters bound to the HC ID's and HC concentrations of the current rb record. The *checkRB* predicate was defined as follows:

```
21  checkRB(List,Spe,ID,Rc1,R1,Rc2,R2,Rc3,R3, Bc1,B1,Bc2,B2,Bc3,B3):-
22    ID=R1, rel_r(List,Spe,Rc1), ! or
23    ID=B1, rel_r(List,Spe,Bc1), ! or
24    ID=R2, rel_r(List,Spe,Rc2), ! or
25    ID=B2, rel_r(List,Spe,Bc2), ! or
26    ID=R3, rel_r(List,Spe,Rc3), ! or
27    ID=B3, rel_r(List,Spe,Bc3).
```

The first task of *checkRB* is to find out whether there is a match between the parameter ID and any one of the six parameters, R1, R2, R3, B1, B2, B3, which bind to the six fields containing HC ID's from the current **rb** record. If such a match is not found, the predicate *checkRB* fails and the control is returned to the calling predicate *detailfind*. If such a match is found, the first statement on one of the lines numbered from 22 to 27 will be satisfied and the second statement on the same line will be activated. Note the regular patterns among the parameters of the clause *checkRB*. The first character of each parameter designates the chamber side where the HC is found ("R" for Red side and "B" for Blue side). The last character of each parameter designates the ordering of the HC in each side of the chamber (up to 3 species are allowed). Each parameter having "c" as its middle character binds to the concentration of the HC having same first and last characters (*e.g.*Rc1 binds to the concentration of the HC bound to R1).

The second task of *checkRB* is to check, by calling the support predicate *rel_r*, the concentration of the successfully matched HC against the restriction, if any, placed by the user on this HC. The predicate *rel_r* was defined as follows:

```
28  rel_r([],_,_):-!.
29  rel_r([eq( Label,r(B) )| _], Label, A) :- !, A= B.
30  rel_r([lt( Label,r(B) )| _], Label, A) :- !, A< B.
31  rel_r([gt( Label,r(B) )| _], Label, A) :- !, A> B.
32  rel_r([range( Label,r(B), r(C) )| _], Label, A) :- !,
      B<= A, A <= C.
33  rel_r([_| T], Label, A) :- rel_r(T, Label, A).
```

The first parameter binds to a list of elements from the domain RELATION (see section "Query Posing" in Chapter 4). The second parameter binds to a label identifying the database field of interest. The third parameter binds to the value of the database field of interest.

If the user never places restriction on any one of the fields associated with HC or NOx concentrations, number of HC instrument used, quality of run, sunlight,

and rank, the y-template will not exist and the first parameter, "List", of *checkRB* which is supposed to contain the inner structure of y-templates will be empty. In this case, line 28 is satisfied and predicate *rel_r* will return a "true" to the calling predicate *checkRB*.

If the user places at least a restriction on at least one of the fields mentioned above, the y-template(s) will exist and the first parameter, List, of *checkRB* will contain the inner structure(s) of y- template(s). Consider two cases. Case 1. The restriction placed on a field is in the first position of the list, which is the first parameter of *rel_r*, and the field, identified by the variable "Label", is currently being examined. In this case, the two labels in the heads of one of the clauses on line 29 through line 32 will be matched. The success or failure of the clause is then determined by whether its tail is true or false. If the field of interest satisfies the restriction placed on it, the tail will be true. Otherwise, the tail will be false. The cut, "!", in the tail of a given clause prevents the Prolog from exploring further on other clauses with the same predicate name. Case 2. The restriction placed on a field is not in the first position of the list of *rel_r* and the field is currently being examined. In this case, the clause on line 33 will be matched. The tail of the clause contains a recursive call to the predicate *rel_r* itself with the tail portion of the original list as its first parameter. This procedure repeats until the condition of case 1 above becomes true (*i.e.* until the restriction placed on the field currently being examined is in the first position of the list). The success or failure of the clause on line 33 depends on the subsequent recursive call which ultimately depends on the success or failure of one of the clauses on line 29 through line 32.

If the user places restrictions on some fields other than the field currently being examined, clauses on line 29 through line 32 will never have a match and be executed. The first clause that matches is the one on line 33, which calls back the predicate *rel_r* with the tail portion of the original list as its first parameter until the list becomes empty and the clause on line 28 is successfully matched. In this case, the call to the predicate *rel_r* will always be successful because, if no restriction is placed on the field, any value in the field of interest is acceptable.

The success or failure of the predicate *rel_r* determines the success or failure of the calling predicate *checkRB* which, in turn, determines the flows of the predicate *detailfind* on which we will now focus.

If any of the calls on line 8 through 10 fails, the system will backtrack to the statement on line 7 and pick up the next record in the database. The fields in the new record will bind to the parameters in the statement and another cycle

of callings to *checkRB* begins. If the three calls to *checkRB* are successful, two calls to the predicate *rel_r* and three calls to the predicate *rel_i* will take place on line 11 through line 15. The *rel_r* examines the restrictions placed on the fields corresponding to red side NOx concentration and blue side NOx concentration, whereas the *rel_i* examines the restrictions placed on the fields related to the number of HC instruments used, quality of run, and sunlight. The predicate *rel_i* is almost identical to the predicate *rel_r* (by replacing all the r's on line 28 through line 33, we get the definition of the predicate *rel_i*). The only different between *rel_r* and *rel_i* is that the former examines the relationship between the fields declared as real type and the corresponding real values placed by the user, whereas the latter examines the relationship between the fields declared as integer type and the corresponding integer values placed by the user. If any of these call fails, the system will backtrack to line 7. Otherwise, line 16 will be activated.

Line 16 achieves the effect of fetching in the first portion of the database record whose second portion has been successfully matched so far (notice that the two portions are linked by the variable Date). The fields of the first portion of the database record bind to the corresponding variables on line 16. We will call this portion of the database record the current ec (for experimental conditions) record.

Line 17, which contains a call to the predicate *checkEC*, is activated subsequently. The task of *checkEC* is to match the experimental conditions placed by the user against that in the current ec record. The *checkEC* predicate was defined as follows:

```
34  checkEC([],_,_,_,_,_,_,_).
35  checkEC([class(A)|T],  A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
36  checkEC([dilute(B)|T], A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
37  checkEC([inject(C)|T], A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
38  checkEC([hctype(D)|T], A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
39  checkEC([proj(E)|T],   A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
40  checkEC([proces(F)|T], A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
41  checkEC([series(W)|T], A,B,C,D,E,F,W) :-
      checkEC(T,A,B,C,D,E,F,W).
```

The first parameter binds to a list of elements from the domain EXPCOND (see section "Query Posing" in Chapter 4). The remaining seven parameters bind to all the fields (except run date) of the current ec record.

If the user places no restriction on any experimental condition, the x- template will not exist and the first parameter, Lis, of *checkEC* (on line 17) which is supposed to contain the inner structure of x- template will be empty. As a result, the clause on line 34 is satisfied and a "true" is returned.

If the user places at least a restriction on at least one of the experimental conditions, the x-template(s) will exist and the first parameter, Lis, of *checkEC* on line 17 will contain the inner structure of x-template(s). If a restriction placed on a given field is satisfied by the attribute of the field of the current ec record, one of the heads of the clauses on line 35 through line 41 will be successfully matched and its tail, which contains a recursive call to *checkEC* with the tail portion of the original list as its first parameter, will be activated. The recursive calls to *checkEC* continue until either line 34 is matched or no clause on lines 34 through 41 can be successfully matched. The former case happens when the corresponding fields of the current ec record satisfy all the restrictions placed by the user. A "true" will be returned. The latter case happens when a single field of the current ec record can not satisfy the restriction placed by the user to the field. A "false" will be returned in this case.

We will again focus on the predicate *detailfind* now. If the predicate *checkEC* returns a "false" to line 17, the system will backtrack to the statement on line 7. Otherwise, a call to *rel_r* will take place on line 18 to check the possible restriction placed on the last field, rank, of the current ec record. A failure of the call on line 18 will cause the system to backtrack, whereas a success of the call will cause the statements on the line 19 to be executed.

Being able to reach line 19 marks the "triumph" of the currently active database record. There are two statements on this line. The first statement is a call to the built-in predicate *assert* which puts the current ec record into the solution template. The second statement is a call to the built-in predicate *fail* which, as its name suggests, always fails and forces backtracking to previous statements. The system will eventually backtrack to line 7, where another fetching cycle begins.

The executions described so far repeat until the end of the database file is reached. Line 20 is subsequently activated. The clause on the line will always succeed and, thus, completes the searching step.

Printing Results

As pointed out in Chapter 4, there are three choices of output devices available for each query and eight choices of output forms available for each output device. We will investigate the strategies used in printing the results of a given query by examining the relevant predicates.

For simplicity, we assume that the solutions for a given query have been put into the z-templates and the user has chosen the desired output device. We will focus only on the "Ordered by series" option because it is similar to the remaining options available in the "Output menu" (except for the "Simple output" option which is much simpler then any other options).

As soon as the "Ordered by series" panel is chosen, the predicate *printBySeries* as defined below will be activated.

```
42  printBySeries :-
43    findall(Series,z(ec(_,_,_,_,_,_,_,_,_,_,Series,_)),L),
44    unik(L,L1),
45    heading,
46    printSeries(L1).
```

Line 43 contains a call to the built-in predicate *findall* which collects the series codes in all of the z-templates and put them into L which is a list. Line 44 contains a call to the predicate *unik* which throws away all the duplicated elements in list L and returns a trimmed list with unique elements to L1. Line 45 is a call to the predicate *heading* which prints the heading of the formatted output as shown in Appendix D. Line 46 is a call to the support predicate *printSeries* defined as follows:

```
47  printSeries([]).
48  printSeries([S|S1]) :-
49    check_ser(S,Series),
50    write(`` »»»» For the Series type of ``, S, Series,
      `` «««« \n ``),
51    writeformat(``series``,S),
52    findall(X,z(ec(X,_,_,_,_,_,_,_,_,_,S,_)),List),
53    listlen(List,Num),
54    write(`` of runs of this series is ``,Num),nl,
55    printSeries(S1).
```

If the list passed to the *printSeries* is empty, line 47 will be satisfied and the control will be returned to the calling predicate on line 46. Otherwise, line 48 will be matched and the statements on line 49 through line 55 will be activated in sequence.

Line 49 contains a call to the predicate "check_ser" which checks whether the series with code bound to S is in the file *ec.cod*. If so, the corresponding series name will be bound to the variable Series. Otherwise, the string " NEW SERIES!!!" will be returned as a warning to the possible inconsistency in the database. Line 50 prints the heading of the beginning of the current series which includes series code and series name from previous statement.

Line 51 contains a call to the predicate *writeformat* whose task is to print out all the runs having the current field attribute, in this case the series code bound to S, in the z-templates. The *writeformat* was defined as:

```
56  writeformat(Label, Field) :-
57    ec_field(Label,Field,
          Date,Class,Dilute,Inject,HC,Proj,Status,Ser,Rank),
58    rb(Date,
             RedNOx,RHCcon1,RHCcode1,RHCcon2,
             RHCcode2,RHCcon3,RHCcode3,
             BlueNOx,BHCcon1,BHCcode1,BHCcon2,
             BHCcode2,BHCcon3,BHCcode3,
             HCinst,Qual,Sun),nl,

59    code_date(Date,Date1),
60    code_rank(Rank,Rank1),
61    write(Date1,'' '',Class,Dilute,Inject,'' '',HC,
          '' '',Ser,'' '',Proj,'' '',Qual,'' '',
          HCinst,'' '',Status,'' '',Sun,'' '',
          Rank1),nl,
62    writerb(''Red  Side'',RedNOx,RHCcon1,RHCcode1),
63    writehc(RHCcon2,RHCcode2),
64    writehc(RHCcon3,RHCcode3),
65    writerb(''Blue Side'',BlueNOx,BHCcon1,BHCcode1),
66    writehc(BHCcon2,BHCcode2),
67    writehc(BHCcon3,BHCcode3),fail.
68  writeformat(_,_).
```

Line 57 contains a call to the predicate *ec_field* defined as follows:

```
69  ec_field(''class'', A, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
70  ec_field(''dilute'',B, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
71  ec_field(''inject'',C, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
72  ec_field(''hctype'',D, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
73  ec_field(''proj'',  E, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
74  ec_field(''status'',F, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
75  ec_field(''series'',G, X,A,B,C,D,E,F,G,R) :-
       z(ec(X,A,B,C,D,E,F,G,R)).
```

When *ec_field* is called with its first parameter bound to "series" and its second parameter bound to the current series code (both passed from the parameters on line 51), the clause on line 75 will have a match and its tail will pick up the first available run with the current series code (now bound to variable "G") in the z-template. All the fields in the run just picked up from the z-template are then passed, via the remaining parameters in the head of the clause on line 75, to the statement on line 57.

On line 58, the second portion of the run, identified by the same run date as the portion just picked up from the z-template, will be fetched in from the database.

Lines 59 and 60 convert the run date and rank to more readable forms. Line 61 through line 67 contain calls to predicates *writerb* and *writehc* which, together, print out the fields of the run according to a fixed format as shown in Appendix D.

The last statement on line 67 is a call to the built-in predicate *fail* which forces the system to backtrack to line 57 where another call to *ec_field* takes place. Line 75 is again matched and its tail will, at this time, pick up the next available run with the current series code in the z-template. The cycle repeats until no more run with the same current series code can be found in the z-templates. At that point the call to *ec_field* on line 57 will fail. Line 68 is subsequently reached and the control returns to line 51.

Line 52 contains a call to *findall* which collects the dates (bound to X) of all the runs with the current series code (bound to S) in the z-templates and put them

into List. The predicate *listlen* on line 53 counts the number of elements in List and returns the number to Num. Line 54 prints out the counts of the runs having the current series code.

Line 55 is subsequently activated. It contains a recursive call to *printSeries* with the tail portion of the list on line 48. The execution cycle repeats until the list becomes empty. Line 47 is subsequently satisfied and the control returns to line 46 where the whole sequence of printing results comes to an end.

6

Future Work

We have pointed out in Chapter One that OSCECIS will ultimately be incorporated into an expert system capable of assisting the photochemical kinetics model developers in testing their models, as well as carrying out automatic model evaluations. In this chapter, we will discuss the plan for the future work along this line.

What Is Next?

Judging by the ease with which the users can query the database, the short searching time, the tolerance of errors made by the user in the interactive session, and other handy helping tools, we are quite confident to say that OSCECIS has achieved its goal as a database management system for the experimental conditions database.

However, with the power that the implementing language, Turbo Prolog, possesses, one may, quite naturally, ask: Can the system do better? Is it possible to write a program which can automatically choose the best runs for model testing? How about model evaluations? Can a program make inferences from a set of testing results and tell the strengths and weaknesses of a given model? Is it possible to have a program which incorporates all the above features and behaves like an expert in the field of photochemical reactions modeling?

These are the questions to be addressed as we discuss the long-term goals of this project. It turns out that the points raised in the above questions fall well within the scope of an expert system we are planning to develop. But what is, in the first place, an expert system?

Features of Expert Systems

Any attempt to clarify the meaning of the term "expert system" is bound to elicit a long list of terms to be clarified in turn. To avoid being carried away, we will

be satisfied with the definition that an expert system is a software package which can perform the tasks of a specific domain at an expert's level. The word "specific" can never be over emphasized since it holds the key to the success of many expert systems.

Early artificial intelligent programs tended to be too broad in scope and consequently could solve only "toy" problems given the limitations of computer resources. It is not until the late 1960's that the importance of domain expert's knowledge to the performance of intelligent programs was appreciated. The recognition of this key factor finally brought about the first batch of commercial products of artificial intelligence which totally changed the image of AI approach as being only able to deal with "toy" problems. A new field known as expert system was born.

The following list was adapted from Forsyth.[16] It outlines the distinctive features found in many expert systems and is a good starting point to know what an expert system is:

1. An expert system is limited to a specific domain of expertise.
2. It can reason with uncertain data.
3. It can explain its chain of reasoning in a comprehensive way.
4. Facts and inference engine are clearly separated.
5. It is designed to grow incrementally.
6. It is typically rule-based.
7. It delivers advice as its output - not tables or figures, nor pretty video screens, but sound advice.

Basic Components of Expert Systems

Figure 15 shows a schematic diagram of four basic components of an expert system. Not all existing expert systems possess all the four components; but the majority of them do.

The Knowledge Base

The knowledge base constitutes the core of an expert system. The major difference between a database and a knowledge base is that the objects of the former are data, whereas the objects of the latter are entities and the rules describing the relations between the entities. Another difference between database and knowledge base is that the database system is passive in the sense that a piece of information is either present or absent in the database, and very little can be done about the missing

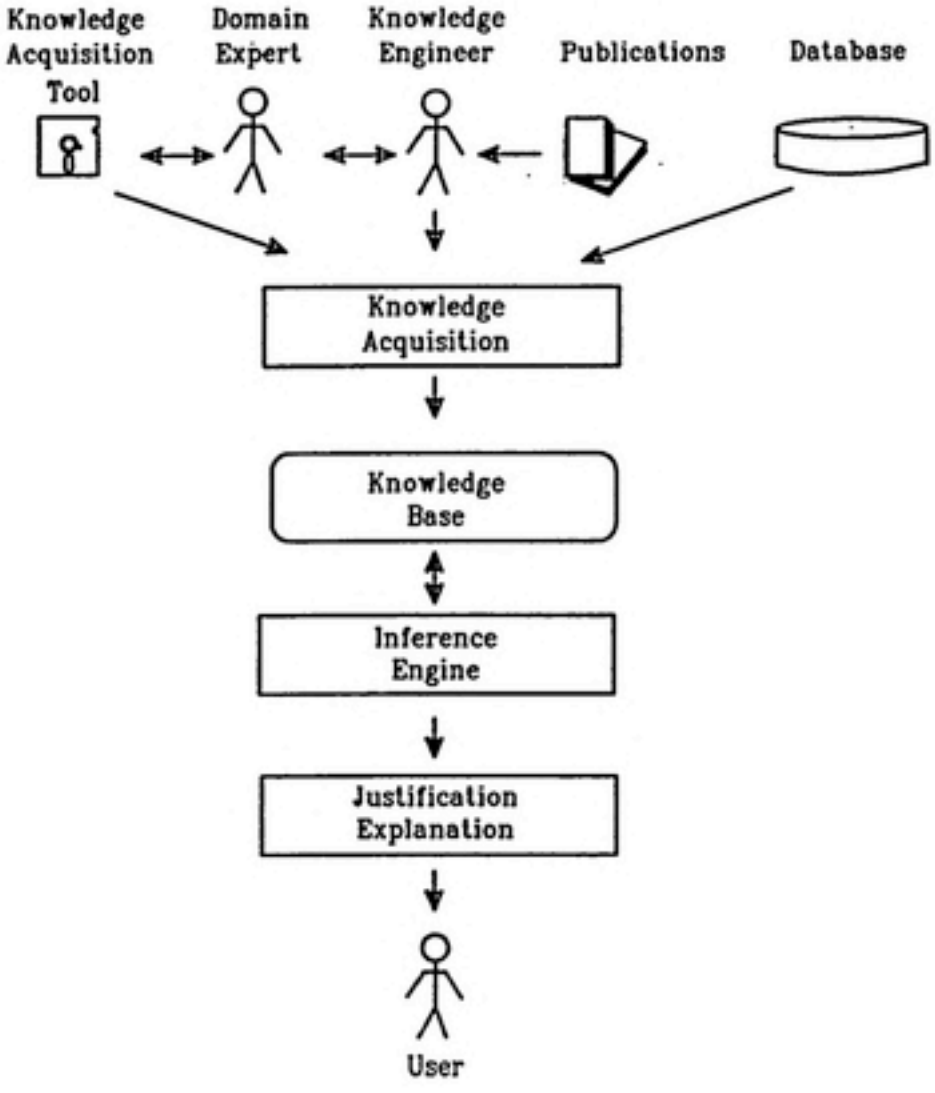

Figure 15. Four Basic Components of a Typical Expert System.

information, whereas the knowledge base system actively tries to fill in the missing information among the entities by applying the relevant rules to them.

Although the production rules, with their IF-THEN format, have been a favorite means of developing knowledge base for some time, other tools such as semantic nets and predicate calculus have also been used. Recently, Prolog has become popular as a powerful knowledge representation language.

The Inference Engine

The inference engine makes use of the rules in the knowledge base and the information provided by the user to find the objects which satisfy the constraints. There are basically three ways by which the inference engines are constructed: forward-chaining, backward-chaining, and rule- value method.

Forward-chaining involves reasoning from data to hypothesis, whereas backward-chaining starts with a hypothesis and attempts to find data to prove or disprove the hypothesis. The rule-value method is generally an improved backward-chaining method. It requests the information that will remove the most uncertainty from the system.[17] The rule-value method is theoretically better than the first two methods, however, it is more difficult to implement.

Justification / Explanation

A distinctive feature of an expert system is its ability to justify or explain to the user the actions it takes. It answers questions about why some conclusion was reached and why some alternative was rejected. Michie and others[18] have warned that the justification facility should not be regarded as an optional extra. It is now generally agreed that an expert system which can not explain its chain of reasoning to the user is unsatisfactory, even if it performs better than a human expert[16].

Knowledge Acquisition

The procedure of extracting knowledge from an expert and transforming it into computer readable form is called knowledge acquisition.[19] Traditionally, the process of knowledge acquisition involves intensive interactions between the knowledge engineer (programmer) and the domain expert over a long period of time. The interactions may take the forms of interview, discussion of domain expert's publications, or even taking classes from the expert. Knowledge acquisition has long been described as the bottleneck of expert system building because it is an extremely time consuming process.

To ease the task of knowledge acquisition, many attempts were made to shift parts of the responsibility from human to computer by teaching computer to learn concepts and rules. This line of research is known as machine learning. Although this approach looks promising, it is very much in the initial stage and many works remain to be done before it can contribute much to the task of knowledge acquisition in expert system building.

An alternative to the above approach is to develop intelligent editing program which understands the structures of the rules in the knowledge base and helps maintain the consistency of the rules in the knowledge base. The expert can interact with the program instead of knowledge engineer. When the knowledge base grows in size with time, the intelligent editing program will become more and more important.

Outline of Future Work

Long-Term Goal

My long-term goal is to develop an expert system capable of:

a) selecting appropriate chamber runs to test photochemical kinetics models;

b) carrying out automatic model evaluations by making inferences from the sets of testing results;

c) explaining the rules it used in the process of model evaluations at different levels of details;

d) acquiring new knowledge from the domain expert through an intelligent editor.

System Configuration

Figure 16 shows the configuration of the prospective expert system which is tentatively called ASKME for Atmospheric Simulation Kinetics Model Expert. The loop in the figure represents the cycle of testing of one chamber run.

Selecting Runs

Each testing cycle begins with the selection of the appropriate run from the experimental conditions database. The criteria used to judge the degree of "appropriateness" are the results of previous testing (including results from previous interactive session, if any), the availability, quality, and rank of the particular type of runs in the experimental conditions database, and the conditions contained in the knowledge base for run sequence.

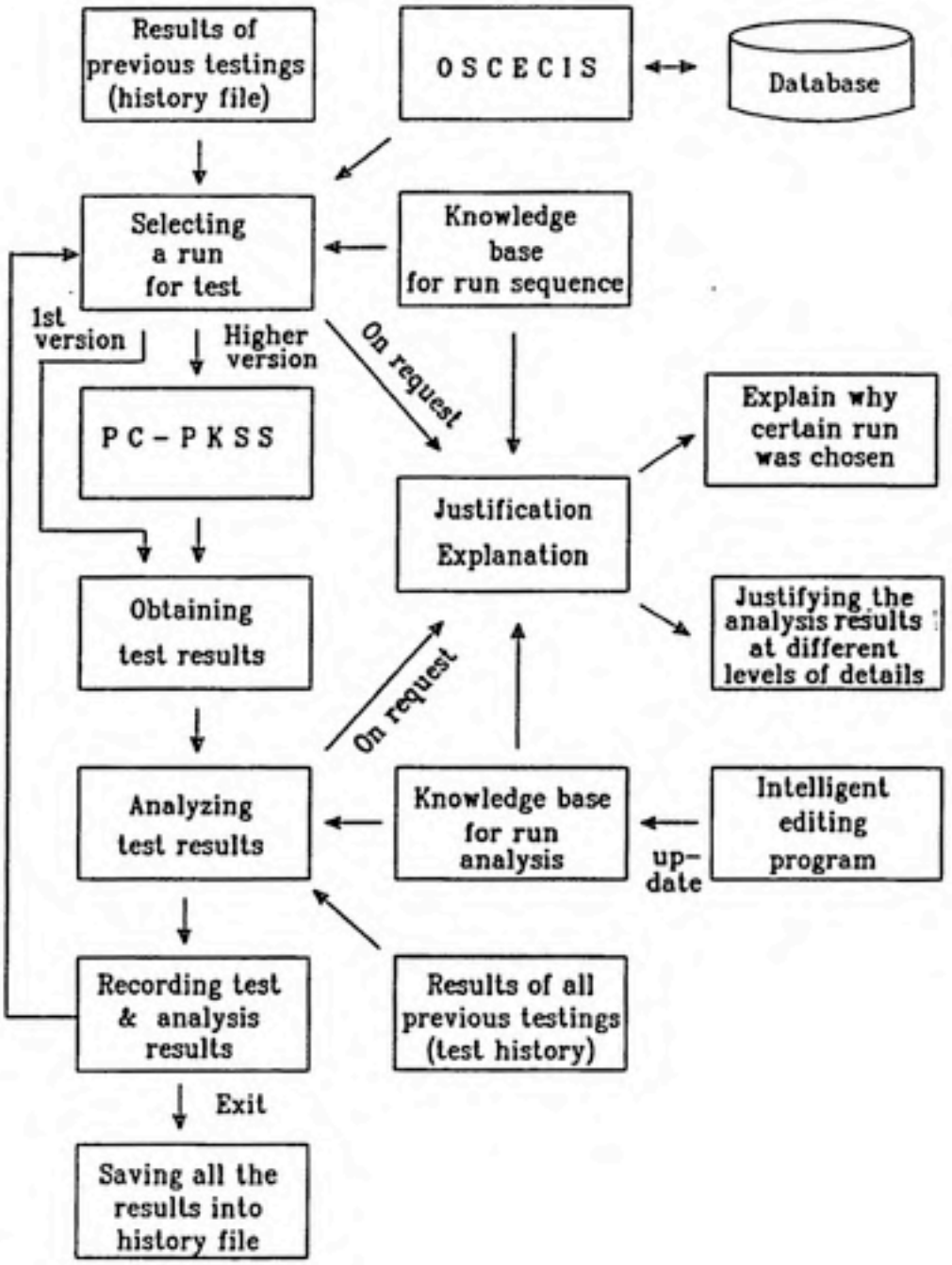

Figure 16. Configuration of ASKME (Atmospheric Simulation Kinetics Model Expert).

A prototype has been built to test the feasibility of automatic selection of runs by the computer. The prototype incorporates the searching mechanism of OSCECIS and a small knowledge base containing 12 rules of test sequence. It requests the user to enter a set of test results for the two sides of the chosen chamber run and suggest the next run to be tested. Although the knowledge base is extremely small and the test results are in highly simplified forms, the performance of the system in selecting runs for testing is encouraging. An ability to justify run selections will be added to the system in the future.

Obtaining Test Results

The simulation of chamber runs will be carried out by the software package PC-PKSS developed by Jeffries.[20] The inputs to the PC-PKSS are the photochemical mechanism to be tested, as well as initial and time dependent conditions. The outputs are concentrations of species as a function of time. Both the outputs of PC-PKSS and the chamber data of the particular run are then imported to a Lotus spreadsheet which facilitates the plotting of the data from the two sources (mechanism prediction and chamber data) for comparison purposes.

In the first version of our future system, the user is expected to run the simulation on a separate personal computer and then answers the questions posed by the expert system after examining the profiles of the plots of chamber data versus mechanism predictions. In the long run, PC- PKSS is likely to be connected to ASKME and thus allows full automation of model evaluations.

Knowledge Base

The knowledge base of ASKME will be divided into two portions. One portion will contain the information of the preferred test sequence and explanations of each run type in the sequence. The preferred test sequence, which founds its basis on the hierarchy of HC species proposed by Whitten, will be constructed in consultation with the domain expert.

Another portion of the knowledge base is devoted to the rules and heuristics (rules of thumb) used by the domain expert in the process of mechanism evaluation. In the process of automatic mechanism testing, the inference engine built into the system will make use of the rules and the previous test results and form its "opinions" about the qualities of the mechanism.

Analyzing Test Results

Upon obtaining the results of the particular run, the system will carry out a series

of analyses based on the knowledge stored in the knowledge base and the results of all previous testings. The results of the particular test and analysis will be stored temporarily in the memory and become part of the test history. If this new piece of information makes any previous test result obsolete, the old information will be modified automatically.

Justification / Explanation

During each testing cycle, the users can ask the system to justify its chains of reasoning in the process of test results analysis or to explain why certain run is chosen for testing purposes. Different portion of the knowledge base will be referred to when either of the above two types of justification/explanation actions is being carried out. The justification of test results analysis will be given at different levels of details. The user can choose to look at one or more levels of justifications in each cycle of testing to have a good grip of how the system reveals the strengths and weaknesses of the mechanism being tested.

History File

As the system goes through each cycle of testing, it accumulates more and more information about the nature of the mechanism (In other words, it learns!) After each testing cycle, the user can choose to go on to next testing cycle or leave the interactive session. If the latter choice is made, the testing information accumulated so far will be saved into a history file. The history file is a regular text file which can be loaded to the system in next interactive session when the testing continues.

Implementation of ASKME

I have examined the configuration of the prospective expert system ASKME. In this section, I will discuss some implementation issues.

Implementing Inference Engine

Backward-chaining inference engine will be implemented in the system to carry out the desired reasoning tasks. One reason for this choice is that it is easier to implement backward-chaining inference engine in Turbo Prolog.[21] Another more important reason is that the hypothesis-driven method supported by this type of inference engine seems to work well with our system design in which the knowledge of a particular mechanism accumulates as the system goes through more and more testing cycles. However, this argument is subjected to justification; that is, we will built another prototype to verify this point.

Knowledge Base and Knowledge Acquisition

I am presently at the initial stage of building the knowledge base. As mentioned previously, the knowledge base will be divided into two portions: one for selecting runs and the other for analyzing test results. I have built a small knowledge base for run selections and attached it to a prototype (also mentioned earlier) to test the feasibility of automatic selection of runs by the computer. The results of the test suggested that not only automatic selection of runs is feasible, but it can also be very efficient if the knowledge base is carefully constructed.

Although the structure of the knowledge base for run selections developed so far seems to work well, it lacks the explanatory capabilities. I will modify the structure of this part of knowledge base in the future to accommodate explanatory capabilities.

The other portion of the knowledge base contains the knowledge for analyzing the test results. This portion of the knowledge base will be constructed initially with the help of the domain expert. Later, an intelligent editor, which can recognize the structures of the knowledge base and check the consistency of the rules in the knowledge base, will be built to help maintain and modify the knowledge base.

The knowledge base of ASKME will be kept in a text file in much the same way as the database of OSCECIS. In the design of ASKME, the knowledge base and the inference engine will be clearly separated. This separation is crucial in the sense that it facilitates the frequent modifications needed for the knowledge base without having to re-run or re-code the source program.

The effective interaction between the expert system programmer and the domain expert plays an important role in the initial stage of knowledge base construction, *i.e.*, knowledge acquisition. Three factors result in an advantage for me in this task. They are:

1. The expert system programmer happens to be the domain expert's student. The programmer actually "goes down the field" and learns the trade of model testing. This situation makes it especially conducive for the transformation of the expert's knowledge into a form understandable by the machine.
2. The domain expert happens to be very articulate. This quality helps the expert system programmer learn the new concepts and "catch" the expert's rules of thumb.
3. The domain expert is also a very knowledgeable computer scientist. This aspect will help reduce the information mismatch between the human and

the computer to a great extent. The task of knowledge acquisition can thus be made a little easier.

Closing Remarks

We have presented the outline of our prospective expert system and discussed some implementation issues. As we know, building an expert system can be very time consuming. The justification of building an expert thus rely partially on whether the time and efforts spent in developing it can be paid off.

I have pointed out in Chapter One that the development of a photochemical model is very painstaking and involves a great deal of complex reasoning and subjective judgement. However, some regularities can also be observed in this process, especially if the hierarchical approach is used. It is these regularities which lead us to believe that an automatic model evaluation can ultimately be achieved by a computer. The complexity of model development mandates the use of artificial intelligent techniques in implementing such a computer system. Taking into account the factors involved in the business of model development, I strongly believe that if an expert system devoted to automatic model evaluation can be successfully implemented, it will certainly speed up the process of model development tremendously.

References

1 Kowalski, R. (1979). *Logic for Problem Solving*, North Holland, NY.

2 Clocksin, W. F., & Mellish, C. S. (1981). *Programming in Prolog*, Springer-Verlag, NY.

3 Borland International, (1986). *Turbo Prolog*, Scotts Valley, CA.

4 Freas, W. P., Martinez, E. L., Meyer, E. L., Possiel, N. C., Sennet, D. H., & Summerhays, J. E., (1978). "Procedures for quantifying Relationships Between Photochemical Oxidants and Precursors: Supporting Documentation" EPA-450/2-77-021b, U.S.Environmental Protection Agency, Research Triangle Park, North Carolina.

5 Jeffries, H. E., Sexton, K. G. (1985). "Outdoor Smog Chamber Experiments To Test Photochemical Models: Phase II", Final Report (EPA/600/S3-85/029), Environmental Protection Agency, Research Triangle Park, N.C.; NTIS No. PB 85-191 542 / AS.

6 Whitten, G. T. (1983). "The Chemistry of Smog Formation: A Review of Current Knowledge", *Environmental International* 9, 447-463.

7 Jeffries, H. E., & Arnold, J. (1986). "The Science of Photochemical Reaction Mechanism Development and Evaluation", Department of Environmental Sciences and Engineering, University of North Carolina, Chapel Hill, NC.

8 Feigenbaum, E. A., Barr, A. (1982). *Handbook of Artificial Intelligence*, William Kaufmann Inc., CA.

9 Pua, K. E. (1984). "Translating Annotated Predicate Calculus to Prolog", ISG 84-1 File No. UIUCDCS-F-84-918, Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL.

10 Pua, K. E. (1984). "An Annotated Predicate Calculus Inference System (APCIS)", ISG 84-8 File No. UIUCDCS-F-84-928, Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL.

11 Barrett, W. A., Couch, J. D. (1979). *Compiler Construction: Theory and Practice*, Science Research Asso. Inc., Chicago, IL.

12 Waterman, D. A. (1985). *A Guide to Expert Systems*, Addison- Wesley, Reading, MA.

13 Jeffries, H. E, Kamens, R. M., Sexton, K. G. & Gerhardt, A. A. (1982). "Outdoor Smog Chamber Experiments to Test Photochemical Models", EPA-600/3-83-016a, U.S. Environmental Protection Agency, Research Triangle Park, NC.

14 Jeffries, H.E., Kamens, R.M., Sexton, K.G., & Gerhardt, A.A. (1982). "Outdoor Smog Chamber Experiments To Test Photochemical Models", EPA-600/3-83-016a, U.S. Environmental Protection Agency, Research Triangle Park, NC.

15 Lloyd, J. W. (1984). *Foundations of Logic Programming*, Springer-Verlag, Berlin.

16 Forsyth, R. ed. (1984). *Expert Systems: Principles and case Studies*, Chapman and Hall, London.

17 Nayer, C. (1983). *Build Your Own Expert System*, Sigma Technical Press / John Wiley, Chichester.

18 Michie, D. ed. (1982). it Introductory Readings in Expert Systems, Gordon and Breach, NY.

19 it Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. eds. (1983). it Building Expert Systems, Addison-Wesley, Reading, MA.

20 Jeffries, H. E. (1987). "User's Guide to Photochemical Kinetics Simulation System", Department of Environmental Sciences and Engineering, University of North Carolina, Chapel Hill, NC.

21 Schildt, H. (1987). Advanced Turbo Prolog, McGraw Hill, Berkeley, CA.

A

User's Guide to OSCECIS

OSCECIS was designed to provide easy access to the smog chamber runs conducted in the UNC Outdoor Smog Chamber for testings of photochemical kinetics models.

The following facilities are available in the system:

1. Accepting query through pop-up menus
2. Displaying output in various forms
3. Updating database
4. Diagnosing database
5. Viewing the database codes
6. Saving updated database
7. Access to DOS commands
8. Access to an editor

System Requirements

OSCECIS requires an IBM PC/XT/AT or compatible computer with at least one floppy disk drive and a minimum of 320 kilobytes of memory. A line printer is also required if the users choose to produce hard copies of the outputs.

Files Needed

The following files should be in the default directory in order to interact with the system successfully.

```
oscecis.exe  --  execution file
ec.tut       --  tutorial
ec.spe       --  species codes
ec.ser       --  series codes
ec.cod       --  codes for database fields
ec.db1       --  database (exper. conditions)
ec.db2       --  database (HC and NOx con.)
```

The follwowing file should be in the default directory if the user wants to invoke the help option from within the editor provided by the system:

```
prolog.hlp  --  help information
```

Getting Started

Type "oscecis" (without quotes) to the DOS prompt and hit *return*.

Important Notes

1. The database should be loaded before any querying could take place. Choose the "Load the database" panel in the main menu and hit the *return* key. It takes approximately 12 seconds to load the database on an IBM AT and approximately 50 seconds to load it on an IBM PC.
2. When you make a mistake, hit the *Esc* key as many times as necessary until the main menu appears again.
3. Choose the "Any" panel in any given menu whenever the the menu is irrelevant to your query.
4. Before choosing the "Line printer" panel in the "Output device" menu, make sure the printer is "on".
5. While in the "Update the database" mode, it is a good practice to check the existence of a record before inserting or deleting it.

Querying the Database

To query the database, choose the "Query the database" panel in the main menu. Four options are available under this mode.

1. Query by species. This option allows a quick way to query the database. The users are only asked to provide the species code(s) for the species of interest.
2. Query by series. This option is similar to the above option except that the users provide series code instead.
3. Query by date. By choosing the "Date(s)" panel in the "Query By Date(s)" menu,the users can enter a series of run dates of the form "yymmdd" and ask the system to display the details of the runs with proper choice of output forms (see next section). Alternatively, by choosing the "From date1 to date2" panel in the "Date(s)" menu, the users can query the database for all the runs bounded by the two dates, in chronological order, provided by the users. The users can also search for runs conducted in a given month of different years by choosing the "Month" panel in the menu.
4. Detailed query. This option constitutes the *core* of the query system. A series of menus corresponding to relevant fields in the database will be displayed. The "Any" panel should be chosen whenever a given menu is irrelevant to the query.

Selecting Output Devices and Forms

After the query has been executed sucessfully, the "Output device" menu will be displayed automatically. For each query, there are three choices of output device and for each output device there are eight ways to arrange the output information. The users are free to choose any combination of output devices and output forms. Multiple copies of outputs to any device are also allowed.

Before choosing the "Line printer" device, the printer should be set to "on".

Viewing the Database Codes

To view the eight types of codes in the database, choose the "View the database codes" panel in the main menu. The first two types of codes, species codes and series codes, are especially useful in the query. The species codes are arranged in increasing order of species names whereas the series codes are arranged in increasing order of the codes. Note that the leading zeros for those series codes smaller than "10" can not be omitted. Searching as well as sreen controlling facilities are also available for species and series codes.

Updating the Database

To update the database, choose the "Update the database" panel in the main menu. Four choices are available under this mode.

1. "Display record" will display the content of a record identified by the run date. When the record with the given run date is not found, a warning message will be given.
2. "Delete record" will delete from the database a record with the given run date. The absence of the to-be-deleted-record will not cause any warning message to be displayed.
3. "Insert record" will insert into the database a record with the given run date. If a record identified by the run date already existed in the database, a warning message would be given. Otherwise, the user will be asked to enter the content of the record through answering a series of questions. The record is then inserted into the database. If the usermakes a mistake in this process (e.g. enter an alphabetic character to where a numeric value is expected), the insertion will be aborted.
4. "Modify record" first checks if a record with the given run date is in the database. If it is not, a warning message will be issued. Otherwise, the user will be prompted by a series of pop-up menus corresponding to each field of the record of interest. Choose "Not to be modified" panel if you do not want the value of a corresponding field to be changed.

It is a good practice to check the existence of a record before inserting or deleting it.

Saving Database on File

To save the updated database on file, choose the "Save database on file" panel in the main menu. The updated database will be saved on the files *ec.db1* (experimental conditions) and *ec.db2* (HC and NOx concentrations) and the old copies of the database will be moved to the files *ecdb.db1* and *ecdb.db2* respectively.

Diagnosing the Database

To find out the possible error in the database, choose the "Diagnose the database" panel in the main menu. The results of the diagnoses can be either directed to the screen or the line printer. Those potnential "pathologic" records will be pointed out together with the suspected field(s) where the error(s) might have occurred.

Miscellaneous

This option allows the users to look at the two distributions of all species or all series types used in the database. The users can choose to send the outputs to either screen, line printer, or file. To look at the distribution of series types, choose the "Number of Runs Containing Series" panel in the "Miscellaneous" menu (appearing after the "Output device" menu). To look at the distribution of species types, choose the "Number of Chamber sides Containing Species" panel in the "Miscellaneous" menu.

Access to DOS Commands

Choosing the "DOS commands" panel in the main menu will allow the users to suspend OSCECIS temporarily and gain access to the DOS commands. To go back to OSCECIS, type EXIT to the DOS prompt.

Editor

The editor is pretty much WORDSTAR-like. It is accessible by choosing the "Editor" panel in the main menu. Besides text editing, the editor supports external text viewing and copying. The latter facility can be used to load any existing file into the editor for text processings.

A quick reference to the functions of the editor follows:

Cursor Movement Commands

Character left	LeftArrow or Ctrl-S
Character right	RightArrow or Ctrl-D
Word left	Ctrl-LeftArrow or Ctrl-A
Word right	Ctrl-RightArrow or Ctrl-F
Line up	UpArrow or Ctrl-E
Line down	DownArrow or Ctrl-X
Page up	PgUp or Ctrl-R
Page down	PgDn or Ctrl-C
Beginning of line	Home or Ctrl-Q-S
End of line	End or Ctrl-Q-D
Top of file	Ctrl-Home or Ctrl-Q-R
End of file	Ctrl-End or Ctrl-Q-C
Beginning of block	Crtl-Q-B
End of block	Crtl-Q-K

Insert and Delete Commands

Insert mode on/off	Ins or Ctrl-V
Delete left chracter	LeftArrow
Delete char under cursor	Del
Delete right word	Alt-RightArrow
Insert line	Ctrl-N
Delete line	Ctrl-Y
Delete to end of line	Ctrl-Q-Y

Block Commands

Mark block begin	Ctrl-K-B
Mark block end	Ctrl-K-K
Copy block	F5 or Ctrl-K-C
Repeat the last copy	Shift-F5
Move block	F6 or Ctrl-K-V
Delete block	F7 or Ctrl-K-Y
Read block from disk	F9 or Ctrl-K-R
Hide/display block	Ctrl-K-H

Miscellaneous Commands

Call the auxilliary editor	F8
Go to line	F2
Which line number	Shift-F2
End Edit	Esc or F10
Auto indent	Ctrl-Q-I
Find	F3 or Ctrl-Q-F
Repeat last find	Shift-F3 or Ctrl-L
Find and replace	F4 or Ctrl-Q-A
Repeat last find and repl	Shift-F4 or Ctrl-L

B

Hierarchy of Predicates

All the predicates defined in OSCECIS can be arranged in a hierarchy according to what predicates they called and by what predicates they are called. This appendix contains the hierarchy constructed according to this criterion. The hierarchy is presented in four blocks: main block, print block, menu block, and writeformat block.

Only defined predicates are entered into the structure; database and built-in predicates (except the *fail* predicate) are not shown. In the structure, those predicates which call themselves are recursive in nature. Likewise, a predicate which calls another predicate which calls back the first predicate is also recursive. The iterative predicates are those which call the defined predicate *repeat* or the built-in predicate *fail*.

Although the hierarchy does not cover every aspect of the source program listed in Appendix C, it holds the key to the understanding of the program.

```
                    ------------
                     MAIN BLOCK
                    ------------

mainmenu
    |  repeat
    |    | repeat
    |   [ MENU BLOCK ]
    |   proces(integer)
    |    | ed(string)
    |    | view
    |    |    | repeat
    |    |    |    | repeat
    |    |    |  [ MENU BLOCK ]
    |    |    |  viewl(integer)
    |    |    |    | viewhc
    |    |    |    |    | fail
    |    |    |    | viewclass
    |    |    |    |    | fail
    |    |    |    | viewdilute
    |    |    |    |    | fail
    |    |    |    | viewinject
    |    |    |    |    | fail
    |    |    |    | viewproj
    |    |    |    |    | fail
    |    |    |    | viewstatus
    |    |    |    |    | fail
    |    | diagnose
    |    |    | repeat
    |    |    |    | repeat
    |    |    |  [ MENU BLOCK ]
    |    |    |  diagnosel(integer)
    |    |    |    | diag
    |    |    |    |    | diag_ec(string,string,string,lists)
    |    |    |    |    |    | member(string,lists)
    |    |    |    |    |    |    | member(string,lists)
    |    |    |    |    | diag_rb(string,string,string,listi)
    |    |    |    |    |    | memberi(integer,listi)
    |    |    |    |    |    |    | memberi(integer,listi)
    |    |    |    |    | fail
    |    | updatedb
    |    |    | repeat
    |    |    |    | repeat
    |    |    |  [ MENU BLOCK ]
    |    |    |  updatel(integer)
    |    |    |    | displays(string)
    |    |    |    | inserts(string)
    |    |    |    |    | i_conc(string,string)
    |    |    |    |    |    | drawbox(string)
    |    |    |    |    |    | removebox
    |    |    |    |    | i_hcID(string,string)
    |    |    |    |    |    | drawbox(string)
```

```
|   |   |   |   |   |   | removebox
|   |   |   |   |   | i_other(string,string)
|   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   | removebox
|   |   |   |   |   | classes(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | dilutes(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | injects(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | hctype(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | projcode(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | pcstatus(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | i_series
|   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   | removebox
|   |   |   |   | modify(string)
|   |   |   |   |   | m_conc(string,string)
|   |   |   |   |   |   | mmenu(string,integer)
|   |   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   | removebox
|   |   |   |   |   | m_hcID(string,string)
|   |   |   |   |   |   | mmenu(string,integer)
|   |   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   | removebox
|   |   |   |   |   | m_other(string,string)
|   |   |   |   |   |   | mmenu(string,integer)
|   |   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   | removebox
|   |   |   |   |   | classes(string)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   | dilutes(string)
```

```
|   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   | injects(string)
|   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   | hctype(string)
|   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   | projcode(string)
|   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   | pcstatus(string)
|   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | index(lists,integer,string)
|   |   |   |   |   |   | index(lists,integer,string)
|   |   |   |   | m_series
|   |   |   |   |   | mmenu(string,integer)
|   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   | drawbox(string)
|   |   |   |   |   | removebox
|   | clear_facts
|   | query
|   |   | query1(integer)
|   |   |   | qmenu(integer)
|   |   |   |   | [ MENU BLOCK ]
|   |   |   | qSpeci(integer)
|   |   |   |   | specID(string)
|   |   |   |   | querySpeci(integer)
|   |   |   |   |   | specifind(integer)
|   |   |   |   |   |   | checkRB(rellist,string,....)
|   |   |   |   |   |   |   | rel_r(rellist,string,real)
|   |   |   |   |   |   |   |   | rel_r(rellist,string,real)
|   |   |   |   |   |   | fail
|   |   |   |   |   | [ PRINT BLOCK ]
|   |   |   | qDtail(integer)
|   |   |   |   | specID(string)
|   |   |   |   | queryDetail(integer)
|   |   |   |   |   | con(string,string)
|   |   |   |   |   |   | repeat
|   |   |   |   |   |   |   | repeat
|   |   |   |   |   |   | relmenu(string,integer)
|   |   |   |   |   |   |   | [ MENU BLOCK ]
|   |   |   |   |   |   |   | drawbox(string)
|   |   |   |   |   |   |   |   | relreal(string,integer)
|   |   |   |   |   |   |   | removebox
|   |   |   |   |   | others(string,string)
|   |   |   |   |   |   | repeat
|   |   |   |   |   |   |   | repeat
```

```
| | | | | | | | relmenu(string,integer)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | drawbox(string)
| | | | | | | |   |   | relint(string,integer)
| | | | | | | |   | removebox
| | | | | | | | classes(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | dilutes(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | injects(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | hctype(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | projcode(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | pcstatus(string)
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | index(lists,integer,string)
| | | | | | | |   |   | index(lists,integer,string)
| | | | | | | | series
| | | | | | | |   | [ MENU BLOCK ]
| | | | | | | |   | series1(integer)
| | | | | | | |   |   | drawbox(string)
| | | | | | | |   |   | removebox
| | | | | | | | detailfind(integer)
| | | | | | | |   | rel_r(rellist,string,real)
| | | | | | | |   |   | rel_r(rellist,string,real)
| | | | | | | |   | rel_i(rellist,string,integer)
| | | | | | | |   |   | rel_i(rellist,string,integer)
| | | | | | | |   | checkRB(rellist,string,...)
| | | | | | | |   |   | rel_r(rellist,string,real)
| | | | | | | |   |   |   | rel_r(rellist,string,real)
| | | | | | | |   | checkEC(cmplist,string,...)
| | | | | | | |   |   | checkEC(cmplist,string,...)
| | | | | | | |   | fail
| | | | | | |   | [ PRINT BLOCK ]
| | | | | querySeries
| | | | |   | series
| | | | |   |   | [ MENU BLOCK ]
| | | | |   |   | series1(integer)
| | | | |   |   |   | drawbox(string)
| | | | |   |   |   | removebox
```

```
|   |   |   |   | seriesfind
|   |   |   |   |   | checkEC(cmplist,string,...)
|   |   |   |   |   |   | checkEC(cmplist,string,...)
|   |   |   |   |   | fail
|   |   |   |   | [ PRINT BLOCK ]
|   |   |   | queryDate
|   |   |   |   | [ MENU BLOCK ]
|   |   |   |   | qdate(integer)
|   |   |   |   |   | clear_ans
|   |   |   |   |   | readdate(string)
|   |   |   |   |   |   | readdate(string)
|   |   |   |   |   | drawbox(string)
|   |   |   |   |   | removebox
|   |   |   |   |   | monthfind(string)
|   |   |   |   |   | fail
|   |   |   |   |   | datefind(string,string)
|   |   |   |   |   |   | fail
|   |   |   |   | [ PRINT BLOCK ]
|   | miscellaneous
|   |   | repeat
|   |   |   | repeat
|   |   | [ MENU BLOCK ]
|   |   | miscel1(integer)
|   |   |   | misc
|   |   |   |   | repeat
|   |   |   |   |   | repeat
|   |   |   |   | [ MENU BLOCK ]
|   |   |   |   | miscl(integer)
|   |   |   |   |   | headser
|   |   |   |   |   | seriesCount
|   |   |   |   |   |   | ser(string,string)
|   |   |   |   |   |   |   | countSer(integer)
|   |   |   |   |   |   |   |   | counts(integer,...)
|   |   |   |   |   |   |   |   | fail
|   |   |   |   |   |   |   | writeSer(string)
|   |   |   |   |   |   | fail
|   |   |   |   |   | headspe
|   |   |   |   |   | speciesCount
|   |   |   |   |   |   | sides(integer,string)
|   |   |   |   |   |   |   | countSide(integer)
|   |   |   |   |   |   |   |   | countUp(integer,...)
|   |   |   |   |   |   |   |   |   | ckSide(integer,...)
|   |   |   |   |   |   |   |   | writeSide(string)
|   |   |   |   |   |   |   |   | fail
|   |   |   |   |   | fail
|   | savedb1
|   | savedb2
```

```
                    ------------
                     PRINT BLOCK
                    ------------

print
   | repeat
   |    | repeat
   | [ MENU BLOCK ]
   | print0(integer)
   |    | clear_ans
   |    |    | fail
   |    | pmenu
   |    |    | repeat
   |    |    |    | repeat
   |    |    | [ MENU BLOCK ]
   |    |    | pauses(integer)
   |    |    | printl(integer)
   |    |    |    | printSimple
   |    |    |    |    | split(string,string,lists,lists)
   |    |    |    |    |    | split(string,string,lists,lists)
   |    |    |    |    | writesol(lists)
   |    |    |    |    |    | writesol(lists)
   |    |    |    |    | listlen(lists,integer)
   |    |    |    |    |    | listlen(lists,integer)
   |    |    |    | printByClass
   |    |    |    |    | unik(lists,lists)
   |    |    |    |    |    | unik(lists,lists)
   |    |    |    |    | heading
   |    |    |    |    | printClass(lists)
   |    |    |    |    |    | check_class(string,string)
   |    |    |    |    |    | [ WRITEFORMAT BLOCK ]
   |    |    |    |    |    | listlen(lists,integer)
   |    |    |    |    |    |    | listlen(lists,integer)
   |    |    |    |    |    | printClass(lists)
   |    |    |    | printByDilution
   |    |    |    |    | unik(lists,lists)
   |    |    |    |    |    | unik(lists,lists)
   |    |    |    |    | heading
   |    |    |    |    | printDilute(lists)
   |    |    |    |    |    | check_dilute(string,string)
   |    |    |    |    |    | [ WRITEFORMAT BLOCK ]
   |    |    |    |    |    | listlen(lists,integer)
   |    |    |    |    |    |    | listlen(lists,integer)
   |    |    |    |    |    | printDilute(lists)
   |    |    |    | printByInjection
   |    |    |    |    | unik(lists,lists)
   |    |    |    |    |    | unik(lists,lists)
   |    |    |    |    | heading
   |    |    |    |    | printInject(lists)
   |    |    |    |    |    | check_inject(string,string)
   |    |    |    |    |    | [ WRITEFORMAT BLOCK ]
   |    |    |    |    |    | listlen(lists,integer)
```

```
|  |  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  | printInject(lists)
|  |  |  |  | printByHCtype
|  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  | heading
|  |  |  |  |  | printHC(lists)
|  |  |  |  |  |  | check_hc(string,string)
|  |  |  |  |  |  | [ WRITEFORMAT BLOCK ]
|  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  | printHC(lists)
|  |  |  |  | printByProj
|  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  | heading
|  |  |  |  |  | printProj(lists)
|  |  |  |  |  |  | check_proj(string,string)
|  |  |  |  |  |  | [ WRITEFORMAT BLOCK ]
|  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  | printProj(lists)
|  |  |  |  | printByStatus
|  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  | heading
|  |  |  |  |  | printStatus(lists)
|  |  |  |  |  |  | check_status(string,string)
|  |  |  |  |  |  | [ WRITEFORMAT BLOCK ]
|  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  | printStatus(lists)
|  |  |  |  | printBySeries
|  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  |  | unik(lists,lists)
|  |  |  |  |  | heading
|  |  |  |  |  | printSeries(lists)
|  |  |  |  |  |  | check_ser(string,string)
|  |  |  |  |  |  | [ WRITEFORMAT BLOCK ]
|  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  |  | listlen(lists,integer)
|  |  |  |  |  |  | printSeries(lists)
```

```
            ------------
             MENU BLOCK
            ------------

menu(row,col,string,lists,integer)
   | maxlen(lists,integer,integer)
   |   | maxlen(lists,integer,integer)
   | listlen(lists,integer)
   |   | listlen(lists,integer)
   | writelist(integer,integer,lists)
   |   | writelist(integer,integer,lists)
   | menu1(row,lists,row,integer,integer)
   |   | readkey(key)
   |   |   | readkey1(key,char,integer)
   |   |   |   | readkey2(key,integer)
   |   |   | menu2(row,lists,row,integer,integer,key)
   |   |   |   | menu1(row,lists,row,integer,integer)


            -------------------
             WRITEFORMAT BLOCK
            -------------------

writeformat(string,string)
   | ec_field(string,...,integer)
   | code_date(string,string)
   | writerb(string,real,real,integer)
   | writehc(real,integer)
   | fail
```

C

Program Listings

The source program of OSCECIS was saved in four text files. The first file, *ecdef27.pro*, contains the global definitions and database declarations. The second file, *ec27a.pro*, contains help predicates, predicates for creating menus, top level of the control portion of the program, predicates for viewing the database codes, and miscellaneous facilities. The third file, *ec27b.pro*, contains predicates for reading the queries and for searching mechanisms. The last file, *ec27c.pro*, contains predicates for printing solutions and updating database.

```
/*--------------------------------*/
/*  OSCECIS          ECDEF27.PRO  */
/*  Version 1        March 8, 87  */
/*  Copyright (C)    Kah-Eng Pua  */
/*--------------------------------*/
GLOBAL DOMAINS
  ROW,COL,LEN - INTEGER
  FILE     - OUTF
  LISTS    - STRING*
  LISTI    - INTEGER*
  EXPCOND  - class(STRING);  dilute(STRING); inject(STRING);
             hctype(STRING); proj(STRING);   proces(STRING);
             series(STRING)
  TYPE     - r(REAL); i(INTEGER)
  RELATION - eq(STRING,TYPE); lt(STRING,TYPE); gt(STRING,TYPE);
             range(STRING,TYPE,TYPE)
  IDTYPE   - specID(STRING,INTEGER)
  CMPLIST  - CMPTERM*
  RELLIST  - RELATION*
  SOLUTION - ec(STRING,STRING,STRING,STRING,STRING,STRING,STRING,STRING,
                INTEGER)

DATABASE
  code_month(STRING,STRING)
  code_class(STRING,STRING)
  code_hc(STRING,STRING)
  code_dilute(STRING,STRING)
  code_inject(STRING,STRING)
  code_proj(STRING,STRING)
  code_status(STRING,STRING)
  code_series(STRING,STRING)
  code_spec(INTEGER,STRING)

  /* ec(Date,Class,Dilution,Injection,HCtype,Proj,PcStat,Series,Rank)  */
     ec(STRING,STRING,STRING,STRING,STRING,STRING,STRING,STRING,INTEGER)
  /* rb(Date,redNOx,HCcon1,HCcode1,HCcon2,HCcode2,HCcon3,HCcode3,      */
     rb(STRING,REAL,REAL,INTEGER,REAL,INTEGER,REAL,INTEGER,
  /*           blueNOx,HCcon1,HCcode1,HCcon2,HCcode2,HCcon3,HCcode3,   */
               REAL,REAL,INTEGER,REAL,INTEGER,REAL,INTEGER,
  /*           HCinstrument,QualityOfRun,SunLight)                     */
        INTEGER,INTEGER,INTEGER)

  w(IDTYPE)
  x(EXPCOND)
  y(RELATION)
  z(SOLUTION)
  counter_i(INTEGER,INTEGER)
  counter_s(INTEGER,STRING)
  dbload
```

```
GLOBAL PREDICATES
  print
  query
  updatedb
  clear_facts
  clear_ans
  repeat
  yes(CHAR)-(i)
  pauses(INTEGER)-(i)
  menu(ROW,COL,STRING,LISTS,INTEGER)-(i,i,i,i,o)
  unik(LISTS,LISTS)-(i,o) /* Eliminate duplicates in a string list */
  index(LISTS,INTEGER,STRING)-(i,i,o) /* Select an element from a list */
  listlen(LISTS,INTEGER)-(i,o)        /* Find the length of a list */
  member(STRING,LISTS)-(i,i)          /* Membership of a list */
  classes(STRING)-(i)
  dilutes(STRING)-(i)
  injects(STRING)-(i)
  hctype(STRING)-(i)
  projcode(STRING)-(i)
  pcstatus(STRING)-(i)
  drawbox(STRING)-(i)
  removebox
  diagnose
```

```
/*---------------------------------*/
/*  OSCECIS          EC27A.PRO     */
/*  Version 1        March 8, 87   */
/*  Copyright (C)    Kah-Eng Pua   */
/*---------------------------------*/
CODE-1500
PROJECT "ECPROJ27.PRJ"
INCLUDE "ECDEF27.PRO"

/*---------------------------------*/
/*      HELP PREDICATES            */
/*---------------------------------*/
PREDICATES
  maxlen(LISTS,INTEGER,INTEGER)/* Find the length of the longest string */
  writelist(INTEGER,INTEGER,LISTS)/* Used by the menu predicate */
  ed(STRING)          /* Save edited text in file */
  savedb1             /* Save database on file */
  savedb2             /* Save database on file */

CLAUSES
  index([X|_],1,X):- !.
  index([_|L],N,X):- N>1,N1=N-1,index(L,N1,X).

  unik([],[]).
  unik([H|T],L):- member(H,T),!,unik(T,L).
  unik([H|T],[H|L]):-unik(T,L).

  maxlen([H|T],MAX,MAX1):-
    str_len(H,LEN),
    LEN>MAX,!,
    maxlen(T,LEN,MAX1).
  maxlen([_|T],MAX,MAX1):-maxlen(T,MAX,MAX1).
  maxlen([],LEN,LEN).

  listlen([],0).
  listlen([_|T],N):-
    listlen(T,X),
    N=X+1.

  writelist(_,_,[]).
  writelist(LI,ANTKOL,[H|T]):-field_str(LI,0,ANTKOL,H),
    LI1=LI+1,writelist(LI1,ANTKOL,T).

  repeat. repeat:-repeat.

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

  clear_facts :- retract(w(_)), fail.
  clear_facts :- retract(x(_)), fail.
  clear_facts :- retract(y(_)), fail.
  clear_facts.
```

```
clear_ans :- retract(z(_)),fail.
clear_ans.

drawbox(Heading) :-
  shiftwindow(20),
  makewindow(3,7,7,Heading,14,30,4,46).

removebox :-
  removewindow,
  shiftwindow(22),
  shiftwindow(2).

yes('y'). yes('Y').

pauses(0) :- !.
pauses(_) :- readchar(_).

ed("") :- !.
ed(L) :-
  write("Save old text? (y/n) "),readchar(Yn),yes(Yn),!,
  write("Name of file : "),readln(N),
  openwrite(outf,N),writedevice(outf),write(L),closefile(outf).
ed(_).

savedb1:-
  ec(Date,A,B,C,D,E,F,Se,Rk),
  write("ec(",Date,",",A,",",B,",",C,",",
            D,",",E,",",F,",",Se,",",Rk,")"),fail.
savedb1.

savedb2:-
  rb(X,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  write("rb(",X,","),
  writef("%5.3,%4.2,%3,%4.2,%3,%4.2,%3,%5.3,%4.2,%3,%4.2,%3,%4.2,%3,
          %1,%1,%3",  G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  write(")"),fail.
savedb2.

/*-----------------------------------*/
/*       READING THE KEYBORD         */
/*-----------------------------------*/
DOMAINS
  KEY = cr ; esc ; break ; tab ; btab ; del ; bdel ; ins ;
        end ; home ; ftast(INTEGER) ; up ; down ; left ; right ;
        ctrlleft; ctrlright; ctrlend; ctrlhome; pgup; pgdn;
        chr(CHAR) ; otherspec

PREDICATES
  readkey(KEY)
  readkey1(KEY,CHAR,INTEGER)
  readkey2(KEY,INTEGER)
```

```
CLAUSES
  readkey(KEY):-readchar(T),char_int(T,VAL),readkey1(KEY,T,VAL).

  readkey1(KEY,_,0):-!,readchar(T),char_int(T,VAL),readkey2(KEY,VAL).
  readkey1(cr,_,13):-!.
  readkey1(esc,_,27):-!.
  readkey1(chr(T),T,_) .

  readkey2(up,72):-!.
  readkey2(down,80):-!.
  readkey2(ftast(N),VAL):-VAL>58,VAL<70,N=VAL-58,!.
  readkey2(otherspec,_).


/*-----------------------------------------*/
/*                 MENU                    */
/* Implements a popup-menu                 */
/* menu(Line,Collum,ListOfChoices,ChoiceNr) */
/* The following keys can be used:         */
/*    arrows up and down: select choice    */
/*    cr and F10: activate choice          */
/*    Esc: abort                           */
/*-----------------------------------------*/

PREDICATES
  menu1(ROW,LISTS,ROW,INTEGER,INTEGER)
  menu2(ROW,LISTS,ROW,INTEGER,INTEGER,KEY)

CLAUSES
  menu(LI,KOL,TXT,LIST,CHOICE):-
    shiftwindow(21),
    maxlen(LIST,0,ANTKOL),
    listlen(LIST,LEN),ANTLI=LEN,LEN>0,
    HH1=ANTLI+2,HH2=ANTKOL+2,
    makewindow(3,7,7,TXT,LI,KOL,HH1,HH2),
    HH3=ANTKOL,
    writelist(0,HH3,LIST),cursor(0,0),
    menu1(0,LIST,ANTLI,ANTKOL,CH),
    CHOICE=1+CH,
    removewindow,
    shiftwindow(22),
    shiftwindow(2).

  menu1(LI,LIST,MAXLI,ANTKOL,CHOICE):-
    field_attr(LI,0,ANTKOL,112),
    cursor(LI,0),
    readkey(KEY),
    menu2(LI,LIST,MAXLI,ANTKOL,CHOICE,KEY).
```

```
  menu2(_,_,_,_,-1,esc):-!.
  menu2(LI,_,_,_,CH,ftast(10)):-!,CH=LI.
  menu2(LI,_,_,_,CH,cr):-!,CH=LI.
  menu2(LI,LIST,MAXLI,ANTKOL,CHOICE,up):-
    LI>0,!,
    field_attr(LI,0,ANTKOL,7),
    LI1=LI-1,
    menu1(LI1,LIST,MAXLI,ANTKOL,CHOICE).

  menu2(LI,LIST,MAXLI,ANTKOL,CHOICE,down):-
    LI<MAXLI-1,!,
    field_attr(LI,0,ANTKOL,7),
    LI1=LI+1,
    menu1(LI1,LIST,MAXLI,ANTKOL,CHOICE).

  menu2(LI,LIST,MAXLI,ANTKOL,CHOICE,_):-
    menu1(LI,LIST,MAXLI,ANTKOL,CHOICE).


/*-----------------------------------*/
/*            MAIN MENU              */
/*-----------------------------------*/
PREDICATES
  mainmenu
  proces(INTEGER)
  view
  miscellaneous

GOAL
  makewindow(2,7,7,"",0,0,24,80),
  cursor(4,28),
  write("<< O S C E C I S >>"),
  cursor(6,6),
  write("Outdoor Smog Chamber Experimental Conditions Information System"),
  cursor(12,32),
  write("Kah-Eng Pua"),
  cursor(15,28),
  write("Copyright (C) 1986"),
  write("        Department of Environmental Sciences and Engineering"),
  write("                        School of Public Health"),
  write("                 University of North Carolina at Chapel Hill"),
  makewindow(19,112,0,"",24,0,1,80),
  write("                                        Press Any Key"),
  readchar(_),
  removewindow,removewindow,

  makewindow(19,112,0,"",24,0,1,80),
  write("  Ctrl S: Stop output    Any key to display menu"),
  makewindow(20,112,0,"",24,0,1,80),
  write("  Enter desired number or character(s) after the prompt."),
  makewindow(21,112,0,"",24,0,1,80),
```

```
  write("Esc: Quit this menu    Use arrow keys to select and "),
  write("hit RETURN to activate."),
  makewindow(22,112,0,"",24,0,1,80),
  write("Esc: Quit     F8: Last line    Ctrl S: Stop output   "),
  write(" End: End of line"),
  makewindow(2,7,7," O S C E C I S ",0,0,24,80),
  mainmenu,
  removewindow,removewindow,removewindow,removewindow,removewindow.

CLAUSES
  mainmenu:-
    repeat,
    clearwindow,
    menu(7,49,"Main Menu",
      [ "Tutorial",
        "Dos commands",
        "Editor",
        "",
        "Load the database",
        "Save database on file",
        "",
        "View the database codes",
        "",
        "Diagnose the database",
        "Update the database",
        "Query the database",
        "",
        "Miscellaneous"
      ],CHOICE),
    proces(CHOICE),
    CHOICE=0,!.

  proces(0):-write("Are you sure you want to quit? (y/n) "),
             readchar(T),yes(T).
  proces(1):-file_str("ec.tut",TXT),display(TXT),clearwindow,!.
  proces(1):-write(">> ec.tut not in default directory. Press any key."),
             readchar(_).
  proces(2):-makewindow(3,7,0,"",0,0,25,80),
             write("Type EXIT to return"),system(""),!,removewindow.
  proces(2):-write(">> command.com not accessible. Press any key"),
             readchar(_),removewindow.
  proces(3):-makewindow(3,7,112,"",5,2,19,75),edit("",L),
             removewindow,ed(L).
  proces(4).
  proces(5):-dbload,!,write("The database hase been loaded!"),beep.
  proces(5):-write("Loading database... please wait"),
             consult("ec.cod"),consult("ec.db1"),consult("ec.db2"),
             assert(dbload),beep,!.
  proces(5):-write(">> Databases missing or in error. Press any key."),
             readchar(_).
```

```
  proces(6):-deletefile("ecdb.db1"),renamefile("ec.db1","ecdb.db1"),
             deletefile("ec.db1"),
             write("Saving database... please wait"),
             openwrite(outf,"ec.db1"),writedevice(outf),savedb1,
             closefile(outf),
             deletefile("ecdb.db2"),renamefile("ec.db2","ecdb.db2"),
             deletefile("ec.db2"),
             openwrite(outf,"ec.db2"),writedevice(outf),savedb2,
             closefile(outf).
  proces(7).
  proces(8):-view.
  proces(9).
  proces(10):- diagnose.
  proces(11):- updatedb.
  proces(12):- clear_facts,query.
  proces(13).
  proces(14):- miscellaneous.

/*----------------------*/
/*    VIEWING CODES     */
/*----------------------*/
PREDICATES
  viewl(INTEGER)
  viewhc
  viewclass
  viewdilute
  viewinject
  viewproj
  viewstatus

CLAUSES
  view :-
    repeat,
    menu(14,55,"Viewing Codes",
      [ "Species Codes",
        "Series Codes",
        "HC Type Codes",
        "Class Codes",
        "Dilution Codes",
        "Injection Codes",
        "Project Codes",
        "Processing Status Codes"
      ],CHOICE),
    clearwindow,
    viewl(CHOICE),
    CHOICE=0,!.

  viewl(0).
  viewl(1):-file_str("ec.spe",TXT),display(TXT),!.
  viewl(1):-write(">> ec.spe not in default directory. press any key."),
            readchar(_).
```

```
  viewl(2):-file_str("ec.ser",TXT),display(TXT),!.
  viewl(2):-write(">> ec.ser not in default directory. press any key."),
            readchar(_).
  viewl(3):-write(" Code    HC Type"),nl,viewhc.
  viewl(4):-write(" Code    Class Type"),nl,viewclass.
  viewl(5):-write(" Code    Dilution Type"),nl,viewdilute.
  viewl(6):-write(" Code    Injection Type"),nl,viewinject.
  viewl(7):-write(" Code    Project Type"),nl,viewproj.
  viewl(8):-write(" Code    Processing Status"),nl,viewstatus.

  viewhc:-
    code_hc(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewhc.

  viewclass:-
    code_class(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewclass.

  viewdilute:-
    code_dilute(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewdilute.

  viewinject:-
    code_inject(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewinject.

  viewproj:-
    code_proj(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewproj.

  viewstatus:-
    code_status(Cod,Spe),
    write("  ",Cod,"    ",Spe),nl,fail.
  viewstatus.

/*-----------------------*/
/*  DIAGNOSING DATABASE  */
/*-----------------------*/
PREDICATES
  diag
  diagnosel(INTEGER)
  diag_ec(STRING,STRING,STRING,LISTS)
  diag_rb(STRING,STRING,INTEGER,LISTI)
  memberi(INTEGER,LISTI)
```

```
CLAUSES
  diagnose:-
    repeat,
    menu(20,63,"Output device",
      [ " Screen",
        " Line Printer  "
      ],CHOICE),
    diagnosel(CHOICE),
    CHOICE=0,!.

  diagnosel(0).
  diagnosel(1) :- diag.
  diagnosel(2) :- writedevice(printer),diag,writedevice(screen).

  diag:-
    write("Diagnosing database..."),
    findall(Cla,code_class(Cla,_),Lclas),
    findall(Dil,code_dilute(Dil,_),Ldilu),
    findall(Inj,code_inject(Inj,_),Linj),
    findall(HCt,code_hc(HCt,_),Lhc),
    findall(Pro,code_proj(Pro,_),Lproj),
    findall(Sta,code_status(Sta,_),Lsta),
    findall(Ser,code_series(Ser,_),Lser),
    findall(Spe,code_spec(Spe,_),Lspe),
    ec(Date,A,B,C,D,E,F,W,_),
    diag_ec(Date," 2nd field - Class   : ",A,Lclas),
    diag_ec(Date," 3rd field - Dilute  : ",B,Ldilu),
    diag_ec(Date," 4th field - Inject  : ",C,Linj),
    diag_ec(Date," 5th field - HCtype  : ",D,Lhc),
    diag_ec(Date," 6th field - Project : ",E,Lproj),
    diag_ec(Date," 7th field - Status  : ",F,Lsta),
    diag_ec(Date," 8th field - Series  : ",W,Lser),
    rb(Date,_,_,I,_,K,_,M,_,_,P,_,R,_,T,_,_,_),
    diag_rb(Date," 4th",I,Lspe),
    diag_rb(Date," 6th",K,Lspe),
    diag_rb(Date," 8th",M,Lspe),
    diag_rb(Date,"11th",P,Lspe),
    diag_rb(Date,"13th",R,Lspe),
    diag_rb(Date,"15th",T,Lspe),fail.
  diag:-write("Done.").

  diag_ec(_,_,Code,List):-
    member(Code,List),!.
  diag_ec(Date,Field,Code,_):-
    write("Run ",Date," * ec * ",Field,Code),nl.

  diag_rb(_,_,Code,List):-
    memberi(Code,List),!.
  diag_rb(Date,Field,Code,_):-
    write("Run ",Date," * rb * ",Field," field - Species : ",Code),nl.

  memberi(X,[X|_]).
  memberi(X,[_|T]) :- memberi(X,T).
```

```
/*----------------------*/
/*    MISCELLANEOUS     */
/*----------------------*/
PREDICATES
  miscell(INTEGER)
  misc
  miscl(INTEGER)

  headspe
  speciesCount
  sides(INTEGER,STRING)
  countSide(INTEGER)
  countUp(INTEGER,INTEGER,INTEGER,INTEGER)
  ckSide(INTEGER,INTEGER,INTEGER,INTEGER)
  writeSer(STRING)

  headser
  seriesCount
  ser(STRING,STRING)
  countSer(STRING)
  counts
  writeSide(STRING)

CLAUSES
  miscellaneous:-
    repeat,
    menu(19,63,"Output device",
      [ " Screen",
        " Line Printer  ",
        " File"
      ],CHOICE),
    miscell(CHOICE),
    CHOICE=0,!.

  miscell(0).
  miscell(1) :- misc.
  miscell(2) :- writedevice(printer),misc,writedevice(screen).
  miscell(3) :- write("Name of your file: "),readln(Name),
    openwrite(outf,Name),writedevice(outf),misc,closefile(outf).

  misc:-
    repeat,
    menu(20,36," Miscellaneous ",
      [ "Number of Runs Containing Series",
        "Number of Chamber Sides Containing Species"
      ],CHOICE),
      shiftwindow(19),
      shiftwindow(2),
      clearwindow,
      miscl(CHOICE),
      pauses(CHOICE),
      CHOICE=0,!.
```

```
miscl(0).
miscl(1):- headser,seriesCount.
miscl(2):- headspe,speciesCount.

headser:-
  write("Number of Runs Containing Series"),
  write("  Runs    Code    Series  ----    ----    -------").

seriesCount:-
  code_series(Code,Ser),
  ser(Code,Ser),fail.
seriesCount.

ser("00",_):-!.
ser(Code,Spe):-
  assert(counter_s(0,Code)),
  countSer(Code),
  writeSer(Spe),!.

countSer(Code):-
  ec(_,_,_,_,_,_,_,Code,_),
  counts,fail.
countSer(_).

counts:-
  retract(counter_s(Count,Code)),
  Count1=Count+1,
  assert(counter_s(Count1,Code)),!.
counts.

writeSer(_):-
  retract(counter_s(0,_)),!.
writeSer(S):-
  retract(counter_s(Count,Code)),
  writef("   %3       %2      ",Count,Code),
  write(S),nl.

headspe:-
  write("Number of Chamber Sides Containing Species"),
  write("  Sides    Code    Species  -----    ----    -------").

speciesCount:-
  code_spec(Code,Spe),
  sides(Code,Spe),fail.
speciesCount.
```

```
sides(0,_):-!.
sides(Code,Spe):-
  assert(counter_i(0,Code)),
  countSide(Code),
  writeSide(Spe),!.

countSide(Code):-
  rb(_,_,_,I,_,K,_,M,_,_,P,_,R,_,T,_,_,_),
  countUp(Code,I,K,M),
  countUp(Code,P,R,T),fail.
countSide(_).

countUp(Code,A,B,C):-
  ckSide(Code,A,B,C),
  retract(counter_i(Count,Code)),
  Count1=Count+1,
  assert(counter_i(Count1,Code)),!.
countUp(_,_,_,_).

ckSide(Code,A,B,C):-
  Code=A or Code=B or Code=C.

writeSide(_):-
  retract(counter_i(0,_)),!.
writeSide(S):-
  retract(counter_i(Count,Code)),
  writef("   %3       %3      ",Count,Code),
  write(S),nl.
```

```
/*--------------------------------*/
/*  OSCECIS          EC27B.PRO    */
/*  Version  1       March 8, 87  */
/*  Copyright (C)    Kah-Eng Pua  */
/*--------------------------------*/
CODE = 1200
PROJECT "ECPROJ27.PRJ"
INCLUDE "ECDEF27.PRO"
/*-------------------------*/
/*    READING QUERIES      */
/*-------------------------*/
PREDICATES
  relmenu(STRING,INTEGER)
  qmenu(INTEGER)

CLAUSES
  relmenu(Title,CHOICE) :-
    menu(8,35,Title,
      [ " Any                              ",
        " Equals",
        " Less than",
        " Greater than",
        " Range"
       ],CHOICE).

  qmenu(CHOICE) :-
    menu(9,45,"# Species",
      [ "Any",
        "One Species / Mixture",
        "Two Species / Mixtures",
        "Three Species / Mixtures"
      ],CHOICE).

PREDICATES
  query1(INTEGER)
  readdate(STRING)
  qSpeci(INTEGER)
  qDtail(INTEGER)
  querySpeci(INTEGER)
  querySeries
  queryDate
  queryDetail(INTEGER)
  qdate(INTEGER)
  specID(STRING)
  con(STRING,STRING)
  others(STRING,STRING)
  series
  series1(INTEGER)

  relreal(STRING,INTEGER)
  relint(STRING,INTEGER)
```

```
  specifind(INTEGER)
  detailfind(INTEGER)
  seriesfind
  datefind(STRING,STRING)
  monthfind(STRING)

CLAUSES
  query :-
    menu(9,50,"Query Type",
      [ "Query By Species",
        "Query By Series",
        "Query By Dates",
        "Detailed Query"
      ],CHOICE),
    query1(CHOICE).

  query1(0).
  query1(1) :- qmenu(CHOICE),qSpeci(CHOICE).
  query1(2) :- querySeries.
  query1(3) :- queryDate.
  query1(4) :- qmenu(CHOICE),qDtail(CHOICE).

  qSpeci(0).
  qSpeci(1) :- querySpeci(0).
  qSpeci(2) :- specID("1st"),querySpeci(1).
  qSpeci(3) :- specID("1st"),specID("2nd"),querySpeci(2).
  qSpeci(4) :- specID("1st"),specID("2nd"),specID("3rd"),querySpeci(3).

  qDtail(0).
  qDtail(1) :- queryDetail(0).
  qDtail(2) :-
    specID("1st"),con("1st Spec / Mix Concentration","HCcon1"),
    queryDetail(1).
  qDtail(3) :-
    specID("1st"),con("1st Spec / Mix Concentration","HCcon1"),
    specID("2nd"),con("2nd Spec / Mix Concentration","HCcon2"),
    queryDetail(2).
  qDtail(4) :-
    specID("1st"),con("1st Spec / Mix Concentration","HCcon1"),
    specID("2nd"),con("2nd Spec / Mix Concentration","HCcon2"),
    specID("3rd"),con("3rd Spec / Mix Concentration","HCcon3"),
    queryDetail(3).

  querySpeci(Num) :-
    write("Searching..."),
    time(0,0,0,0),
    specifind(Num),
    time(_,_,S,H),
    write("Time : ",S," Sec and ",H," hundredths"),
    print.
```

```
querySeries :-
  series,
  write("Searching..."),
  time(0,0,0,0),
  seriesfind,
  time(_,_,S,H),
  write("Time : ",S," Sec and ",H," hundredths"),
  print.

queryDate:-
  menu(10,50," Query by Date(s) ",
    [" Any",
     " Date(s)",
     " Month",
     " From date1 to date2 "
    ],CHOICE),
  qdate(CHOICE),
  print.

qdate(0):-write("Clearing memory...please wait"),clear_ans.
qdate(1):- ec(Date,A,B,C,D,E,F,W,Rk),
           assert(z(ec(Date,A,B,C,D,E,F,W,Rk))),fail.
qdate(1).
qdate(2):-
  write("Enter as many date(s) as applicable. "),
  write("Keep each date[yymmdd] on separate line."),
  write("Strike <return> twice to execute."),
  readln(Date),readdate(Date).
qdate(3):-
    drawbox(""),
    write(" Month [2 chars] : "),readln(M),
    removebox,
    monthfind(M).
qdate(4):-
  drawbox(" Date(s) "),
  write("Range from [yymmdd] : "),readln(D1),
  write("        to [yymmdd] : "),readln(D2),
  removebox,
  datefind(D1,D2).

  readdate(""):-!.
  readdate(Date):-
    ec(Date,A,B,C,D,E,F,W,Rk),!,
    assert(z(ec(Date,A,B,C,D,E,F,W,Rk))),readln(D1),readdate(D1).
  readdate(Date):-
    write("Run with date ",Date," not in database. Retry."),
    readln(D1),readdate(D1).
```

```
queryDetail(Num) :-
  con("Red Side NOx","redNOx"),
  con("Blue Side NOx","blueNOx"),
  others("# of HC instruments used","hcinst"),
  others("Overall Quality","qual"),
  others("Rank","rank"),
  others("Sun Light %","sun"),
  classes(" Any"),
  dilutes(" Any"),
  injects(" Any"),
  hctype(" Any"),
  projcode(" Any"),
  pcstatus(" Any"),
  series,
  write("Searching...please wait"),
  time(0,0,0,0),
  detailfind(Num),
  time(_,_,S,H),
  write("Time : ",S," Sec and ",H," hundredths"),
  print.

specID(S) :-
  repeat,
  drawbox(""),
  write(S," Species / Mixture ID Number : "),
  readint(X),!,assert(w(specID(S,X))),
  removebox.

classes(S) :-
  findall(Class,code_class(_,Class),L),
  menu(8,30,"Class Types",[S|L],CHOICE),
  findall(Code,code_class(Code,_),L1),
  CH=CHOICE-1,
  index(L1,CH,Code1),
  assert(x(class(Code1))),!.
classes(_).

dilutes(S) :-
  findall(Dilute,code_dilute(_,Dilute),L),
  menu(8,40,"Dilution Types",[S|L],CHOICE),
  findall(Code,code_dilute(Code,_),L1),
  CH=CHOICE-1,
  index(L1,CH,Code1),
  assert(x(dilute(Code1))),!.
dilutes(_).

injects(S) :-
  findall(Inject,code_inject(_,Inject),L),
  menu(8,36,"Injection Types",[S|L],CHOICE),
  findall(Code,code_inject(Code,_),L1),
  CH=CHOICE-1,
```

```
    index(L1,CH,Code1),
    assert(x(inject(Code1))),!.
  injects(_).

  hctype(S) :-
    findall(HC,code_hc(_,HC),L),
    menu(8,38,"HC Types",[S|L],CHOICE),
    findall(Code,code_hc(Code,_),L1),
    CH=CHOICE-1,
    index(L1,CH,Code1),
    assert(x(hctype(Code1))),!.
  hctype(_).

  projcode(S) :-
    findall(Proj,code_proj(_,Proj),L),
    menu(8,36,"Project Types",[S|L],CHOICE),
    findall(Code,code_proj(Code,_),L1),
    CH=CHOICE-1,
    index(L1,CH,Code1),
    assert(x(proj(Code1))),!.
  projcode(_).

  pcstatus(S) :-
    findall(PS,code_status(_,PS),L),
    menu(8,36,"Processing Status",[S|L],CHOICE),
    findall(Code,code_status(Code,_),L1),
    CH=CHOICE-1,
    index(L1,CH,Code1),
    assert(x(proces(Code1))),!.
  pcstatus(_).

  series :-
    menu(8,45,"Series Code",
      [ " Any",
        " Specifcic   "
      ],CHOICE),
    series1(CHOICE).

  series1(0).
  series1(1).
  series1(2) :-
    drawbox(""),
    write("Series Code [2 chars] : "),
    readln(X),assert(x(series(X))),
    removebox.

CLAUSES
  con(S1,S2) :-
    relmenu(S1,CHOICE),
    repeat,
    drawbox(S1),
```

```
    relreal(S2,CHOICE),!,
    removebox.

  others(S1,S2) :-
    relmenu(S1,CHOICE),
    repeat,
    drawbox(S1),
    relint(S2,CHOICE),!,
    removebox.

  relreal(_,0).
  relreal(_,1).
  relreal(String,2) :- write("Equals [ppm] : "),readreal(X),
                       assert(y(eq(String,r(X)))).
  relreal(String,3) :- write("Less than [ppm] : "),readreal(X),
                  assert(y(lt(String,r(X)))).
  relreal(String,4) :- write("Greater than [ppm] : "),readreal(X),
                  assert(y(gt(String,r(X)))).
  relreal(String,5) :- write("Range from [ppm] : "),readreal(X),
                  write("        to [ppm] : "),readreal(Y),
                  assert(y(range(String,r(X),r(Y)))).

  relint(_,0).
  relint(_,1).
  relint(String,2) :- write("Equals [integer] : "),readint(X),
                  assert(y(eq(String,i(X)))).
  relint(String,3) :- write("Less than [integer] : "),readint(X),
                  assert(y(lt(String,i(X)))).
  relint(String,4) :- write("Greater than [integer] : "),readint(X),
                  assert(y(gt(String,i(X)))).
  relint(String,5) :- write("Range from [integer] : "),readint(X),
                  write("        to [integer] : "),readint(Y),
                  assert(y(range(String,i(X),i(Y)))).

/*-------------------------------*/
/*     FINDING ANSWERS           */
/*-------------------------------*/
PREDICATES
  checkEC(CMPLIST,STRING,STRING,STRING,STRING,STRING,STRING,STRING)
  rel_r(RELLIST,STRING,REAL)
  rel_i(RELLIST,STRING,INTEGER)
  checkRB(RELLIST,STRING,INTEGER,
          REAL,INTEGER,REAL,INTEGER,REAL,INTEGER,
          REAL,INTEGER,REAL,INTEGER,REAL,INTEGER)

CLAUSES
  specifind(0) :-
    ec(X,A,B,C,D,E,F,W,Rk),
    assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
  specifind(0).
```

```
specifind(1) :-
  w(specID("1st",ID)),
  rb(X,_,H,I,J,K,L,M,_,O,P,Q,R,S,T,_,_,_),
  checkRB([],"HCcon1",ID,H,I,J,K,L,M,O,P,Q,R,S,T),
  ec(X,A,B,C,D,E,F,W,Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
specifind(1).
specifind(2) :-
  w(specID("1st",ID1)),
  w(specID("2nd",ID2)),
  rb(X,_,H,I,J,K,L,M,_,O,P,Q,R,S,T,_,_,_),
  checkRB([],"HCcon1",ID1,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB([],"HCcon2",ID2,H,I,J,K,L,M,O,P,Q,R,S,T),
  ec(X,A,B,C,D,E,F,W,Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
specifind(2).
specifind(3) :-
  w(specID("1st",ID1)),
  w(specID("2nd",ID2)),
  w(specID("3rd",ID3)),
  rb(X,_,H,I,J,K,L,M,_,O,P,Q,R,S,T,_,_,_),
  checkRB([],"HCcon1",ID1,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB([],"HCcon2",ID2,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB([],"HCcon3",ID3,H,I,J,K,L,M,O,P,Q,R,S,T),
  ec(X,A,B,C,D,E,F,W,Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
specifind(3).

detailfind(0) :-
  findall(Term,x(Term),Lis),
  findall(Rel,y(Rel),List),
  rb(X,G,_,_,_,_,_,_,N,_,_,_,_,_,_,U,V,SU),
  rel_r(List,"redNOx", G),
  rel_r(List,"blueNOx",N),
  rel_i(List,"hcinst", U),
  rel_i(List,"qual",   V),
  rel_i(List,"sun",    SU),
  ec(X,A,B,C,D,E,F,W,Rk),
  checkEC(Lis,A,B,C,D,E,F,W),
  rel_i(List,"rank",Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
detailfind(0).
detailfind(1) :-
  w(specID("1st",ID)),
  findall(Term,x(Term),Lis),
  findall(Rel,y(Rel),List),
  rb(X,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  checkRB(List,"HCcon1",ID,H,I,J,K,L,M,O,P,Q,R,S,T),
  rel_r(List,"redNOx", G),
  rel_r(List,"blueNOx",N),
  rel_i(List,"hcinst", U),
```

```
  rel_i(List,"qual",   V),
  rel_i(List,"sun",   SU),
  ec(X,A,B,C,D,E,F,W,Rk),
  checkEC(Lis,A,B,C,D,E,F,W),
  rel_i(List,"rank",Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
detailfind(1).
detailfind(2) :-
  w(specID("1st",ID1)),
  w(specID("2nd",ID2)),
  findall(Term,x(Term),Lis),
  findall(Rel,y(Rel),List),
  rb(X,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  checkRB(List,"HCcon1",ID1,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB(List,"HCcon2",ID2,H,I,J,K,L,M,O,P,Q,R,S,T),
  rel_r(List,"redNOx", G),
  rel_r(List,"blueNOx",N),
  rel_i(List,"hcinst", U),
  rel_i(List,"qual",   V),
  rel_i(List,"sun",   SU),
  ec(X,A,B,C,D,E,F,W,Rk),
  checkEC(Lis,A,B,C,D,E,F,W),
  rel_i(List,"rank",Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
detailfind(2).
detailfind(3) :-
  w(specID("1st",ID1)),
  w(specID("2nd",ID2)),
  w(specID("3rd",ID3)),
  findall(Term,x(Term),Lis),
  findall(Rel,y(Rel),List),
  rb(X,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  checkRB(List,"HCcon1",ID1,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB(List,"HCcon2",ID2,H,I,J,K,L,M,O,P,Q,R,S,T),
  checkRB(List,"HCcon3",ID3,H,I,J,K,L,M,O,P,Q,R,S,T),
  rel_r(List,"redNOx", G),
  rel_r(List,"blueNOx",N),
  rel_i(List,"hcinst", U),
  rel_i(List,"qual",   V),
  rel_i(List,"sun",   SU),
  ec(X,A,B,C,D,E,F,W,Rk),
  checkEC(Lis,A,B,C,D,E,F,W),
  rel_i(List,"rank",Rk),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
detailfind(3).

datefind(D1,D2):-
  ec(Date,A,B,C,D,E,F,W,Rk),
  Date>=D1,Date<=D2,
  assert(z(ec(Date,A,B,C,D,E,F,W,Rk))),fail.
datefind(_,_).
```

```
monthfind(M):-
  ec(Date,A,B,C,D,E,F,W,Rk),
  frontstr(2,Date,_,MMDD),
  frontstr(2,MMDD,MM,_),
  M=MM,
  assert(z(ec(Date,A,B,C,D,E,F,W,Rk))),fail.
monthfind(_).

seriesfind :-
  ec(X,A,B,C,D,E,F,W,Rk),
  findall(Se,x(Se),Lis),
  checkEC(Lis,A,B,C,D,E,F,W),
  assert(z(ec(X,A,B,C,D,E,F,W,Rk))),fail.
seriesfind.

checkRB(List,Spe,ID,Rc1,R1,Rc2,R2,Rc3,R3,Bc1,B1,Bc2,B2,Bc3,B3) :-
  ID=R1,rel_r(List,Spe,Rc1),! or
  ID=B1,rel_r(List,Spe,Bc1),! or
  ID=R2,rel_r(List,Spe,Rc2),! or
  ID=B2,rel_r(List,Spe,Bc2),! or
  ID=R3,rel_r(LIst,Spe,Rc3),! or
  ID=B3,rel_r(List,Spe,Bc3).

rel_r([],_,_) :- !.
rel_r([eq(Label,r(B))|_],Label,A) :- !,A=B.
rel_r([lt(Label,r(B))|_],Label,A) :- !,A<B.
rel_r([gt(Label,r(B))|_],Label,A) :- !,A>B.
rel_r([range(Label,r(B),r(C))|_],Label,A) :- !,B<=A,A<=C.
rel_r([_|T],Label,A) :- rel_r(T,Label,A).

rel_i([],_,_) :- !.
rel_i([eq(Label,i(B))|_],Label,A) :- !,A=B.
rel_i([lt(Label,i(B))|_],Label,A) :- !,A<B.
rel_i([gt(Label,i(B))|_],Label,A) :- !,A>B.
rel_i([range(Label,i(B),i(C))|_],Label,A) :- !,B<=A,A<=C.
rel_i([_|T],Label,A) :- rel_i(T,Label,A).

checkEC([],_,_,_,_,_,_,_).
checkEC([class(A)|T], A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([dilute(B)|T],A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([inject(C)|T],A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([hctype(D)|T],A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([proj(E)|T],  A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([proces(F)|T],A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
checkEC([series(W)|T],A,B,C,D,E,F,W) :- checkEC(T,A,B,C,D,E,F,W).
```

```
/*--------------------------------*/
/*  OSCECIS          EC27C.PRO    */
/*  Version 1        March 8, 87  */
/*  Copyright (C)    Kah-Eng Pua  */
/*--------------------------------*/
CODE = 1750
PROJECT "ECPROJ27.PRJ"
INCLUDE "ECDEF27.PRO"

/*-----------------------------*/
/*   PRINTING SOLUTIONS        */
/*-----------------------------*/
PREDICATES
  split(STRING,STRING,LISTS,LISTS)
  pmenu
  print0(INTEGER)
  print1(INTEGER)
  printSimple
  printByClass
  printClass(LISTS)
  printByDilution
  printDilute(LISTS)
  printByInjection
  printInject(LISTS)
  printByHCtype
  printHC(LISTS)
  printByProj
  printProj(LISTS)
  printByStatus
  printStatus(LISTS)
  printBySeries
  printSeries(LISTS)

  heading
  writeformat(STRING,STRING)
  writehc(REAL,INTEGER)
  writerb(STRING,REAL,REAL,INTEGER)
  ec_field(STRING,STRING,STRING,STRING,STRING,STRING,STRING,
           STRING,STRING,STRING,INTEGER)
  code_date(STRING,STRING)
  code_rank(INTEGER,STRING)
  writesol(LISTS)
  write_sol(LISTS,INTEGER)

  check_class(STRING,STRING)    /* Checking database consistency */
  check_dilute(STRING,STRING)
  check_inject(STRING,STRING)
  check_hc(STRING,STRING)
  check_proj(STRING,STRING)
  check_status(STRING,STRING)
  check_ser(STRING,STRING)
```

```
CLAUSES
  split(L,H,[A|X],[A|Y]) :- L<=A,A< H,!,split(L,H,X,Y).
  split(L,H,[_|X],Y) :- split(L,H,X,Y).
  split(_,_,[],[]).

  print :-
    repeat,
    menu(19,63,"Output device",
      [ " Screen",
        " Line Printer  ",
        " File"
      ],CHOICE),
    print0(CHOICE),
    CHOICE=0,!.

  print0(0) :- write("Clearing memory... please wait"),clear_ans.
  print0(1) :- pmenu.
  print0(2) :- writedevice(printer),pmenu,writedevice(screen).
  print0(3) :- write("Name of your output file : "),readln(Name),
    openwrite(outf,Name),writedevice(outf),pmenu,closefile(outf).

  pmenu :-
    repeat,
    menu(14,50,"Output menu",
      [ "Simple output (date only)",
        "Grouped by class",
        "Grouped by dilution type",
        "Grouped by injection type",
        "Grouped by HC type",
        "Grouped by project type",
        "Grouped by processing status",
        "Grouped by series"
      ],CHOICE),
    shiftwindow(19),
    shiftwindow(2),
    clearwindow,
    print1(CHOICE),
    pauses(CHOICE),
    CHOICE=0,!.

  print1(0).
  print1(1) :- printSimple.
  print1(2) :- printByClass.
  print1(3) :- printByDilution.
  print1(4) :- printByInjection.
  print1(5) :- printByHCtype.
  print1(6) :- printByProj.
  print1(7) :- printByStatus.
  print1(8) :- printBySeries.
```

```
printSimple :-
  findall(Date,z(ec(Date,_,_,_,_,_,_,_,_,_)),L),
  write(">>>>> Date of Run <<<<<"),
  split("000000","780000",L,L1),
  split("780000","790000",L,L2),
  split("790000","800000",L,L3),
  split("800000","810000",L,L4),
  split("810000","820000",L,L5),
  split("820000","830000",L,L6),
  split("830000","840000",L,L7),
  split("840000","850000",L,L8),
  split("850000","860000",L,L9),
  split("860000","999999",L,L10),
  writesol(L1),
  writesol(L2),
  writesol(L3),
  writesol(L4),
  writesol(L5),
  writesol(L6),
  writesol(L7),
  writesol(L8),
  writesol(L9),
  writesol(L10),
  listlen(L,Leng),
  write("Total number of runs ",Leng),nl.

printByClass :-
  findall(Class,z(ec(_,Class,_,_,_,_,_,_,_,_)),L),
  unik(L,L1),
  heading,
  printClass(L1).

  printClass([]).
  printClass([C|Cl]) :-
    check_class(C,Class),
    write(">>>>> For the Class type of ",C,Class," <<<<<"),
    writeformat("class",C),
    findall(X,z(ec(X,C,_,_,_,_,_,_,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this class is ",Num),nl,
    printClass(Cl).

printByDilution :-
  findall(Dilute,z(ec(_,_,Dilute,_,_,_,_,_,_,_)),L),
  unik(L,L1),
  heading,
  printDilute(L1).
```

```
  printDilute([]).
  printDilute([D|Dl]) :-
    check_dilute(D,Dilute),
    write(">>>>> For the Dilution type of ",D,Dilute," <<<<<"),
    writeformat("dilute",D),
    findall(X,z(ec(X,_,D,_,_,_,_,_,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this dilution type is ",Num),nl,
    printDilute(Dl).

printByInjection :-
  findall(Inject,z(ec(_,_,_,Inject,_,_,_,_,_,_)),L),
  unik(L,Ll),
  heading,
  printInject(Ll).

  printInject([]).
  printInject([I|Il]) :-
    check_inject(I,Inject),
    write(">>>>> For the Injection type of ",I,Inject," <<<<<"),
    writeformat("inject",I),
    findall(X,z(ec(X,_,_,I,_,_,_,_,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this injection type is ",Num),nl,
    printInject(Il).

printByHCtype :-
  findall(HC,z(ec(_,_,_,_,HC,_,_,_,_,_)),L),
  unik(L,Ll),
  heading,
  printHC(Ll).

  printHC([]).
  printHC([HC|HCl]) :-
    check_hc(HC,HCtype),
    write(">>>>> For the HC type of ",HC,HCtype," <<<<<"),
    writeformat("hctype",HC),
    findall(X,z(ec(X,_,_,_,HC,_,_,_,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this HC type is ",Num),nl,
    printHC(HCl).

printByProj :-
  findall(Proj,z(ec(_,_,_,_,_,_,Proj,_,_,_)),L),
  unik(L,Ll),
  heading,
  printProj(Ll).
```

```
  printProj([]).
  printProj([P|Pl]) :-
    check_proj(P,Proj),
    write(">>>>> For the Project type of ",P,Proj," <<<<<"),
    writeformat("proj",P),
    findall(X,z(ec(X,_,_,_,_,_,P,_,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this project type is ",Num),nl,
    printProj(Pl).

printByStatus :-
  findall(Status,z(ec(_,_,_,_,_,_,_,Status,_,_)),L),
  unik(L,Ll),
  heading,
  printStatus(Ll).

  printStatus([]).
  printStatus([S|Sl]) :-
    check_status(S,Status),
    write(">>>> For the Processing status of ",S,Status," <<<<"),
    writeformat("status",S),
    findall(X,z(ec(X,_,_,_,_,_,_,S,_,_)),List),
    listlen(List,Num),
    write("Number of runs of this processing status is ",Num),nl,
    printStatus(Sl).

printBySeries :-
  findall(Series,z(ec(_,_,_,_,_,_,_,_,Series,_)),L),
  unik(L,Ll),
  heading,
  printSeries(Ll).

  printSeries([]).
  printSeries([S|Sl]) :-
    check_ser(S,Series),
    write(">>>>> For the Series type of ",S,Series," <<<<<"),
    writeformat("series",S),
    findall(X,z(ec(X,_,_,_,_,_,_,_,S,_)),List),
    listlen(List,Num),
    write("Number of runs of this series is ",Num),nl,
    printSeries(Sl).

heading:-
  write("-------------------------------------------------"),
  write("   Date   CDI HC  Ser   Proj Q I P   Sun   Rank"),
  write("-------------------------------------------------").

writeformat(Label,Field) :-
  ec_field(Label,Field,Date,Class,Dilute,Inject,HC,Proj,Status,Ser,
           Rank),
```

```
  rb(Date,RedNOx,RHCcon1,RHCcode1,RHCcon2,RHCcode2,RHCcon3,RHCcode3,
        BlueNOx,BHCcon1,BHCcode1,BHCcon2,BHCcode2,BHCcon3,BHCcode3,
        HCinst,Qual,Sun),nl,
  code_date(Date,Date1),
  code_rank(Rank,Rank1),
  write(Date1," ",Class,Dilute,Inject," ",HC,"  ",Ser,"     ",
        Proj,"  ",Qual," ",HCinst," ",Status,"   ",Sun,"   ",
        Rank1),nl,
  writerb("Red  Side",RedNOx,RHCcon1,RHCcode1),
  writehc(RHCcon2,RHCcode2),
  writehc(RHCcon3,RHCcode3),
  writerb("Blue Side",BlueNOx,BHCcon1,BHCcode1),
  writehc(BHCcon2,BHCcode2),
  writehc(BHCcon3,BHCcode3),nl,fail.
writeformat(_,_).

writehc(_,0) :- !.
writehc(Con,Code) :- nl,
  code_spec(Code,Code1),!,
  writef("%40.2f   ",Con),
  write(Code1).
writehc(Con,Code) :-
  writef("%40.2f   ",Con),
  write(" NEW SPECIES ",Code," !!!").

writerb(_,_,_,0) :- !.
writerb(Side,NOx,Con,Code) :- nl,
  code_spec(Code,Code1),!,
  write("          ",Side),
  writef("%8.3f  NOx  %6.2f   ",NOx,Con),
  write(Code1).
writerb(Side,NOx,Con,Code):-
  write("          ",Side),
  writef("%8.3f  NOx  %6.2f   ",NOx,Con),
  write(" NEW SPECIES ",Code," !!!").

ec_field("class" ,A, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("dilute",B, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("inject",C, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("hctype",D, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("proj"  ,E, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("status",F, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).
ec_field("series",G, X,A,B,C,D,E,F,G,R) :- z(ec(X,A,B,C,D,E,F,G,R)).

code_date(Date,NewDate) :-
  frontstr(2,Date,YY,MMDD),
  frontstr(2,MMDD,MM,DD),
  code_month(MM,NewM),
  concat(DD,"-",C1),
  concat(C1,NewM,C2),
  concat(C2,"-",C3),
  concat(C3,YY,NewDate).
```

```
code_rank(1,"Best in HC type"):- !.
code_rank(I,"Best of kind")   :- 1<I,I<11,!.
code_rank(I,"Sastisfactory")  :- 10<I,I<51,!.
code_rank(I,"Supporting")     :-  50<I,I<101,!.
code_rank(I,"Some Problems")  :- 100<I,I<501,!.
code_rank(_,"Not for modeling").

/* Checking database consistency */

check_class(C,Class):-
  code_class(C,Class),!.
check_class(_," NEW CLASS !!!").

check_dilute(D,Dilute):-
  code_dilute(D,Dilute),!.
check_dilute(_," NEW DILUTION TYPE!!!").

check_inject(I,Inject):-
  code_inject(I,Inject),!.
check_inject(_," NEW INJECTION TYPE!!!").

check_hc(HC,HCtype):-
  code_hc(HC,HCtype),!.
check_hc(_," NEW HC TYPE!!!").

check_proj(P,Proj):-
  code_proj(P,Proj),!.
check_proj(_," NEW PROJECT!!!").

check_status(S,Status):-
  code_status(S,Status),!.
check_status(_," NEW STATUS!!!").

check_ser(S,Series):-
  code_series(S,Series),!.
check_ser(_," NEW SERIES!!!").

writesol([]):-!.
writesol(L) :-nl,write_sol(L,0).

write_sol([],_):-nl.
write_sol([H|T],10) :- !,write(H," "),nl,write_sol(T,0).
write_sol([H|T], N) :- write(H," "),N1=N+1,write_sol(T,N1).

/*-------------------------*/
/*    UPDATING DATABASE    */
/*-------------------------*/
PREDICATES
  update1(INTEGER)
  displays(STRING)
  inserts(STRING)
```

```
  insertRec(STRING)
  modify(STRING)
  modifyRec(STRING)

  i_hcID(STRING,STRING)
  i_conc(STRING,STRING)
  i_other(STRING,STRING)
  i_series
  m_hcID(STRING,STRING)
  m_conc(STRING,STRING)
  m_other(STRING,STRING)
  m_series

  imenu(STRING,INTEGER)
  yr(STRING,REAL,REAL)
  yi(STRING,INTEGER,INTEGER)
  mw(STRING,INTEGER,INTEGER)
  mx(STRING,STRING,STRING)

CLAUSES
  updatedb :-
    repeat,
    menu(1,59,"Update Database",
      [ " Display record   ",
        " Delete record",
        " Insert record",
        " Modify record"
      ],CHOICE),
    update1(CHOICE),
    pauses(CHOICE),
    CHOICE=0,!.

  update1(0).
  update1(1) :-
    write("Displaying...----------"),
    write("Enter Date of Run [yymmdd] : "),readln(Date),
    displays(Date).
  update1(2) :-
    write("Deleting...---------"),
    write("Enter Date of Run [yymmdd] : "),readln(Date),
    write("Are you sure you want to delete this record? (y/n) "),
    readchar(YN),yes(YN),
    write("Run with date ",Date," has been deleted."),
    retract(ec(Date,_,_,_,_,_,_,_,_)),
    retract(rb(Date,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_)).
  update1(3) :-
    clear_facts,
    write("Inserting...----------"),
    write("Enter Date of Run [yymmdd] : "),readln(Date),
    inserts(Date).
```

```
updatel(4) :-
  clear_facts,
  write("Modifying..........."),
  write("Enter Date of Run [yymmdd] : "),readln(Date),
  modify(Date).

displays(Date) :-
  ec(Date,A,B,C,D,E,F,W1,W2),!,
  rb(Date,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,SU),
  writef("Date   -%-6 Class  -%-6 Dilution    -%-6 Injection-%-2",
         Date,A,B,C),
  writef("HC Type-%-6 Project-%-6 Proce Status-%-6 Series   -%-2",
         D,E,F,W1),
  writef("HC Inst-%-6 Quality-%-6 Rank of Run -%-6 Sun Light-%-2",
         U,V,W2,SU),
  writef("Red NOx-%-8.3f Red Con 1 -%-8.2f Red HC 1 - %-2",G,H,I),
  writef("               Red Con 2  - %-8.2f Red HC 2  - %-2",J,K),
  writef("               Red Con 3  - %-8.2f Red HC 3  - %-2",L,M),
  writef("Blue NOx-%-8.3f Blue Con 1-%-8.2f Blue HC 1-%-2",N,O,P),
  writef("               Blue Con 2 - %-8.2f Blue HC 2 - %-2",Q,R),
  writef("               Blue Con 3 - %-8.2f Blue HC 3 - %-2",S,T).
displays(_) :- write("Warning: Record not in database.").

inserts(D) :-
  ec(D,_,_,_,_,_,_,_,_),!,
  write("Warning : Record with same date already in database.").
inserts(Date) :-
  i_conc("Red Side NOx Concentration","R0"),
  i_hcID("Red Side 1st HC ID number","R1"),
  i_conc("Red Side 1st HC Concentration","R1"),
  i_hcID("Red Side 2nd HC ID number","R2"),
  i_conc("Red Side 2nd HC Concentration","R2"),
  i_hcID("Red Side 3rd HC ID number","R3"),
  i_conc("Red Side 3rd HC Concentration","R3"),
  i_conc("Blue Side NOx Concentration","B0"),
  i_hcID("Blue Side 1st HC ID number","B1"),
  i_conc("Blue Side 1st HC Concentration","B1"),
  i_hcID("Blue Side 2nd HC ID number","B2"),
  i_conc("Blue Side 2nd HC Concentration","B2"),
  i_hcID("Blue Side 3rd HC ID number","B3"),
  i_conc("Blue Side 3rd HC Concentration","B3"),
  i_other("Number of HC instruments","hcinst"),
  i_other("Overall quality","qual"),
  i_other("Rank","rank"),
  i_other("Sun Light %","sun"),
  classes(""),
  dilutes(""),
  injects(""),
  hctype(""),
  projcode(""),
  pcstatus(""),
```

```
  i_series,
  write("Are your inputs correct? (y/n) "),
  readchar(Yn),yes(Yn),
  insertRec(Date),
  write("Run with date ",Date," has been inserted.").

insertRec(Date) :-
  y(eq("R0",r(R0))),w(specID("R1",I1)),
  w(specID("R2",I2)),w(specID("R3",I3)),
  y(eq("R1",r(R1))),y(eq("R2",r(R2))),
  y(eq("R3",r(R3))),
  y(eq("B0",r(B0))),w(specID("B1",I4)),
  w(specID("B2",I5)),w(specID("B3",I6)),
  y(eq("B1",r(B1))),y(eq("B2",r(B2))),
  y(eq("B3",r(B3))),
  y(eq("hcinst",i(HC))),
  y(eq("qual",i(QL))),
  y(eq("rank",i(RK))),
  y(eq("sun",i(SU))),
  assert(rb(Date,R0,R1,I1,R2,I2,R3,I3,B0,B1,I4,B2,I5,B3,I6,HC,QL,SU)),
  x(class(CT)),x(dilute(DT)),
  x(inject(IT)),x(hctype(HT)),
  x(proj(PT)),x(proces(PS)),
  x(series(ST)),
  assert(ec(Date,CT,DT,IT,HT,PT,PS,ST,RK)).

modify(Date) :-
  ec(Date,_,_,_,_,_,_,_,_),!,
  m_conc("Red Side NOx Concentration","R0"),
  m_hcID("Red Side 1st HC ID number","R1"),
  m_conc("Red Side 1st HC Concentration","R1"),
  m_hcID("Red Side 2nd HC ID number","R2"),
  m_conc("Red Side 2nd HC Concentration","R2"),
  m_hcID("Red Side 3rd HC ID number","R3"),
  m_conc("Red Side 3rd HC Concentration","R3"),
  m_conc("Blue Side NOx Concentration","B0"),
  m_hcID("Blue Side 1st HC ID number","B1"),
  m_conc("Blue Side 1st HC Concentration","B1"),
  m_hcID("Blue Side 2nd HC ID number","B2"),
  m_conc("Blue Side 2nd HC Concentration","B2"),
  m_hcID("Blue Side 3rd HC ID number","B3"),
  m_conc("Blue Side 3rd HC Concentration","B3"),
  m_other("Number of HC instruments","hcinst"),
  m_other("Overall quality","qual"),
  m_other("Rank","rank"),
  m_other("Sun Light %","sun"),
  classes(" Not to be modified"),
  dilutes(" Not to be modified"),
  injects(" Not to be modified"),
  hctype(" Not to be modified"),
  projcode(" Not to be modified"),
```

```
    pcstatus(" Not to be modified"),
    m_series,
    write("Are your inputs correct? (y/n) "),
    readchar(Yn),yes(Yn),
    modifyRec(Date),
    write("Record with date ",Date," has been modified.").
  modify(_) :- write("Warning: Record not in database.").

  modifyRec(Date) :-
    retract(ec(Date,Cl,Di,In,Ht,Pj,St,Se,Rk)),
    mx("class",Cl,Cll), mx("dilute",Di,Dil), mx("inject",In,Inl),
    mx("hctype",Ht,Htl),mx("proj",Pj,Pjl),   mx("status",St,Stl),
    mx("series",Se,Sel),
    yi("rank",RK,RKn),
    assert(ec(Date,Cll,Dil,Inl,Htl,Pjl,Stl,Sel,Rkn)),
    retract(rb(Date,R0,R1,I1,R2,I2,R3,I3,B0,B1,I4,B2,I5,B3,I6,HC,QL,SU)),
    yr("R0",R0,R0n),
    yr("R1",R1,R1n), mw("R1",I1,I1n),
    yr("R2",R2,R2n), mw("R2",I2,I2n),
    yr("R3",R3,R3n), mw("R3",I3,I3n),
    yr("B0",B0,B0n),
    yr("B1",B1,B1n), mw("B1",I4,I4n),
    yr("B2",B2,B2n), mw("B2",I5,I5n),
    yr("B3",B3,B3n), mw("B3",I6,I6n),
    yi("hcinst",HC,HCn),
    yi("qual",QL,QLn),
    yi("sun",SU,SUn),
    assert(rb(Date,R0n,R1n,I1n,R2n,I2n,R3n,I3n,
                   B0n,B1n,I4n,B2n,I5n,B3n,I6n,HCn,QLn,SUn)),!.

CLAUSES
  imenu(Title,CHOICE) :-
    menu(12,35,Title,
      ["       Not to be modified              ",
       "       To be modified"
      ],CHOICE).

  i_hcID(S1,N) :-
    drawbox(""),
    write(S1," [integer] : "),readint(X),
    assert(w(specID(N,X))),
    removebox.

  i_conc(S1,S2) :-
    drawbox(""),
    write(S1," [ppmC] : "),readreal(X),
    assert(y(eq(S2,r(X)))),
    removebox.
```

```
i_other(S1,S2) :-
  drawbox(""),
  write(S1," [integer] : "),readint(X),
  assert(y(eq(S2,i(X)))),
  removebox.

i_series:-
  drawbox(""),
  write("Series Code [2 chars] : "),readln(X),
  assert(x(series(X))),
  removebox.

m_hcID(S1,N) :-
  imenu(S1,CHOICE),CHOICE > 1,!,
  drawbox(""),
  write(S1," [integer] : "),readint(X),
  assert(w(specID(N,X))),
  removebox.
m_hcID(_,_).

m_conc(S1,S2) :-
  imenu(S1,CHOICE),CHOICE > 1,!,
  drawbox(""),
  write(S1," [ppmC] : "),readreal(X),
  assert(y(eq(S2,r(X)))),
  removebox.
m_conc(_,_).

m_other(S1,S2) :-
  imenu(S1,CHOICE),CHOICE > 1,!,
  drawbox(""),
  write(S1," [integer] : "),readint(X),
  assert(y(eq(S2,i(X)))),
  removebox.
m_other(_,_).

m_series:-
  imenu("Series Code",CHOICE),CHOICE > 1,!,
  drawbox(""),
  write("Series Code [2 chars] : "),readln(X),
  assert(x(series(X))),
  removebox.
m_series.

yr(S,_,New) :- y(eq(S,r(New))),!.
yr(_,Old,Old).

yi(S,_,New) :- y(eq(S,i(New))),!.
yi(_,Old,Old).

mw(S,_,New) :- w(specID(S,New)),!.
mw(_,Old,Old).
```

```
mx("class", _,New) :- x(class(New)),!.
mx("dilute",_,New) :- x(dilute(New)),!.
mx("inject",_,New) :- x(inject(New)),!.
mx("hctype",_,New) :- x(hctype(New)),!.
mx("proj",  _,New) :- x(proj(New)),!.
mx("status",_,New) :- x(proces(New)),!.
mx("series",_,New) :- x(series(New)),!.
mx(_,Old,Old).
```

D

Sample Outputs

This appendix contains sample outputs of ethylene runs in the experimental conditions database. One way to query ethylene runs is to choose the "Query By Species" panel in the "Query Type" menu of OSCECIS, followed by entering the species code (2) of ethylene to the system.

The first part of the outputs was obtained by the "Simple output (date only)" option in the "Output Menu", whereas the latter two parts were obtained by choosing "Grouped by series" and "Grouped by HC type" options respectively.

>>>>> Date of Run <<<<<

771018 771112 771119 771120

780110 780616 780701 780730 780806 780815 780821 780823 780914 780915 780918
780919 780921 781003 781012 781018 781025 781107

790804 790805 790816 790907

800708 800801 800814 800826

810708 810930

830916 831015

841005 841011 841012

860704 860709 860712 860713 860716

Total number of runs 42

Date	CDI	HC	Ser	Proj	Q	I	P	Sun	Rank
>>>>> For the Series type of 09 Matched other <<<<<									
23-Aug-78	ONI	1O	09	ET	8	0	S	97	Best of kind
	Red Side		0.507	NOx				2.93	ETHYLENE
	Blue Side		0.521	NOx				2.97	ETHYLENE
Number of runs of this series is 1									
>>>>> For the Series type of 08 Matched mixture <<<<<									
07-Sep-79	ONI	MB	08	ET	5	0	S	33	Best of kind
	Red Side		0.547	NOx				0.09	ETHYLENE
								1.25	FORMALDEHYDE
	Blue Side		0.549	NOx				0.06	ETHYLENE
								1.25	FORMALDEHYDE
01-Aug-80	ONI	MB	08	ET	9	0	S	73	Best of kind
	Red Side		0.533	NOx				0.48	ETHYLENE
								0.23	TRANS-2-BUTENE
	Blue Side		0.551	NOx				0.46	ETHYLENE
								0.23	TRANS-2-BUTENE
14-Aug-80	ONI	MB	08	ET	6	0	S	74	Sastisfactory
	Red Side		0.470	NOx				0.91	ETHYLENE
								0.29	TRANS-2-BUTENE
	Blue Side		0.466	NOx				0.95	ETHYLENE
								0.32	TRANS-2-BUTENE
Number of runs of this series is 3									
>>>>> For the Series type of 34 Delta NOx <<<<<									
15-Sep-78	ONI	1U	34	ET	3	0	S	53	Supporting
	Red Side		0.526	NOx				2.98	ETHYLENE
	Blue Side		0.268	NOx				2.00	FORMALDEHYDE

```
04-Jul-86 ONI 10  34     ET  5 0 P   0   Best of kind

          Red  Side   0.150  NOx    1.77    ETHYLENE
          Blue Side   0.310  NOx    1.88    ETHYLENE

Number of runs of this series is 2

>>>>> For the Series type of 33 Delta HC <<<<<

18-Oct-77 ONI 10  33     ET  6 0 S   92   Best of kind

          Red  Side   0.494  NOx    3.84    ETHYLENE
          Blue Side   0.484  NOx    1.88    ETHYLENE

08-Jul-80 ONI 10  33     ET  5 0 S   32   Best of kind

          Red  Side   0.754  NOx    2.10    ETHYLENE
          Blue Side   0.754  NOx    3.23    ETHYLENE

26-Aug-80 ONI 10  33     ET  8 0 S   100   Best of kind

          Red  Side   0.472  NOx    1.99    ETHYLENE
          Blue Side   0.464  NOx    1,00    ETHYLENE

05-Oct-84 ONI 10  33     ET  9 5 U   70   Best of kind

          Red  Side   0.365  NOx    3.13    ETHYLENE
          Blue Side   0.370  NOx    1.80    ETHYLENE

09-Jul-86 ONI 10  33     ET  5 0 P   0   Sastisfactory

          Red  Side   0.350  NOx    1.00    ETHYLENE
          Blue Side   0.350  NOx    2.00    ETHYLENE

Number of runs of this series is 5

>>>>> For the Series type of 18 Reactivity Comparison <<<<<

12-Nov-77 ONI 1U  18     ET  6 0 S   83   Sastisfactory

          Red  Side   0.503  NOx    2.00    ETHYLENE
          Blue Side   0.471  NOx    1.72    ACETALDEHYDE

19-Nov-77 ONI 1U  18     ET  7 0 S   99   Sastisfactory

          Red  Side   0.451  NOx    3.78    ETHYLENE
          Blue Side   0.832  NOx    3.82    ACETALDEHYDE
```

20-Nov-77	ONI 1U	18	ET	8 0 S	100	Sastisfactory	
	Red Side	0.446	NOx		4.37	ETHYLENE	
	Blue Side	0.881	NOx		3.92	ACETALDEHYDE	
10-Jan-78	ONI 1O	18	ET	8 0 S	100	Sastisfactory	
	Red Side	0.462	NOx		3.25	PROPYLENE	
	Blue Side	0.482	NOx		4.35	ETHYLENE	
16-Jun-78	ONI 1O	18	ET	7 0 S	91	Sastisfactory	
	Red Side	0.634	NOx		3.95	ETHYLENE	
	Blue Side	0.640	NOx		2.00	PROPYLENE	
01-Jul-78	ONI 1O	18	ET	7 0 S	83	Best of kind	
	Red Side	0.887	NOx		1.45	ETHYLENE	
	Blue Side	0.935	NOx		1.51	PROPYLENE	
30-Jul-78	ONI 1O	18	ET	6 0 S	82	Sastisfactory	
	Red Side	0.474	NOx		1.32	ETHYLENE	
	Blue Side	0.483	NOx		1.25	PROPYLENE	
06-Aug-78	ONI 1O	18	ET	4 0 S	57	Some Problems	
	Red Side	0.561	NOx		1.69	PROPYLENE	
	Blue Side	0.569	NOx		2.59	ETHYLENE	
15-Aug-78	ONI 1O	18	ET	7 0 S	87	Best of kind	
	Red Side	0.543	NOx		1.45	PROPYLENE	
	Blue Side	0.563	NOx		1.58	ETHYLENE	
21-Aug-78	ONI 1O	18	ET	9 0 S	100	Sastisfactory	
	Red Side	0.978	NOx		1.39	ETHYLENE	
	Blue Side	0.981	NOx		1.28	PROPYLENE	
14-Sep-78	ONI 1W	18	ET	5 0 S	71	Sastisfactory	
	Red Side	0.292	NOx		2.24	TOLUENE	
	Blue Side	0.294	NOx		0.97	ETHYLENE	
18-Sep-78	ONI 1W	18	ET	5 0 S	91	Supporting	
	Red Side	0.503	NOx		2.98	ETHYLENE	
	Blue Side	0.496	NOx		4.21	TOLUENE	

19-Sep-78	ONI 1U	18	ET	7 0 S	91	Supporting	
	Red Side	0.688	NOx		1.88	ETHYLENE	
	Blue Side	0.690	NOx		2.00	FORMALDEHYDE	
21-Sep-78	ONI 1U	18	ET	5 0 S	90	Supporting	
	Red Side	0.258	NOx		1.93	ETHYLENE	
	Blue Side	0.257	NOx		1.97	FORMALDEHYDE	
03-Oct-78	ONI MB	18	ET	6 0 S	49	Sastisfactory	
	Red Side	0.497	NOx		0.98	ETHYLENE	
	Blue Side	0.492	NOx		0.14	N-BUTANE	
					1.36	PROPYLENE	
12-Oct-78	ONI MB	18	ET	9 0 S	92	Sastisfactory	
	Red Side	0.479	NOx		0.20	ETHYLENE	
					1.26	ACETALDEHYDE	
	Blue Side	0.479	NOx		1.33	PROPYLENE	
18-Oct-78	ONI 1O	18	ET	9 0 S	83	Sastisfactory	
	Red Side	0.456	NOx		3.12	ETHYLENE	
	Blue Side	0.456	NOx		1.52	PROPYLENE	
25-Oct-78	ONI MB	18	ET	9 0 S	100	Sastisfactory	
	Red Side	0.444	NOx		0.20	ETHYLENE	
					1.16	ACETALDEHYDE	
	Blue Side	0.442	NOx		1.22	PROPYLENE	
07-Nov-78	ONI 1O	18	ET	5 0 S	64	Sastisfactory	
	Red Side	0.441	NOx		1.37	PROPYLENE	
	Blue Side	0.441	NOx		2.67	ETHYLENE	
04-Aug-79	ONI 1U	18	ET	6 0 S	59	Sastisfactory	
	Red Side	0.227	NOx		0.88	ETHYLENE	
	Blue Side	0.230	NOx		0.56	FORMALDEHYDE	
05-Aug-79	ONI 1U	18	ET	5 0 S	14	Best of kind	
	Red Side	0.638	NOx		4.11	ETHYLENE	
	Blue Side	0.539	NOx		1.20	FORMALDEHYDE	

16-Aug-79	ONI	MB	18	ET	9 0 S	97	Best of kind
	Red Side		0.443	NOx		1.42	PROPYLENE
	Blue Side		0.436	NOx		0.17	ETHYLENE
						1.30	ACETALDEHYDE
08-Jul-81	ONI	10	18	ET	7 4 L	76	Sastisfactory
	Red Side		0.500	NOx		2.00	ISOPRENE
	Blue Side		0.500	NOx		2.00	ETHYLENE
30-Sep-81	ONI	MC	18	RE	8 5 L	94	Best of kind
	Red Side		0.300	NOx		1.40	UNCMIX
						0.60	SIMARO
	Blue Side		0.300	NOx		0.17	BUTANE/PROPYLENE
						0.13	ETHYLENE
						0.10	TRANS-2-BUTENE
16-Sep-83	ONI	MB	18	RE	5 7 P	72	Best of kind
	Red Side		0.228	NOx		0.43	ETHYLENE
						1.57	1-BUTENE
						1.00	PROPYLENE
	Blue Side		0.208	NOx		0.84	ETHYLENE
						1.02	1-BUTENE
						1.04	PROPYLENE
15-Oct-83	ONI	MB	18	RE	8 5 P	76	Best of kind
	Red Side		0.481	NOx		1.13	PROPYLENE
						0.44	ETHYLENE
						1.60	1-BUTENE
	Blue Side		0.497	NOx		1.00	PROPYLENE
						0.88	ETHYLENE
						1.10	1-BUTENE
11-Oct-84	ONI	10	18	ET	9 5 U	80	Sastisfactory
	Red Side		0.351	NOx		2.86	ETHYLENE
	Blue Side		0.354	NOx		2.25	PROPYLENE
12-Oct-84	ONI	10	18	ET	9 5 U	80	Sastisfactory
	Red Side		0.710	NOx		2.68	ETHYLENE
	Blue Side		0.682	NOx		1.96	PROPYLENE

12-Jul-86	ONI	10	18	ET	5	0 U	0	Sastisfactory
	Red Side		0.350	NOx			2.00	ETHYLENE
	Blue Side		0.350	NOx			1.00	PROPYLENE
13-Jul-86	ONI	10	18	ET	5	0 P	0	Best of kind
	Red Side		0.350	NOx			0.50	PROPYLENE
	Blue Side		0.350	NOx			1.00	ETHYLENE
16-Jul-86	ONI	1U	18	ET	5	0 U	0	Sastisfactory
	Red Side		0.350	NOx			2.00	ETHYLENE
	Blue Side		0.350	NOx			1.00	FORMALDEHYDE

Number of runs of this series is 31

```
-------------------------------------------------
   Date   CDI HC  Ser   Proj Q I P   Sun   Rank
-------------------------------------------------

>>>>> For the HC type of 1W Single, ole vs aro <<<<<

14-Sep-78 ONI 1W  18     ET  5 0 S   71   Sastisfactory

          Red  Side   0.292  NOx    2.24    TOLUENE
          Blue Side   0.294  NOx    0.97    ETHYLENE

18-Sep-78 ONI 1W  18     ET  5 0 S   91   Supporting

          Red  Side   0.503  NOx    2.98    ETHYLENE
          Blue Side   0.496  NOx    4.21    TOLUENE

Number of runs of this HC type is 2

>>>>> For the HC type of MC Mixture, complex <<<<<

30-Sep-81 ONI MC  18     RE  8 5 L   94   Best of kind

          Red  Side   0.300  NOx    1.40    UNCMIX
                                     0.60    SIMARO
          Blue Side   0.300  NOx    0.17    BUTANE/PROPYLENE
                                     0.13    ETHYLENE
                                     0.10    TRANS-2-BUTENE

Number of runs of this HC type is 1

>>>>> For the HC type of MB Mixture, simple <<<<<

03-Oct-78 ONI MB  18     ET  6 0 S   49   Sastisfactory

          Red  Side   0.497  NOx    0.98    ETHYLENE
          Blue Side   0.492  NOx    0.14    N-BUTANE
                                     1.36    PROPYLENE

12-Oct-78 ONI MB  18     ET  9 0 S   92   Sastisfactory

          Red  Side   0.479  NOx    0.20    ETHYLENE
                                     1.26    ACETALDEHYDE
          Blue Side   0.479  NOx    1.33    PROPYLENE

25-Oct-78 ONI MB  18     ET  9 0 S   100  Sastisfactory

          Red  Side   0.444  NOx    0.20    ETHYLENE
                                     1.16    ACETALDEHYDE
          Blue Side   0.442  NOx    1.22    PROPYLENE
```

16-Aug-79	ONI	MB	18	ET	9 0 S	97	Best of kind
	Red	Side	0.443	NOx		1.42	PROPYLENE
	Blue	Side	0.436	NOx		0.17	ETHYLENE
						1.30	ACETALDEHYDE
07-Sep-79	ONI	MB	08	ET	5 0 S	33	Best of kind
	Red	Side	0.547	NOx		0.09	ETHYLENE
						1.25	FORMALDEHYDE
	Blue	Side	0.549	NOx		0.06	ETHYLENE
						1.25	FORMALDEHYDE
01-Aug-80	ONI	MB	08	ET	9 0 S	73	Best of kind
	Red	Side	0.533	NOx		0.48	ETHYLENE
						0.23	TRANS-2-BUTENE
	Blue	Side	0.551	NOx		0.46	ETHYLENE
						0.23	TRANS-2-BUTENE
14-Aug-80	ONI	MB	08	ET	6 0 S	74	Sastisfactory
	Red	Side	0.470	NOx		0.91	ETHYLENE
						0.29	TRANS-2-BUTENE
	Blue	Side	0.466	NOx		0.95	ETHYLENE
						0.32	TRANS-2-BUTENE
16-Sep-83	ONI	MB	18	RE	5 7 P	72	Best of kind
	Red	Side	0.228	NOx		0.43	ETHYLENE
						1.57	1-BUTENE
						1.00	PROPYLENE
	Blue	Side	0.208	NOx		0.84	ETHYLENE
						1.02	1-BUTENE
						1.04	PROPYLENE
15-Oct-83	ONI	MB	18	RE	8 5 P	76	Best of kind
	Red	Side	0.481	NOx		1.13	PROPYLENE
						0.44	ETHYLENE
						1.60	1-BUTENE
	Blue	Side	0.497	NOx		1.00	PROPYLENE
						0.88	ETHYLENE
						1.10	1-BUTENE

Number of runs of this HC type is 9

>>>>> For the HC type of 10 Single, olefins <<<<<

18-Oct-77	ONI	10	33	ET	6 0 S	92	Best of kind
	Red Side		0.494	NOx	3.84	ETHYLENE	
	Blue Side		0.484	NOx	1.88	ETHYLENE	
10-Jan-78	ONI	10	18	ET	8 0 S	100	Sastisfactory
	Red Side		0.462	NOx	3.25	PROPYLENE	
	Blue Side		0.482	NOx	4.35	ETHYLENE	
16-Jun-78	ONI	10	18	ET	7 0 S	91	Sastisfactory
	Red Side		0.634	NOx	3.95	ETHYLENE	
	Blue Side		0.640	NOx	2.00	PROPYLENE	
01-Jul-78	ONI	10	18	ET	7 0 S	83	Best of kind
	Red Side		0.887	NOx	1.45	ETHYLENE	
	Blue Side		0.935	NOx	1.51	PROPYLENE	
30-Jul-78	ONI	10	18	ET	6 0 S	82	Sastisfactory
	Red Side		0.474	NOx	1.32	ETHYLENE	
	Blue Side		0.483	NOx	1.25	PROPYLENE	
06-Aug-78	ONI	10	18	ET	4 0 S	57	Some Problems
	Red Side		0.561	NOx	1.69	PROPYLENE	
	Blue Side		0.569	NOx	2.59	ETHYLENE	
15-Aug-78	ONI	10	18	ET	7 0 S	87	Best of kind
	Red Side		0.543	NOx	1.45	PROPYLENE	
	Blue Side		0.563	NOx	1.58	ETHYLENE	
21-Aug-78	ONI	10	18	ET	9 0 S	100	Sastisfactory
	Red Side		0.978	NOx	1.39	ETHYLENE	
	Blue Side		0.981	NOx	1.28	PROPYLENE	
23-Aug-78	ONI	10	09	ET	8 0 S	97	Best of kind
	Red Side		0.507	NOx	2.93	ETHYLENE	
	Blue Side		0.521	NOx	2.97	ETHYLENE	
18-Oct-78	ONI	10	18	ET	9 0 S	83	Sastisfactory
	Red Side		0.456	NOx	3.12	ETHYLENE	
	Blue Side		0.456	NOx	1.52	PROPYLENE	

07-Nov-78	ONI	10	18	ET	5	0	S	64	Sastisfactory
	Red Side		0.441	NOx		1.37		PROPYLENE
	Blue Side		0.441	NOx		2.67		ETHYLENE
08-Jul-80	ONI	10	33	ET	5	0	S	32	Best of kind
	Red Side		0.754	NOx		2.10		ETHYLENE
	Blue Side		0.754	NOx		3.23		ETHYLENE
26-Aug-80	ONI	10	33	ET	8	0	S	100	Best of kind
	Red Side		0.472	NOx		1.99		ETHYLENE
	Blue Side		0.464	NOx		1.00		ETHYLENE
08-Jul-81	ONI	10	18	ET	7	4	L	76	Sastisfactory
	Red Side		0.500	NOx		2.00		ISOPRENE
	Blue Side		0.500	NOx		2.00		ETHYLENE
05-Oct-84	ONI	10	33	ET	9	5	U	70	Best of kind
	Red Side		0.365	NOx		3.13		ETHYLENE
	Blue Side		0.370	NOx		1.80		ETHYLENE
11-Oct-84	ONI	10	18	ET	9	5	U	80	Sastisfactory
	Red Side		0.351	NOx		2.86		ETHYLENE
	Blue Side		0.354	NOx		2.25		PROPYLENE
12-Oct-84	ONI	10	18	ET	9	5	U	80	Sastisfactory
	Red Side		0.710	NOx		2.68		ETHYLENE
	Blue Side		0.682	NOx		1.96		PROPYLENE
04-Jul-86	ONI	10	34	ET	5	0	P	0	Best of kind
	Red Side		0.150	NOx		1.77		ETHYLENE
	Blue Side		0.310	NOx		1.88		ETHYLENE
09-Jul-86	ONI	10	33	ET	5	0	P	0	Sastisfactory
	Red Side		0.350	NOx		1.00		ETHYLENE
	Blue Side		0.350	NOx		2.00		ETHYLENE
12-Jul-86	ONI	10	18	ET	5	0	U	0	Sastisfactory
	Red Side		0.350	NOx		2.00		ETHYLENE
	Blue Side		0.350	NOx		1.00		PROPYLENE

```
13-Jul-86 ONI 10  18     ET  5 0 P   0   Best of kind

          Red  Side   0.350  NOx    0.50    PROPYLENE
          Blue Side   0.350  NOx    1.00    ETHYLENE

Number of runs of this HC type is 21

>>>>> For the HC type of 1U Single, ole vs ald <<<<<

12-Nov-77 ONI 1U  18     ET  6 0 S   83   Sastisfactory

          Red  Side   0.503  NOx    2.00    ETHYLENE
          Blue Side   0.471  NOx    1.72    ACETALDEHYDE

19-Nov-77 ONI 1U  18     ET  7 0 S   99   Sastisfactory

          Red  Side   0.451  NOx    3.78    ETHYLENE
          Blue Side   0.832  NOx    3.82    ACETALDEHYDE

20-Nov-77 ONI 1U  18     ET  8 0 S   100  Sastisfactory

          Red  Side   0.446  NOx    4.37    ETHYLENE
          Blue Side   0.881  NOx    3.92    ACETALDEHYDE

15-Sep-78 ONI 1U  34     ET  3 0 S   53   Supporting

          Red  Side   0.526  NOx    2.98    ETHYLENE
          Blue Side   0.268  NOx    2.00    FORMALDEHYDE

19-Sep-78 ONI 1U  18     ET  7 0 S   91   Supporting

          Red  Side   0.688  NOx    1.88    ETHYLENE
          Blue Side   0.690  NOx    2.00    FORMALDEHYDE

21-Sep-78 ONI 1U  18     ET  5 0 S   90   Supporting

          Red  Side   0.258  NOx    1.93    ETHYLENE
          Blue Side   0.257  NOx    1.97    FORMALDEHYDE

04-Aug-79 ONI 1U  18     ET  6 0 S   59   Sastisfactory

          Red  Side   0.227  NOx    0.88    ETHYLENE
          Blue Side   0.230  NOx    0.56    FORMALDEHYDE

05-Aug-79 ONI 1U  18     ET  5 0 S   14   Best of kind

          Red  Side   0.638  NOx    4.11    ETHYLENE
          Blue Side   0.539  NOx    1.20    FORMALDEHYDE
```

16-Jul-86 ONI 1U .16 . ET 5 0 U 0 Sastisfactory

Red	Side	0.350	NOx	2.00	ETHYLENE
Blue	Side	0.350	NOx	1.00	FORMALDEHYDE

Number of runs of this HC type is 9