MOSA DEVELOPER'S GUIDE

by

**Jin Wang, Bradford G. Nickerson,
Ronald M. Lees**

TR96-109, April 1996

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

MOSAA Developer's Guide

by

Jin Wang
Bradford G.Nickerson
Ronald M. Lees

Faculty of Computer Science
Physics Department
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada
Phone: (506) 453-4566

Fax: (506)453-3566
Email: fcs@unb.ca

May 31, 1996

MOSAA Developer's Guide

Faculty of Computer Science Technical Report 96-109

Jin Wang
Bradford G. Nickerson
Ronald M. Lees

University of New Brunswick
Faculty of Computer Science
Physics Department
PO Box 4400
Fredericton, N.B., Canada
E3B 5A3

E-mail: e7qz@unb.ca (Jin Wang)
E-mail: bgn@unb.ca (Bradford G. Nickerson)
E-mail: lees@unb.ca (Ronald M. Lees)

May 1996

Abstract

MOSAA(MOlecular Spectroscopic Assignment Assistant) is a knowledge-based system that can assist physicists in spectroscopic assignment.

This guide presents the MOSAA system from a developer's view-point. It gives the rule grammar as well as descriptions of rule structure and components associated with it. The inference engine is briefly described using diagrams.

Some examples are provided to help the developer understand 'how' to modify or expand the rule base in simple cases. Also, the process of recompiling MOSAA is discussed. A pointer to the source code files is also provided.

Contents

List of Figures **III**

List of Tables **IV**

1 INTRODUCTION **1**

2 GRAMMAR **5**

2.1 Grammar in bison syntax 5

2.2 Parameters 8

2.3 MOSAA functions 14

2.4 Subroutines 17

2.5 Properties 17

2.6 Explanation 18

3 RULES **19**

3.1 Rule structure 19

3.1.1 Rule categories 19

3.1.2 Conditions 20

3.1.3 Conclusion-conditions 23

3.2 An example of modifying the rule base 25

4 INFERENCE ENGINE **31**

4.1 Three types of rules 31
4.2 Special properties 33

5 RECOMPILING MOSAA 40

5.1 Source files 41
5.2 User input files 41
5.3 Command line version 44
5.3.1 Source files 44
5.3.2 Compiling files 45
5.3.3 Debugging 46
5.4 xview mode 47
5.4.1 xview version source files 47
5.4.2 xview programming 48
5.5 Auxiliary files 49

6 Conclusions 51

A lex file 54

B Some input files 63

B.1 Rule Summary 63
B.2 _parm file 65
B.3 _sub file 76
B.4 _mosaa file 81
B.5 _prop file 89
B.6 parm.const file 90

C Inference Engine Diagrams 92

D Output file 103

List of Figures

1.1 MOSAA top level user's perspective. 2

1.2 MOSAA top level developer's perspective. 4

2.1 MOSAA rule grammar in bison syntax notation. 6

2.2 A sample used to explain Parm `Cur_R_Peaks[14].wv`. 11

2.3 parameter file (`_.parm`) grammar. 13

2.4 Some example parameter definitions from parm file `mar.parm`. 14

2.5 Grammar of MOSAA functions in file `_.mosaa`. 15

3.1 Rule structure (also see Fig 2.1 line '9'). 20

3.2 A sample consequence rule. 21

3.3 A sample MOSAA-function rule. 22

3.4 An antecedent rule: A_R60. 25

3.5 A piece of spreadsheet produced during the inference engine running. 26

3.6 A piece of spectrum. 28

3.7 A piece of spreadsheet produced during the inference engine running, after adjusting the intensity of one line. 29

3.8 The C++ code added for adding a new mosaafunction '`equal`'. 30

4.1 A sample goal rule. 32

4.2 A sample of using property "`Relative ´A´, x`". 34

4.3 An example of using property "`Relative ´C´,x`". 36

4.4 A sample of using property "TRY". 37
4.5 A group of hypothetical rules used to illustrate the problem of mixing the use of 'Relative' and 'TRY'. 38
4.6 Simple deduction paths for the rules of Fig 4.5. 39

5.1 Two versions of MOSAA. 40
5.2 A portion of a combination difference file (`fe24x6.GD`). 42
5.3 A portion of a peakfinder file (`fe24x6.pkf`). 43
5.4 A portion of a spectrum file (`fe24x6.tra`). 44
5.5 Auxiliary output file `mosaa.idx`. 49

C.1 `mosaa` Toplevel. 93
C.2 Inference engine. 94
C.3 Monitor mechanism. 95
C.4 Findout mechanism. 96
C.5 Simple findout machanism. 97
C.6 Procedure 'is_right'-determines if the premise of a consequence rule is right. 98
C.7 Procedure 'ante_is_right'-determines if the premise of an antecedent rule is right. 99
C.8 Do conclusion procedure- 'doconclu'. 100
C.9 Simple monitor machanism. 101
C.10 Antecedent rule invoking procedure-'ante_invoke'. 102

List of Tables

2.1 Symbols used in defining MOSAA grammars. 9

3.1 The three types of loops. 24

B.1 Numbers of different rule types 63

Chapter 1

INTRODUCTION

Studying the energy level structure of different molecules by means of spectroscopy is one of the fields in Physics and Chemistry. Presently, the group of Dr. R.M.Lees in the Physics Dept. in the University of New Brunswick is focusing on studying the rotational and vibrational energy levels of methanol and its isotopic species [3, 4, 7, 9, 10]. The method currently used involves first obtaining an infrared or far-infrared absorption spectrum of the molecule being studied, which shows the transitions between the various energy levels of the molecule. The wavenumber of the useful peaks of the absorption spectrum and their corresponding intensities are stored in a Peak Finder file. Then, using the current knowledge about energy level structures as well as a variety of analytic techniques, the peaks in the peakfinder file are labeled with the appropriate quantum number transitions. This leads to an understanding of the energy level structure and the internal interactions of the molecules.

MOSAA[1] is a knowledge-based system which can assist researchers in the assignment of the peaks of the molecules in question by using the spectral information provided and a knowledge base containing known energy levels of the given molecules and the rules provided by the experts for manipulating the information.

As shown in the top level architecture of Fig 1.1, MOSAA has two kinds of user

[1]MOlecular Spectroscopic Assignment Assistant

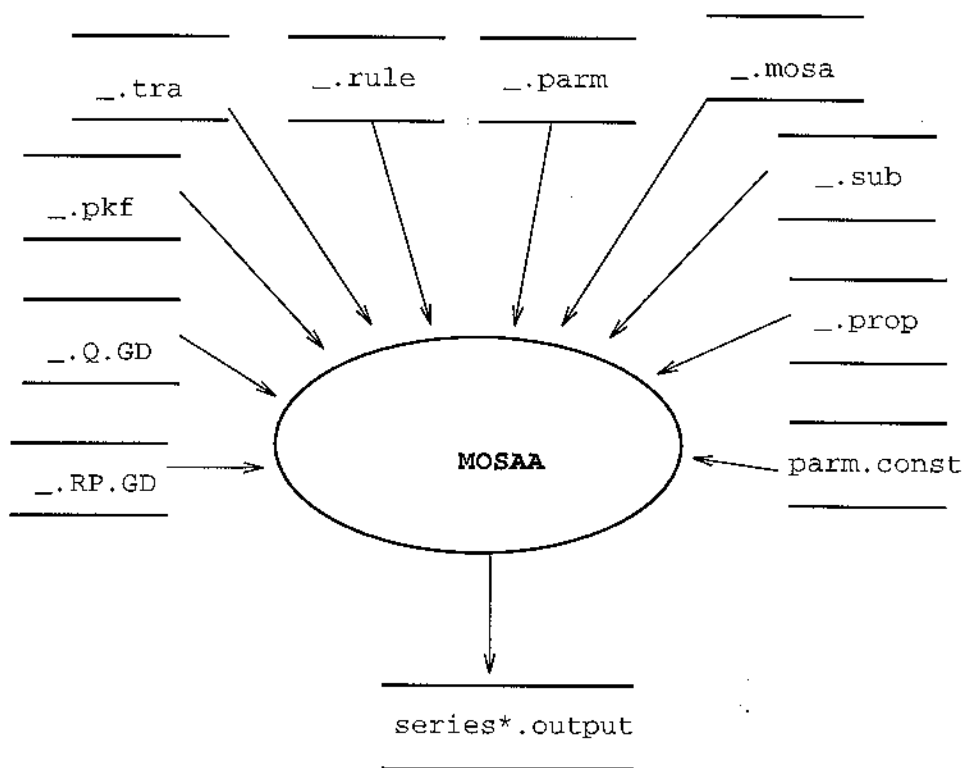

Figure 1.1: MOSAA top level user's perspective.

input files:

- Spectroscopy files -

  _.tra Spectrum plot file, used for graphically displaying the spectrum.

  _.pkf Peak finder file, data for assignment manipulation, also used for graphically displaying the "stick" spectrum.

  _.Q.GD calculated R-Q combination difference file.

  _.RP.GD calculated R-P combination difference file.

  Section 5.2 gives more details about these input files.

- Knowledge_base files - (Appendix B)

  _.rule contains all the rules in different groups.

_.parm contains the definitions of all the parameters used in the rules.

_.mosaa contains the definitions of all the MOSAA functions used in the rules.

_.sub contains the definitions of all the subroutines used in the rules.

_.prop contains the definitions of all the properties used in the rules

parm.const constant parameters' values.

The output is a group of files (e.g. Series0.output) which records the assigned peak information. Appendix D gives an example of such output files.

The conventions of file names here are: 1) a leading '_' character means the user provides one name for each file type; 2) an '*' means one or more names.

The MOSAA system architecture from a developer's perspective is shown in Fig 1.2.

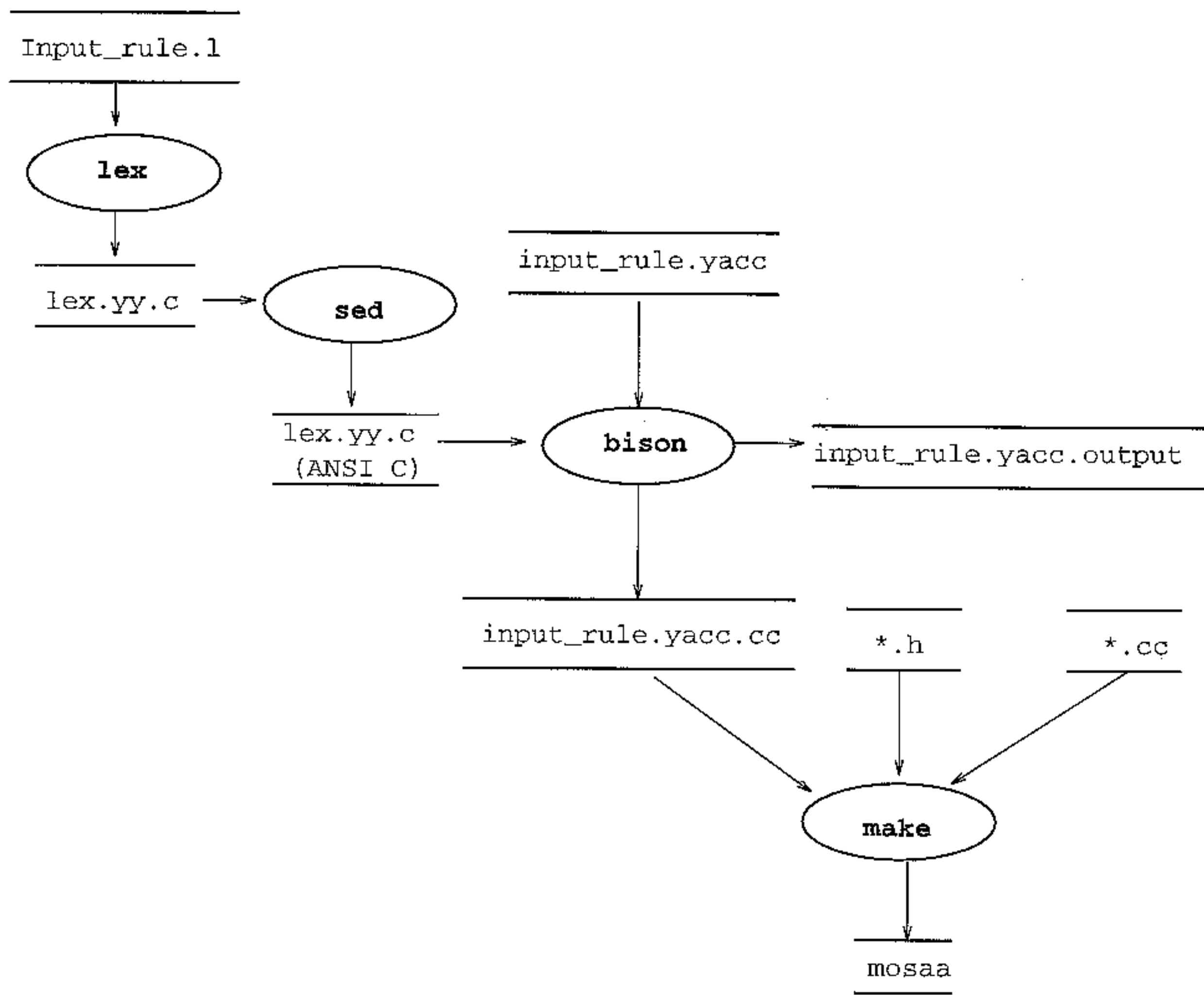

Figure 1.2: MOSAA top level developer's perspective.

Chapter 2

GRAMMAR

Rules in the MOSAA KBS have the normal **"IF-THEN"** structure with extra **Property** and **Explanation** parts. **Parameters**, **MOSAA_functions** and **Subroutines** are three basic components associated with rules. Section 3.1 gives more details about the rule structure.

2.1 Grammar in bison syntax

The compiler used to read in the rules is produced by lexical analyzer *lex* and parser generator *bison* [2, 5].

MOSAA rules' lexical convention is not given here; Appendix A gives the complete lex file `input_rule.l` which is used to produce the lexical analyzer.

Fig 2.1 gives the grammar of MOSAA rules in the syntax notation of *bison*, which is *essentially a machine-readable* Backus-Naur format(BNF) [2].

This figure is abstracted from the auxiliary output file `y.output` of

```
yacc -v input_rule.yacc
```

The reason that we use the auxiliary output file of `yacc` instead of the `input_rule.yacc.output` from `bison` is that the syntax notation in `y.output` is

```
Grammar
  0  $accept : begin $end
  1  begin : start
  2  start : rule_list
  3        | start rule_list
  4  rule_list : '%' CR c_ruleset
  5            | '%' GR ag_ruleset
  6            | '%' AR ag_ruleset
  7  c_ruleset : c_rule
  8            | c_ruleset c_rule
  9  c_rule : cr_ruleid IF premise THEN conclusion property comment
 10  cr_ruleid : CR
 11  premise : condition
 12          | premise '&' condition
 13  condition : loop
 14            | parm
 15            | subroutine
 16            | mosa_func
 17  loop : loop_type loop_var '=' loop_start ';'
                          loop_end loop_step '{' loop_body '}'
 18  loop_var : LOOPI
 19           | LOOPD
 20  loop_type : FOR
 21            | ANYIF
 22            | DO
 23  loop_start : exp
 24  loop_end : exp
 25  loop_step :
 26            | ';' exp
 27  loop_body : condition
 28            | loop_body '&' condition
 29  parm : parm_name
 30       | parm_name '[' index ']'
 31       | parm_name '.' field
 32       | parm_name '[' index ']' '.' field
 33  index : INTEGER
 34        | LOOPI
 35        | parm
 36  parm_name : IDENTIFIER
 37  field : IDENTIFIER
```

Figure 2.1: MOSAA rule grammar in bison syntax notation.

```
38  subroutine : ´$´ name ´(´ arglist ´)´
39             | ´$´ name ´(´ ´)´
40  mosa_func : mosa_name arglist
41            | mosa_name
42  mosa_name : MOSA_NAME
43            | ´=´
44  conclusion : conclu_condition
45             | conclusion ´&´ conclu_condition
46  conclu_condition : conclu_loop
47                   | subroutine
48                   | mosa_func
49  conclu_loop : loop_type loop_var ´=´ loop_start ´;´
              loop_end loop_step ´{´ conclu_loop_body ´}´
50  conclu_loop_body : conclu_condition
51                   | conclu_loop_body ´&´ conclu_condition
52  property :
53           | ´#´ prop
54  prop : prop_item
55       | prop ´&´ prop_item
56  prop_item : name
57            | name arglist
58            | name ´{´ conclusion ´}´
59  name : IDENTIFIER
60  comment :
61          | COMMENT
62  arglist : arg
63          | arglist ´,´ arg
64  arg : exp
65      | ´(´ arg_func ´)´
66  arg_func : mini_mosa_func
67           | mini_subroutine
68  mini_mosa_func : mosa_name explist
69                 | mosa_name
70  mini_subroutine : ´$´ name ´(´ explist ´)´
71                  | ´$´ name ´(´ ´)´
72  explist : exp
73          | explist ´,´ exp
```

Fig 2.1 (continued)

```
74  exp : single_value
75      | LOOPI
76      | LOOPD
77      | parm
78      | exp '+' exp
79      | exp '-' exp
80      | exp '*' exp
81      | exp '/' exp
82      | '-' exp
83      | '(' exp ')'
84  single_value : INTEGER
85               | DOUBLE
86               | CHAR
87               | STRING
88  ag_ruleset : ag_rule
89             | ag_ruleset ag_rule
90  ag_rule : ag_ruleid IF premise THEN conclusion property comment
91  ag_ruleid : GR
92            | AR
```

Fig 2.1 (continued)

closer to the context free notation of a rule grammar. Since both `input_rule.yacc.output` from `bison` and `y.output` from `yacc` are obtained from the grammar of `input_rule.yacc`, so there is no difference except the notation used.

The details of BNF and the grammar of *bison* are not given here; please refer to [1] for the description of BNF, and [2, 5] for further information about *bison*.

The basic symbols used in Fig 2.1 are listed in Table 2.1. The last four rows in table are not used in the bison grammar; they are used in defining parameters and functions (see Fig 2.3 and Fig 2.5).

2.2 Parameters

Parameters are one of the most important concepts in the MOSAA system. A parameter is a structure that identifies or contains a piece of information that the inference

Table 2.1: Symbols used in defining MOSAA grammars.

<table>
<tr><th>Symbol</th><th>Meaning</th></tr>
<tr><td>:</td><td>is defined to be</td></tr>
<tr><td>|</td><td>alternatively</td></tr>
<tr><td>lowercase words
(e.g. <code>premise</code>, <code>rulelist</code>)</td><td>nonterminal</td></tr>
<tr><td>uppercase words
(e.g. <code>CR</code>, <code>IDENTIFIER</code>)</td><td rowspan="2">both are terminal</td></tr>
<tr><td>'single_char'
(e.g. ´=´, ´%´, ´+´)</td></tr>
<tr><td>*</td><td>The preceding syntactic unit can be repeated zero or more times</td></tr>
<tr><td>+</td><td>The preceding syntactic unit can be repeated one or more times</td></tr>
<tr><td>{}</td><td>The enclosed syntactic units are grouped as a single syntactic unit</td></tr>
<tr><td>[]</td><td>The enclosed syntactic unit is optional, may occur zero or one time</td></tr>
</table>

engine uses to arrive at a conclusion [6].

Parameters are the basic components of the conditions of the premise in a rule. They can even be used alone as one complete condition of a premise. Parameters have special roles in the MOSAA inference engine (see chapter 4).

Several ways can be used to classify the parameters:

1. appearance in the rules -

   Simple Parm -

   Plain identifier; e.g.

   `S_idx ,Pro_L, Fir_L_S_idx, TEMPI1`

   `S_idx` is a temporary parameter, but it usually is used to refer to a line's index in the array of a group of lines which belong to one series branch.

   `Pro_L` usually shows the current problem line's S_idx.

   `Fir_L_S_idx` refers to the first line of "three-lines" in the current series.

   `TEMPI1` is a temporary parameter without special spectroscopic meaning.

   Index Parm -

   Plain identifier associated with an extra index; e.g.

Cur_CD[0], Cur_CD[8] ...

We haven't used any of this kind of parameter yet (even Cur_CD[0] is just shown here as example). Since the rule base is under construction, this parameter type is kept for later convenience.

Field Parm -

Plain identifier associated with fields; e.g.

Cur_Series.K, Cur_Series.Symm ...

Cur_Series is a parm with several fields, it records the current series' properties such as 'K' value, 'Symm' value. Since 'K' is one of the properties, it becomes one of the fields of this parameter which appears as Cur_Series.K.

These two parameters have the same fields, so in the parm file, they are defined together using the same 'Cur_Series' with different fields.

Index_Field Parm -

Identifier associated with index and fields; e.g.

Cur_R_Peaks[14].wv, Cur_P_Peaks[0].intens ...

Each series has 'R', 'P', and 'Q' branches. Each branch has several lines, where each line has several properties such as wavenumber and intensity. Cur_R_Peaks[14].wv records the wavenumber of the line whose index in array Cur_R_Peaks is 14. It should be noticed that, 14 here doesn't mean that the line is the **14th** line of this branch. The current top line's index of this R branch is in Cur_R_Series.top. Comparing with Cur_R_Series.top, we can know the real position of this line in the branch. This is like a double pointer in computer data structures. Fig 2.2 shows that when Cur_R_Series.top equals 9, Cur_R_Peaks[14].wv is the wavenumber of the sixth line in the R branch we have found up to this point during the searching process.

Index	J	wv	delta1	delta2	intens	comments
9.		1019.771244	1.391835		0.538795	Cur_R_Series.top
10.		1021.163079	1.374718	-0.017117	0.463053	
11.		1022.537797	1.357109	-0.017609	0.391154	
12.		1023.894906	1.339657	-0.017452	0.384100	
13.		1025.234563	1.322407	-0.017250	0.388059	
14.		1026.556970	1.304775	-0.017632	0.385666	Cur_R_Peaks[14]
15.		1027.861745	1.287316	-0.017459	0.383996	
16.		1029.149061	1.269753	-0.017563	0.391489	
17.		1030.418814	1.252379	-0.017374	0.412323	Cur_R_Series.bottom
18.		1031.671193	1.234256	-0.018123	0.292232	

Figure 2.2: A sample used to explain Parm `Cur_R_Peaks[14].wv`.

2. actual data types -

   Each parameter has one of the following data types:

   i-*int*, **d**-*double*, c-*char*, s-*string*, k-*stack*.

   All the parameters with the same name, but different indices, have the same type, while different fields can have different types. "**stack**" is a special data type in MOSAA; it appears in the rule structure as a *Simple Parm*, while the inference engine, MOSAA functions and subroutines treat it as a complex parameter, such as *Index_Field Parm*. For example:

```
CD_Candidate_Stack nindex 0 k nques
CD_Candidate_Stack_Num  nindex 0 i nques
```

   `CD_Candidate_Stack` is a stack parameter; each stack parameter always has a `~_Num` parameter associated with it. For `CD_Candidate_Stack`, it has `CD_Candidate_Stack_Num` to show the number of items in the stack. In rules, normally we can only find `CD_Candidate_Stack`, not `CD_Candidate_Stack_Num`; e.g.

```
IF  .....
   & Pop_CD CD_Candidate_Stack
    ....
THEN
    ....

IF  ....
THEN
    RESERVE CD_Candidate_Stack
```

where the real data are in CD_Candidate_Stack.K, CD_Candidate_Stack.Symm, CD_Candidate_Stack.n, CD_Candidate_Stack.T.

3. "askable" property

   If a parameter has the "askable" property, the value of this parameter can be obtained from the user's answer by proper prompting during running of the MOSAA inference engine.

4. others -

   Most parameters have specific meanings related to spectroscopic knowledge or the process of performing spectroscopic assignments. However, some of the parameters are just used as "temp" variables, such as "*TEMPD3*" for reserving a double value temporarily. The scope of these "temp" parameters is just within the current rule.

   Most parameters' values are obtained from deduction or feedback from the user if they have the "askable" property. Some of the parameter's values are set during preprocessing, from a special file parm.const (Appendix B.6).

All the parameters must be defined in the file _.parm (Appendix B.2), which gives sufficient information for later rule compiling and the running of the inference engine.

The grammar defining a parameter in file _.parm is in Fig 2.3.

```
parm_definition:    parm_name  HAS_INDEX  fields_num  fields
                  | parm_name  HAS_INDEX   0  DATA_TYPE ASKABLE
parm_name:    IDENTIFIER
fields_num:    NUMBER
fields:       { field_name DATA_TYPE ASKABLE}+

HAS_INDEX:    'y'
            | 'n'
DATA_TYPE:    'i' | 'd' | 'c' | 's' | 'k'
ASKABLE:      'y'
            | 'n'
NUMBER:     {[1-9]}{[0-9]}*
```

Figure 2.3: parameter file (_.parm) grammar.

There are five DATA_TYPE here, where:

'i' - *int*, 'd'-*double*, 'c'-*char*, 's'-*string*, 'k'-*stack*

HAS_INDEX shows whether this parameter has an index or not. If the fields_num is 0, the following parts are the data type of this parameter, and 'y' or 'n' to show whether this parameter is askable or not. If the fields_num is not 0, the following parts give each field's information; (1) the name of the field, (2) the data type of the parameter with this field, and (3) whether this parameter with field is askable. Attention should be paid to the fact that the definition of each parameter must be in a single line, which means no nl (new line) character can be encountered before the end of this parameter's definition.

Fig 2.4 shows some examples of parameter definitions, which are abstracted from the file mar.parm.

For the mosaa program, what it really reads from HAS_INDEX is just the first character, 'y' or 'n'. So 'nindex' shown in figure 2.4 is the same as 'n'. The same situation applies to DATA_TYPE and ASKABLE.

```
Fir_L_S_idx  nindex 0 i yques
Cur_CD  yindex   0  d nques
Cur_Series  nindex  7 branch c yques K i nques n i nques  t i nques
                      prop  c  yques series_no i nques Symm s nques

Cur_Peaks  yindex  10  J i nques wv d nques intens d nques
      pkf_idx i yques delta1 d nques delta2 d nques series_no i nques
      bias d nques zone_no i nques confirm i nques
```

Figure 2.4: Some example parameter definitions from parm file `mar.parm`.

Parameter `Fir_L_S_idx` is a *Simple Parm.* In Fig 2.4, `nindex` shows that it doesn't have any index, and `0` shows that there are no fields with it. Thus the following two terms in the line are data type `i` and askable property `yques`.

Parameter `Cur_CD` is an *Index Parm.* The only difference between the definition of `Cur_CD` and `Fir_L_S_idx` is the `HAS_INDEX` term.

Parameter `Cur_Series` is a *Field Parm.* In Fig 2.4, `field_num` is positive 7, which indicates that it has 7 fields. Since this kind of parameter can't exist without its fields, there is no data type for `Cur_Series`. Instead, each field has a data type and askable property with it such as `Cur_Series.K`; the data type is 'i' and it is nonaskable.

Parameter `Cur_Peaks` has the similar situation with `Cur_Series`, with `HAS_INDEX` being different.

2.3 MOSAA functions

MOSAA functions can be classified into two groups:

normal MOSAA functions -

This kind of MOSAA function is almost identical to subroutines (see section 2.4 with the exception that it has a different appearance in the rule structure.

MOSAA-function rule functions -

These typical functions are not written as normal C++ functions; they are actual rules in the rule base. However, they are used (being called) as normal MOSAA functions.

Section 3.1.2 gives more details about how these MOSAA functions are used in the rule structure.

The description of all the MOSAA functions which appear in rules must be contained in the file `_.mosaa`. The following grammar in Fig 2.5 is used to define MOSAA functions in the `_.mosaa` file.

```
MOSAAfunc_definition : MOSAAfunc_name  RETURN_TYPE {arg_type}*
MOSAAfunc_name:  IDENTIFIER
arg_type:     VALUE_TYPE
   |  PARM_NAME_TYPE  MODIFY_TYPE

RETURN_TYPE:  'i' | 'd' | 'c' | 's'
VALUE_TYPE:   'i' | 'd' | 'c' | 's'
PARM_NAME_TYPE:  't' | 'e' | 'r' | 'g' | 'k'
MODIFY_TYPE:  'm' | 'r' | 'u'
```

Figure 2.5: Grammar of MOSAA functions in file `_.mosaa`.

`MOSAAfunc_name` usually is all lowercase characters.

If the `arg_type` is `VALUE_TYPE`, what the MOSAA function needs is a value. Thus in the rule, this argument can be a const, or a parameter. The `VALUE_TYPE` can be:

'i' - *int*, 'd'-*double*, 'c'-*char*, 's'-*string*.

If the `arg_type` is `PARM_NAME_TYPE`, what the MOSAA function needs is the name of a parameter. Since each parameter has its data type, there are six `PARM_NAME_TYPE`, as follows:

't'-*int*, 'e'-*double*, 'r'-*char*, 'g'-*string*, 'p'-*any kind of parm* , 'k'-*stack*

which means that the argument must be a parm name instead of a parm value. k always appears as a parm name.

Once this special PARM_NAME_TYPE argument type appears in the _.mosaa file, there must be a MODIFY type following it, which can be:

'm'-*modifying*, 'r'-*reasoning*, 'u'-*nothing* .

The *modify* type is used to tell the inference engine that after doing this subroutine, the parameter will be assigned a value('r'), be modified('m'), or not affected('u'). This property is very important in controlling later inference engine running (see chapter 4).

Here we use an example to show how to put a MOSAA function definition in the _.mosaa file.

```
= int_return  tparm reasoning  i_value
  %assign an int value 'i_value' to int parm 'tparm'
```

This is an 'assign' function. The line after '%' are the comments. The return type of this function is *int* and the first argument must be a name of an *int* parameter, so we use type 't'. Since it is a PARM_NAME_TYPE type, its MODIFY type - 'reasoning' follows it. The second argument is an integer value, so 'i_value' appears there. As the comment shows, this MOSAA function assigns an integer value to an *int* parameter. After this assignment, the parameter's value is determined. When it is read in from later mosaa program, the assign function is the same as :

```
= i t r i
```

The initial MOSAA function definition

```
= int_return  tparm reasoning  i_value
```

makes the file more readable compared to

```
= i t r i
```

To overload a function, we just give another definition. For example, if we want another '=' function to do double value assignment, we define the assign function as follows:

```
= int_return  eparm reasoning  d_value
        %assign a double value 'd_value' to double parm 'eparm'
```

2.4 Subroutines

Subroutines are normal functions written in C++; they are one of the components of rules. The descriptions of all the subroutines which can be used in rules are contained in file _.sub (see Appendix B.3) . These descriptions give the return type and types of all the arguments for the subroutines, which are in the same format as MOSAA functions. A simple subroutine definition example is:

```
show_peak int int
  %graphically display the new found peak of the current series
```

2.5 Properties

Properties are associated with rules, and they give some additional control to the running of the inference engine, or make it easier to write or read some rules.

Here is a brief description of property "**MATCH**". When a rule has a property such as:

```
MATCH L4, S_idx
```

then, besides the parameter L4 being assigned the value S_idx, the other six L parameters-

```
L0, L1, L2, L3, L5, L6
```

will automatically be assigned to

```
S_idx-4, S_idx-3, S_idx-2, S_idx-1, S_idx+1, S_idx+2
```

respectively. This makes writing current rules easier, and makes them more readable. Some other properties are very important in running the inference engine, and section 4.2 gives further details.

2.6 Explanation

The explanation part of a rule gives a brief English description of what this rule does. It explains the rule from the molecular assignment view point, and is also used by the "Explanation Facility" during running of the inference engine.

Chapter 3

RULES

The basic component of the MOSAA knowledge base is the rule base. Rules in MOSAA are written in plain ASCII characters and contained in file `_.rule` . Before the inference engine starts, this file is read in, and the rules are compiled into an internal rule structure. To add, remove or modify a rule, the ASCII `_.rule` file is edited appropriately.

3.1 Rule structure

As mentioned in chapter 2, rules in the MOSAA KBS have a normal **"IF-THEN"** structure with extra **Property** and **Explanation** parts. Fig 3.1 shows the MOSAA rule structure.

Fig 3.2 and Fig 3.3 are two sample rules from the rule base.

3.1.1 Rule categories

"*label*" in fig 3.1 contains the rule_id and the group this rule lies in. MOSAA rules are categorized into three groups: *G-Goal* , *C-Consequence* and *A-Antecedent* rules.

There are no major structure differences between these three types of rules, but

```
label   IF      condition_1
                condition_2
                ...                    Premise
                condition_n

        THEN    conclu_1
                conclu_2
                ...                    Conclusion
                conclu_n

        #   prop_1
            prop_2
            ...                        Property
            prop_n

        /*  .........                  Explanation
        */
```

Figure 3.1: Rule structure (also see Fig 2.1 line '9').

in the inference engine, different kinds of rules act very differently. Section 4.1 gives a brief description about the role of each rule type in the inference engine.

The compiler won't take care of *rule_id*, so the rule_id can be any integer number, as long as it is in "int" range.

3.1.2 Conditions

"*condition*" in the Premise can be one of the following clauses:

single parameter -

If the parameter's value is known and $\neq 0$, this condition becomes true.

MOSAA function -

If this MOSAA function returns a value $\neq 0$, this condition becomes true.

```
C_R38  IF   <  Cur_R_Series.top,Cur_R_Series.top_of_three
                                                    /%condition_1%/
     &  FOR i= Cur_R_Series.top_of_three-1;
                   Cur_R_Series.bottom_of_three /%condition_2%/
          {
              Transfer_Line_R_To_P  i         /%loop_clause_1%/
            & Transfer_Line_R_To_Q i          /%loop_clause_2%/
            & UNFOUND R_To_Q_Pro_L            /%loop_clause_3%/
          }
     &  FOR i=Cur_R_Series.top_of_three;
                   Cur_R_Series.bottom_of_three  /%condition_3%/
          {
              RPQ_Confirm i
          }
       THEN
            =  Transfer_TJ_To_P, 1                        /%conclu_1%/
       /*
           IF    there are already four R branch  lines which includes
                 three_lines and one line above the top of the
                 three_lines
             AND
                 from these four R branch lines, can find corresponding
                 P and Q lines
             AND
                 using these R, P,Q branch lines, we have confirmed
                 three_lines
           THEN
                 Transfer three_lines to  P has been done
       */
```

Figure 3.2: A sample consequence rule.

```
C_R250 IF   UNFOUND Stop_Transfer
         &  UNFOUND Cur_Series.K
         &  == Search_Dire, 'P'
         &  Set Pkf_idx1, ( $get_P_from_R(Cur_R_Peaks[S_idx].wv,
                Cur_R_Peaks[S_idx].J))  /%condition_4%/
         &  Pkf_idx1
       THEN
            Set TEMPI1, S_idx+2
         &  Set Cur_P_Peaks[TEMPI1].pkf_idx, Pkf_idx1
         &  Set Cur_P_Peaks[TEMPI1].J, Cur_R_Peaks[S_idx].J+2
         &  = Cur_Found_P_Line, TEMPI1
         &  Transfer_Line_R_To_P  S_idx
       /* IF    Transfer has not been stopped
            AND Current series' K value is already known
            AND Searching from R branch to P branch
            AND calling subroutine 'get_P_from_R' has found the
                corresponding P branch peak for R branch peak S_idx
          THEN
                this MOSAA-func rule is true (Transfer_Line_R_To_P S_idx).
            AND
                record corresponding information for this current found
                P branch peak.  Since Cur_Found_Line_P_Line value is
                determined by '=' instead of 'Set', it may invoke antecedent
                rules to do further processing
       */
```

Figure 3.3: A sample MOSAA-function rule.

"*condition_1*" in fig 3.2 calls a MOSAA function condition. "<" is the name of the function, and the following two symbols `Cur_R_Series.top` and `Cur_R_Series.top_of_three` are two arguments of this function. The argument can be a single parameter, an expression, or even a bracketed MOSAA function or subroutine.

"*loop_clause_1*" calls a special kind of MOSAA function, which in fact is a *MOSAA- function rule* function. This kind of MOSAA function is not a C++ function; instead it is a consequence rule and can only be a consequence rule.

Fig 3.3 shows this *MOSAA-function rule.* We can see that for the caller, "*loop_clause_1*" in CR_38 *condition_2*, there is no difference in calling a *normal* MOSAA function, or a *MOSAA-function rule* function.

subroutine -

If this subroutine returns a value $\neq 0$, this condition becomes true.

A subroutine condition is similar to a normal MOSAA function condition, with a trival difference in format.

loop -

Loop is a special case of condition. Each loop has a head part and a body part. The head part shows the loop type and sets the start value, end value and step value of the loop variables. The body part is a group of clauses, which can be one of the three previously introduced conditions. When all the clauses in the loop body are true, this loop body becomes true.

There are three kinds of loops, and Table 3.1 gives a brief description of them.

3.1.3 Conclusion-conditions

conclus in Conclusion can be one of the following clauses:

Table 3.1: The three types of loops.

loop	condition value
FOR	During the loop, anytime the loop body fails the condition fails
ANYIF	During the loop, anytime the loop body succeeds the condition becomes true
DO	Just execute the clauses in the loop body; the condition is always set to be true

MOSAA function -

If the MOSAA function is a normal one, simply call it to do the corresponding process. If it is a *MOSAA-function rule* function name, which means the current rule itself is a *MOSAA-function rule* being called, then do nothing.

For example, rule `C_R250` in Fig 3.3 is a *MOSAA-function rule.* Conclusion condition `Transfer_Line_R_To_P S_idx` shows this is a *MOSAA-function rule.* This rule is tried when the MOSAA function name appears in some other rules, such as in `C_R38`. When this rule is fired, it is as if MOSAA function is called. Therefore, all the other conditions in the conclusion are done, except the *MOSAA-function rule* function condition.

subroutine -

Call this subroutine.

loop -

Do this loop (can only be "**FOR** " or "**DO**").

All the parameters appearing in the conclusion must already be known; otherwise an error message is provided.

```
A_R60   IF    !Cur_Found_Line
          &  == Search_Dire, 'U'
          &  ANYOF i = L4; L6
             {
               Adjust_intens i
             }
        THEN
             None_Operation
        #  MATCH L4, Top_L_S_idx
         & Relative 'C', 21
        /*
           IF    during up searching, can't find more lines
             AND one of the top three lines' intensities can be adjusted
           THEN
                 restart consequence rule 21 to redo up searching from the
                 beginning
        */
```

Figure 3.4: An antecedent rule: A_R60.

3.2 An example of modifying the rule base

The MOSAA system is used for assisting in making molecular assignments. The current rule base contains a set of rules which can be used for a basic and simple series assignments. More and more rules can be added to the rule base to make the system more powerful. The following is an example of how to add a new rule to the rule base.

Antecedent rule A_R60(Fig 3.4) is a rule used to help in dealing with the situation when the up searching line process is stuck. At first we remove this rule and the associated rules C_R234 to C_R239 from the rule base, assuming that we don't yet have these rules (all the rules we mention here are referenced in Appendix B.1).

C_R23 is a rule trying to search upward for a next line. If this rule fails, C_R100 is going to be fired and parameter *Cur_Found_Line* is assigned the value 0. One of the reasons that it can't find the next line is because of the wrong intensity search range

which was set by the previous three lines. If one of the three lines is overlapped, the intensity range will be wrong, and the next line can't be found. So A_R60 is added to the rule base.

The last line we got is *Top_L_S_idx*, and we want to see if one of the three top lines *L4*, *L5* and *L6* is overlapped and its intensity can be adjusted. So we need a function; in this case, a *MOSAA-function rule* will be proper. Since there are all kinds of situations of three lines, we produced a group of consequence rules C_R234 to C_R239 to handle the different cases.

We use one example to show how the rules we added help dealing with the overlap case.

```
SpreadSheet is:
R,
Index   J        wv           delta1       delta2      intens
12.             1018.445679   1.410012                 0.466641
13.             1019.855691   1.393535   -0.016477    0.401719
14.             1021.249226   1.376004   -0.017531    0.299412
15.             1022.625230   1.358654   -0.017350    0.355367
16.             1023.983884   1.341201   -0.017453    0.337332
```

Figure 3.5: A piece of spreadsheet produced during the inference engine running.

Now, we are in the up searching process, and we have found lines up to `1018.445679`; the index of this line in the `Cur_Peaks` array is `Top_L_S_idx` since it is the top line we have got so far. The corresponding spreadsheet is shown in Fig 3.5. The intensity searching range obtained from `C_R23` is [0.189257, 0.589257]. In this range we couldn't find any line. Thus the inference is stuck there, and `C_R23` fails. `C_R100` then is fired, which invokes `A_R60` trying to check if one of the three top lines is overlapped.

During searching if the previous lines are overlapped, `C_R234` is going to be fired, which means line `1021.249226` probably is overlapped according to its strange intensity comparing to its neighbours. Fig 3.6 shows a piece of the spectrum around these

lines.

We can see that line `1021.249226` is kind of strong, although it is not 'fat' like some other overlapped lines. This illusrates the case where there can be one small line underneath which has almost the same wavenumber.

When the user confirms this overlap and wishes to adjust its intensity temporarily, `C_R234` is fired; it adjusts the intensity of line `1021.249226` from the initial value `0.299412` to `0.378543` according to its neighbouring line intensity values.

Therefore rule `A_R60` is fired; it reserves the intensity adjustment of line `1021.249226` and restarts the inference engine from rule `C_R22` to redo the upsearching. This time, one more line `1017.018113` is found, and we get the spreadsheet as shown in Fig 3.7.

There is a new MOSAA function "**Adjust_intens**", and the line

```
Adjust_intens  int  int
```

must be added into the `_.mosaa` file. Also, in the C++ file `func_table.cc`, the following code must be included :

```
case 42: return 0;
```

where "42" is the index of the function "Adjust_intens" in func_table (one way to get the index is described in section 5.5).

For this case, there is just one line "`return 0`" since it is a *MOSAA-function rule.*

For *normal MOSAA functions*, a real C++ function must be built in `mosaafunc.h`, `mosaafunc.cc`, and the associated code should be included into `func_table.cc`. This is the same process as that required for adding a new subroutine. For example, to add a new MOSAA function: "=="(for int values), then one would add the lines

```
==  int  i_value1  i_value2
 % if i_value1 is equal to i_value2, return 1, else return 0.
```

in the `_.mosaa` file, and the lines of source

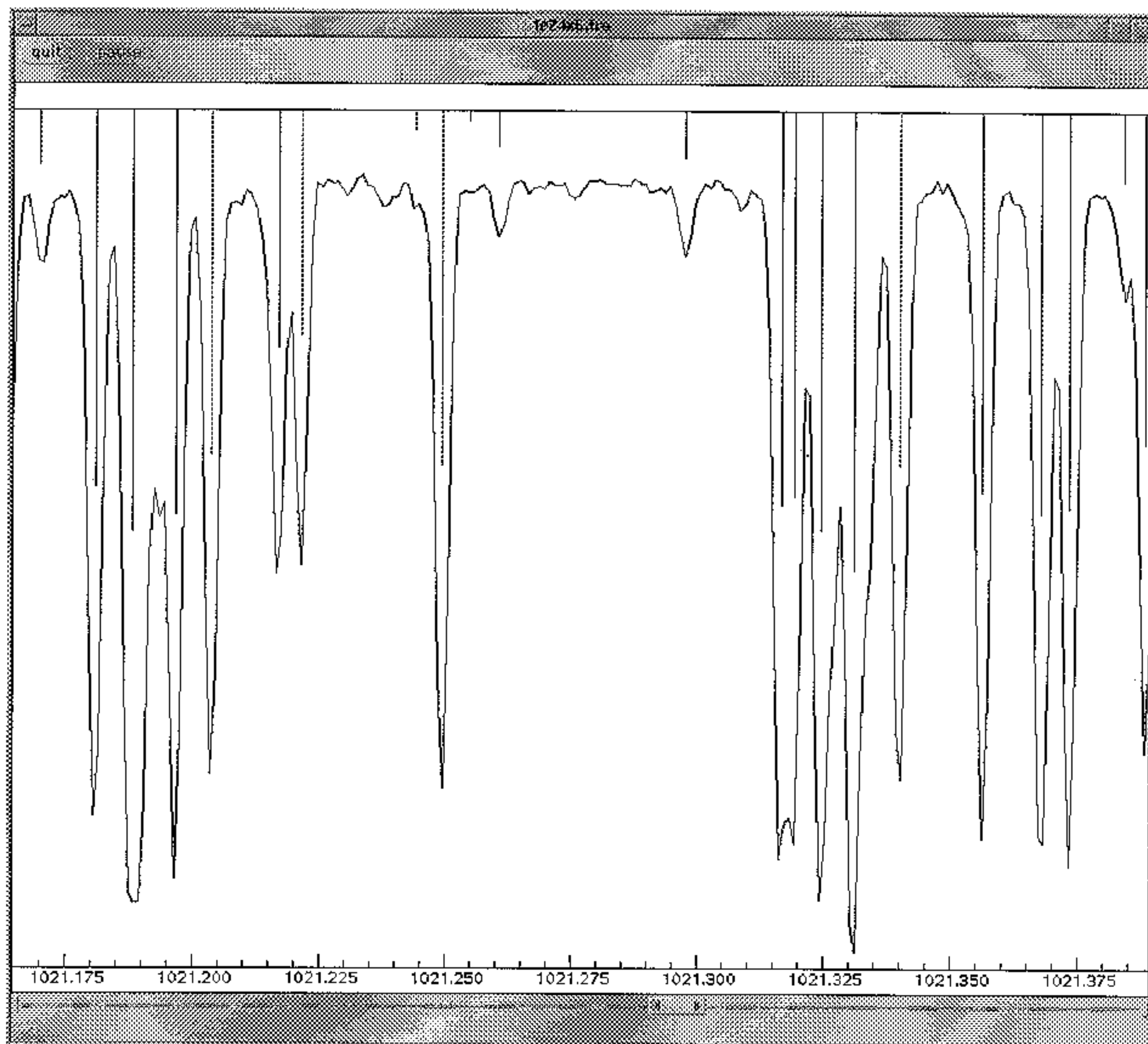

Figure 3.6: A piece of spectrum.

```
SpreadSheet is:
 R,
Index     J        wv          delta1       delta2       intens
11.            1017.018113   1.427566                  0.595500
12.            1018.445679   1.410012   -0.017554     0.466641
13.            1019.855691   1.393535   -0.016477     0.401719
14.            1021.249226   1.376004   -0.017531     0.378543
15.            1022.625230   1.358654   -0.017350     0.355367
16.            1023.983884   1.341201   -0.017453     0.337332
```

Figure 3.7: A piece of spreadsheet produced during the inference engine running, after adjusting the intensity of one line.

```
case 4:  if (equal(p_arg_value[0].i, p_arg_value[1].i))
           b_value.i=1;
         else
           b_value.i=0;
         b_type='i';
         return 1;
```

in `func_table.cc` to call the real C++ function, and set the return value and type. We also must give the real C++ function 'equal' in `mosaafunc.h` and `mosaafunc.cc`; this is quite simple for this `equal` function as shown on Fig 3.8.

Now, if one of the MOSAA-func rules that includes "Adjust_intens" is fired, and the premise of A_ R59 becomes true, in this case, there is no operation since what we want is to invoke a restart. Therefore, there is a property

```
& Relative 'C', 22
```

which will restart the whole up searching process, after the intensity of one line is adjusted. Section 4.2 gives further details about this '*restart*' process.

```
int equal(int A, int B);  //prototype in mosaafunc.h


int equal(int A,int B)    //C++ function in mosaafunc.cc
{
  if (A==B)
    return 1;
  return 0;

}
```

Figure 3.8: The C++ code added for adding a new mosaafunction 'equal'.

Chapter 4

INFERENCE ENGINE

The MOSAA system inference engine combines *backward chaining* and *forward chain ing*, as well as two special mechanisms *try* and *restart*. Appendix C gives the diagrams of this inference engine; here only some important concepts are discussed.

4.1 Three types of rules

As we mentioned, there are three groups of rules in the inference engine, and each group has a different role.

1. **Goal Rule**

   The goal rule starts the inference engine by trying to determine if the premise in the goal rule is true.

   Fig 4.1 is a goal rule which starts the whole assignment engine. At first it does several initial processes. If these finish successfully, it then trys to determine if parameter `All_Assign_Done` is true, which starts the whole engine.

2. **Consequence Rules**

   Consequence rules are used in the backward chaining. When the inference engine needs to determine a parameter's value, the consequence rules are tried one

```
G_R1    IF     $main_preprocess()
          &    All_Assign_Done
        THEN
               $final_step()
```

Figure 4.1: A sample goal rule.

by one. The consequence rule is tried only when it has a MOSAA function or subroutine condition, which can determine the inquiring parameter's value (modify type of this parameter must be 'r').

Rule `C_R38` in Fig 3.2 is a consequence rule. When the parameter `Transfer_TJ_to_P` value needs to be determined, `C_R38` is tried, since there is a MOSAA function '=' in the conclusion which can determine the value of parameter `Transfer_TJ_to_P`.

3. **Antecedent Rules**

   Antecedent rules are used in forward chaining. Once a parameter's value is determined or modified during or after doing the conclusions of one rule (which can be a Consequence or Antecedent rule), the forward chaining is invoked. An antecedent rule is tried when its premise has the parameter whose value has just been determined.

   As long as one antecedent rule is fired, no more antecedent rules are going to be tried, unless being forced by the "*Relative*" property.

   When there are any unknown parameters in the premise, this rule is skipped, unless it has the "*INVOKE*" property.

   There was a sample antecedent rule `A_R60` shown in Fig 3.4 previously, as well as some explanation about this rule in that section (section 3.2).

4.2 Special properties

Besides the basic backward chaining and forwarding chaining, some special properties of the rules also control the inference engine running.

INVOKE -

This property can only appear in an antecedent rule. Normally the rule is skipped, if it has an unknown parameter. If the rule has this **INVOKE** property, the 'simple' backward chaining is invoked to try to determine the value of this unknown parameter. The reason that we call it 'simple' backward chaining is that no forward chaining is invoked during this 'simple' backward chaining.

Relative -

This property also only can appear in an antecedent rule. It can be

Relative 'A', *rule_id* //first case

or

Relative 'C', *rule_id* //second case

When a rule with such a property is fired and all the conclusions are done, in the first case, the corresponding antecedent rule with that *rule_id* is invoked. For example, Fig 4.2 shows a group of A_R rules.

This group of antecedent rules are used to do the processing after the second line of three_lines has been found. If A_R4 is fired, A_R6 and A_R7 are successively tried.

The second case invokes a special mechanism "**restart**". Once the consequence rule with *rule_id* is the ancestor of the consequence rule which invokes the current antecedent rule, the inference engine will restart from that particular point. The complete environment (the value of all parameters) are reset at that point,

```
A_R4    IF     Sec_L_S_idx
        THEN
               Set TEMPI1, Cur_Peaks[Sec_L_S_idx].pkf_idx
          &    Set Cur_Peaks[Sec_L_S_idx].wv, PKF[TEMPI1].wv
          &    Set Cur_Peaks[Sec_L_S_idx].intens, PKF[TEMPI1].intens
          &    Set Cur_Peaks[Sec_L_S_idx].series_no, Cur_Series_No
          &    $show_peak(TEMPI1)
          &    Set Top_L_S_idx, (Minimum Fir_L_S_idx, Sec_L_S_idx)
          &    Set Bottom_L_S_idx, (Maximum Fir_L_S_idx,Sec_L_S_idx)
        #  Relative 'A', 6
         & Relative 'A', 7

A_R6    IF   Sec_L_S_idx
          &  > Cur_Peaks[Bottom_L_S_idx].wv, 0
          &  > Cur_Peaks[Top_L_S_idx].wv, 0
        THEN
             Set Cur_Peaks[Top_L_S_idx].delta1,
        Cur_Peaks[Bottom_L_S_idx].wv-Cur_Peaks[Top_L_S_idx].wv

A_R7    IF   Sec_L_S_idx
        THEN
             REMOVE  Search_Zone_No
          &  REMOVE  S_Low_Range
          &  REMOVE  S_High_Range
          &  REMOVE  I_Low_Range
          &  REMOVE  I_High_Range
```

Figure 4.2: A sample of using property "Relative 'A', x".

except for some parameters whose values are reserved by the MOSAA function "*RESERVE*". Fig 4.3 shows an example.

Basically, C_R18 tries to find the third line of three_lines. If C_R13 fails, parm `Thi_L_S_idx` is not determined; then the premise of C_R105 becomes true and C_R105 is fired, thus invoking antecedent rules. If A_R32 is fired, which means that there are some other second lines that can be used, after some processing, the inference engine restarts the deduction from the consequence rule C_R18. All the modifications of the parameters which were done up to this point are removed, and since we want to keep the information that another second line has been chosen, 'RESERVE' is used to save the new second line information.

TRY -

This property can only appear in a consequence rule. The basic idea is that, once a consequence rule is fired, the conclusion may have more than two choices. If, later on, the inference engine gets stuck, it can come back to make another choice.

C_R9 is a rule with the "TRY" property, which determines the search range for the second line of the three lines (see Fig 4.4). Once C_R10 which invoked C_R9 fails, the environment is reset, and the inference engine can come back to pick another choice of search range.

One must be very careful when mixing the **TRY** and **Relative**(restart) properties, since after restart, the deduction path may be changed by the parameters which are reserved, which can make 'TRY' fail. We will use an example to briefly discuss this problem; Fig 4.5 gives a group of hypothetical rules that we are going to use.

Assume that the inference engine wants to determine the value of parameter `Final_Parm`. Up to this point, we say the environment is *environment1*. The deduction path at first is as shown in Figure 4.6 (a). `C_R4` fails since `Parm_A` is less than 10.

```
C_R13   IF   FOUND Sec_L_S_idx
          &  UNFOUND Thi_L_S_idx
          &   .........
        THEN
             =  Thi_L_S_idx,  TEMPI1

C_R18  IF   Thi_L_S_idx
       THEN
            =  Thi_L_Found, 1


C_R105  IF   UNFOUND  Thi_L_S_idx
       THEN
             =  Thi_L_S_idx, 0

A_R32  IF   !Thi_L_S_idx
         &  Set TEMPI1, ( Pop_Line Line_Reserve_Stack, Sec_L_S_idx )
         &  TEMPI1
       THEN
            Set Cur_Peaks[Sec_L_S_idx].pkf_idx, TEMPI1
         &  = Sec_L_S_idx, Sec_L_S_idx       /%just for invoking purposes%/
         &  RESERVE Cur_Peaks[Sec_L_S_idx].pkf_idx
         &  RESERVE Cur_Peaks[Sec_L_S_idx].zone_no
         &  RESERVE Sec_L_S_idx, ´y´
         &  RESERVE Line_Reserve_Stack
       #Relative ´C´, 18
```

Figure 4.3: An example of using property "`Relative ´C´,x`".

```
C_R9    IF   FOUND Fir_L_S_idx
          &  UNFOUND Sec_L_S_idx
          &  == Cur_Series.branch, 'R'
        THEN
             =  Search_Zone_No,  Cur_Peaks[Fir_L_S_idx].zone_no+1
          &  Set TEMPI1,  Search_Zone_No-1
          &  Set TEMPD1, Cur_Peaks[Fir_L_S_idx].wv
                             + Zone_Area[TEMPI1].av_sp
          &  =  S_Low_Range,  TEMPD1 - S_RANGE1
          &  =  S_High_Range, TEMPD1 + S_RANGE1
          &  =  I_Low_Range, Cur_Peaks[Fir_L_S_idx].intens - I_EXT
          &  =  I_High_Range, Cur_Peaks[Fir_L_S_idx].intens+I_EXT
        #TRY
         {
             =  Search_Zone_No, Cur_Peaks[Fir_L_S_idx].zone_no-1
          &  Set TEMPD1, Cur_Peaks[Fir_L_S_idx].wv
                                   - Zone_Area[Search_Zone_No].av_sp
          &  =  S_Low_Range, TEMPD1 -  S_RANGE1
          &  =  S_High_Range, TEMPD1 + S_RANGE1
          &  =  I_Low_Range, Cur_Peaks[Fir_L_S_idx].intens - I_EXT
          &  =  I_High_Range, Cur_Peaks[Fir_L_S_idx].intens + I_EXT
         }
        /*_ P39 R38,39
          IF     first line of three lines is found(!=0)
          THEN   search the second line in the zone preceding the current
                 zone (zone which first line lies in),the wv search range
                 for this is  the first line - the distance between two
                 zones +-S_RANGE1 ( which is the search extension)
                 the intens of the second line must be in range intens of
                 the first line  +-I_EXT
          Property
              also can try:
                 search the second line in the zone following current zone
                 (zone which first line lies in),the wv search range for
                 this is  the first line - the distance between two zones
                 +-S_RANGE1 ( which is the search extension)
                 the intens of the second line must be in range intens of
                 the first line  +-I_EXT
*/
```

Figure 4.4: A sample of using property "TRY".

```
C_R1      IF     Parm_C                          A_R1  IF   !parm_B
            &    == Parm_C, 2                          THEN
          THEN                                             = Parm_A, 12
                 = Final_Parm, 1                         & RESERVE  Parm_A
                                                       #Relative 'C', 2
C_R2      IF     Parm_A
            &    Parm_B
          THEN
                 =  Parm_C,  1
          # TRY { = Parm_C, 2}

C_R3      IF     function_1
          THEN
                 =  Parm_A, 2

C_R4      IF     >  Parm_A, 10
          THEN
                 =  Parm_B, 4

C_R5      IF     UNFOUND  Parm_B
          THEN
                 =  Parm_B,0

C_R6      IF     UNFOUND  Final_Parm
          THEN
                 = Final_Parm, 0
```

Figure 4.5: A group of hypothetical rules used to illustrate the problem of mixing the use of 'Relative' and 'TRY'.

Then C_R5 is fired, and this invokes A_R1 to restart the deduction from C_R2. This time, Parm_A is already known since it was reserved by A_R1, and the deduction path changes to look like Figure 4.6 (b) . Now, C_R4 succeeds and determines Parm_B, which makes C_R2 become true and Parm_C to be 1.

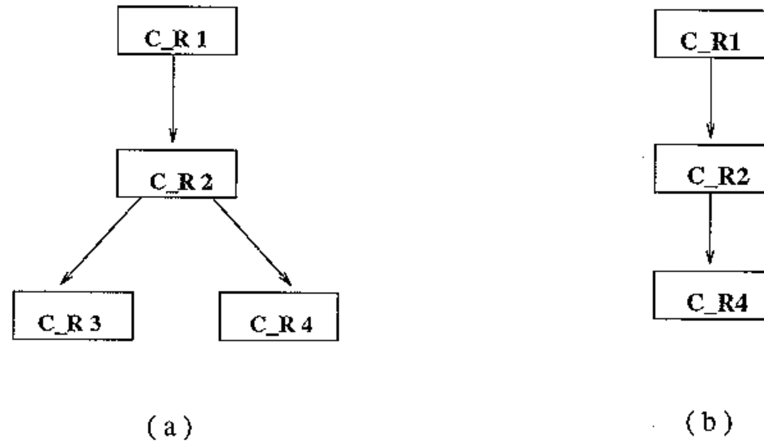

Figure 4.6: Simple deduction paths for the rules of Fig 4.5.

Returning to C_R1, condition 2 fails, before C_R1 fails, and the engine starts to search for "TRY" in C_R1's branches, and finds one in C_R2. The environment is reset as at the very beginning (*environment*1), and the engine starts deduction from C_R1 again. The deduction path now becomes figure 4.6 (a) again, while it should be the same as Figure 4.6 (b)- which sets the Parm_B value, so "TRY" fails.

This case which mixes the **TRY** and **Relative** properties should be avoided when adding or modifying rules in the rule base.

Chapter 5

RECOMPILING MOSAA

There are two versions of MOSAA; the *command line* version and the *xview* version.

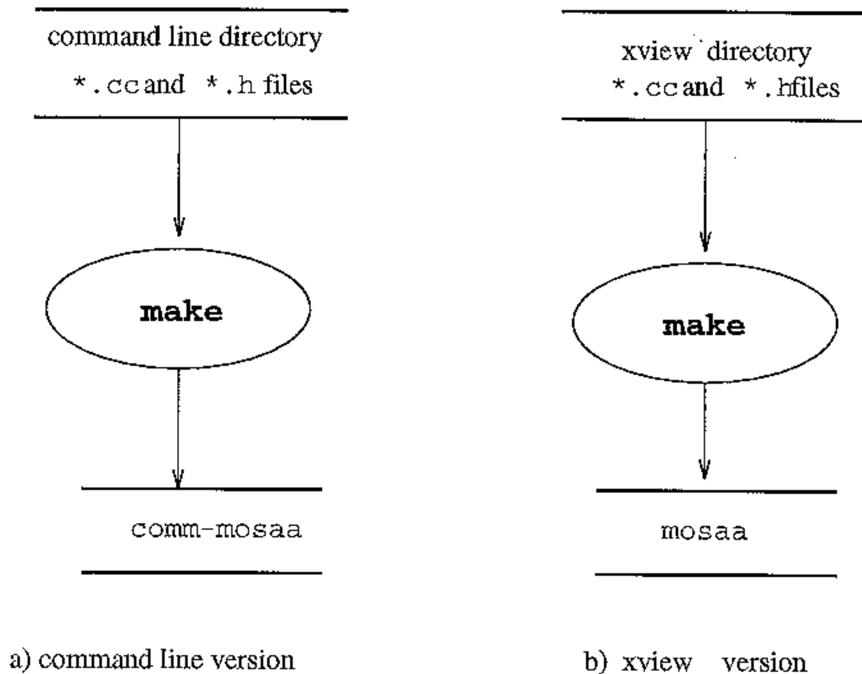

Figure 5.1: Two versions of MOSAA.

The final version will be the *xview* version, since it includes a graphical user interface. During the development of MOSAA, the *command line* version is more convenient for debugging both the C++ MOSAA system files, and the _.rule file.

5.1 Source files

As of May, 1996, all of the source files for building MOSAA are in anonymous ftp site

```
physi02.novlab.unb.ca/pub/mosaa/command (command line version)
                  ~                  /xview    (xview version)
                  ~                  /input-file (input files)
```

5.2 User input files

As figure 1.1 shows, there are several input files for running `mosaa`, and those files are all in the `input-file` directory.

Spectroscopy files

`calo18.Q.GD`	`fe24x6.pkf`
`calo18.RP.GD`	`fe24x6.tra`

These are a group of files for the $CH_3^{18}OH$ co-stretch band.

`calo18.Q.GD` is the calculated R-Q combination difference and `calo18.RP.GD` is the calculated R-P combination difference. Fig 5.2 is a portion of the `calo18.Q.GD` file, where '`%`' lines are comment lines and '`#`' shows the end of the file.

`fe24x6.pkf` is the peakfinder file for this spectrum. Figure 5.3 shows a portion of the `fe24x6.pkf` file.

The lines before line '`7459`' are comment lines. During the running of the `mosaa` program, the user is prompted for the number of comment lines. After the comment lines, is the number of peaks '`7459`' following by the peak information (*wavenumber* and *intensity*).

```
%HENNINGSEN ENERGIES  R M LEES  PHYSICS  PHONE 4723
%MODIFIED 23 FEB 87
%
% CONVERSION FACTOR= 505376.0 AMU-A2-MHZ
%SPEED OF LIGHT= 29979.25
%

(000) E1

    1.546694
    3.093124
    4.639031
      ....
      ....

   59.069812
   60.401280

(000) E2

    1.546694
    3.093124
    4.639031
       ....
       ....

   57.716561
   59.216926
   60.715852
   62.213299

#
```

Figure 5.2: A portion of a combination difference file (fe24x6.GD).

```
181
7459
    899.669788      0.963311
    900.881746      0.959347
    908.112281      0.948798
    908.176817      0.924650

        ..............
        ..............

   1098.530339      0.921143
   1098.537077      0.927045
   1098.548244      0.895166
   1098.561766      0.963333
```

Figure 5.3: A portion of a peakfinder file (`fe24x6.pkf`).

`fe24x6.tra` is the plotting file for displaying the spectrum on screen in the *xview* version. Fig 5.4 shows a portion of such a file.

The lines before line `0.00099817261382` are the comment lines; the number of the comment lines is given during the running of the `mosaa` program.

In the following parts of the file, '`0.00099817261382`' gives $\triangle x$, '`899.21877175110001`' gives the start x, and '`1009.99996112400004`' is the end x. The following parts are y for each of the plotting dots.

Knowledge_base files

`mar.mosaa` `mar.prop` `mar.sub`

`mar.parm` `mar.rule`

There is substantial discussion of these files in previous chapters 1, 2 and 3.

```
fe24x6

110985
   0.00099817261382
 899.21877175110001
1009.99996112400004
0.994100  0.995800  0.996900  0.990200  0.991400  0.993300  0.996600
0.998600  0.994800  0.993700  0.994500  0.989500    ....
          .............
          .............
0.97053  0.97285  0.99376  0.99370  0.98753  0.98388  0.99183  0.97765
0.98336  0.99018  0.97419  0.98664  0.97605  0.97114  0.98514  0.99331
```

Figure 5.4: A portion of a spectrum file (fe24x6.tra).

5.3 Command line version

5.3.1 Source files

The source files for building a command line version of MOSAA are in the

`command`

directory, which includes:

- lex&yacc files

    input_rule.l

    input_rule.yacc

- .h files

cd_base.h	mosaafunc.h	comm-inter.h
parm_table.h	comm-spc.h	pkf.h
const.h	prop_table.h	error_stack.h

rule.h	func_table.h	sub.h
hfile_style.h	wm.h	ht.h
zone_area.h	main_func_ptype.h	

- .cc files

cd_base.cc	mosaafunc.cc	comm-inter.cc
parm_table.cc	comm-mosaa.cc	pkf.cc
comm-spc.cc	prop_table.cc	common.cc
rule.cc	error_stack.cc	sub.cc
func_table.cc	tt.cc	global_var.cc
union.cc	ht.cc	wm.cc
zone_area.cc		

5.3.2 Compiling files

Once a source file is modified, we need to recompile. The files for recompiling in directory `physi02:/home/e7qz/command` include:

```
sed-script.lex.yy.c
doit
Makefile
```

'`doit`' is a batch file which has only a few lines, as follows:

```
lex input_rule.l
sed -f sed-script.lex.yy.c lex.yy.c >>lex.yy.cc
mv lex.yy.cc lex.yy.c
bison -d -t   input_rule.yacc
mv input_rule.yacc.tab.c input_rule.yacc.tab.cc
```

Except for calling `lex` and `bison` to generate the corresponding files as shown in Figure 1.2, it also calls a unix utility `sed` to do some modification in the lex output file `lex.yy.c`. The output file of `lex` is not an ANSI C file. Since we are using GNU C++ for compiling, some modifications must be done to `lex.yy.c`, such as changing the function prototype style, to change `lex.yy.c` to be an ANSI C file.

If one of the lex&yacc files is modified, two commands should be used:

```
doit
make
```

If one of the `.h` or `.cc` files is modified while the `.l` and `.yacc` files are untouched, we only need to run `make`.

5.3.3 Debugging

Due to the complexity of the source files and the inference engine produced, some way to make debugging easier is necessary.

Inside the source code, there are lots of `printf` lines. They are used to show the process of the inference engine, which can be used to find bugs in the source code producing the inference engine, and are also very useful to check if the rules in the `_.rule` file are proper.

When the program is small, or, the rough location of the bug is already known, a debug tool such as `xxgdb` is very useful. To use `xxgdb`, one must make sure to set the debug parameter '-g' in `Makefile` for compiling.

The reason we have a command line version, which is not a final version, is because of the difficulty of the debugging problem. Since the only difference between the 'command line' version and the 'xview' version is just the user interface, we transfer the source code into the xview directory when it is known to work correctly.

5.4 xview mode

5.4.1 xview version source files

The source files for building the xview version MOSAA are in the

```
xview
```

directory. This directory is still being constructed, so the files described here are subject to change.

The names of the sets of files are almost the same as for the command line version, where some files have a lot differences, and some have relatively trivial differences.

The following files only appear in xview:

```
grah_func_ptype.h
assign.cc      interface.cc   media.cc
```

The following files have a lot differences compared to the command line version:

```
spc.h          main_func_ptype.h
spc.cc         mosaa.cc
global_var.cc
```

The following files have relatively trivial differences compared to the command line version:

```
common.cc      rule.cc        error_stack.cc
func_table.cc  wm.cc          parm_table.cc
ht.cc          pkf.cc         prop_table.cc
```

These files can be easily transfered from command line mode by modifying the `include` files. For example, instead of using `comm-inter.h`, we use `grah_func_ptype.h`. Thus we get a graphical user interface.

The following files are the same as the command line version:

const.h	pkf.h	rule.h
error_stack.h	func_table.h	parm_table.h
wm.h	ht.h	prop_table.h

5.4.2 xview programming

Although the graphical user interface was built using X window(xview) programming, there is one concept one needs to be concerned about.

Window applications usually are event driven; since `mosaa` basically is run by the inference engine, it is mainline driven [8]. Instead of using `xv_main_loop`, we use:

```
while (!finished)
  {
    notify_dispatch();
    XFlush(main_dpy);
    if (start_assign==1)
assign();
  }
```

`assign()` is a function that drives the whole inference engine, which invokes a lot of user interactive functions. If we put `assign()` into a callback, the events won't be dispatched until the callback returns. This is not convenient for an interactive interface. When the function `assign` is not a callback, we can put `notify_dispatch` in any place (except in the call back) to explicitly dispatch the event. This is very convenient since `mosaa` basically is a mainline driven program. For further information

about X window/xview programming, please refer to an xview programming reference book, e.g. [8].

5.5 Auxiliary files

To add a new MOSAA function or subroutine to the rule base, the definition of this function needs to be added into the `_.mosaa` or `_.sub` file, as mentioned in section 3.2. We also need to add some code into `func_table.c`, where the index of this function in the func_table is needed.

As long as this function is in the `_.mosaa` or `_.sub` file, we can run '`mosaa`' to get two auxiliary files: "`sub.idx`" and "`mosaa.idx`", which tells the index of this function in the corresponding table. These two files are automativally generated whenever `mosaa` is executed. Figure 5.5 is a portion of file `mosaa.idx`.

```
                    mosaa.idx

 0.    != i i i
 1.    != i d d
 2.    != i c c
 3.    != i s s
 4.    == i i i
 5.    == i d d
 6.    == i c c
 7.    == i s s
 8.    > i i i
 9.    > i d d
10.    > i i d
11.    > i d i
```

Figure 5.5: Auxiliary output file `mosaa.idx`.

Section 3.2 discussed about how to add in a new function '`==`'. The code we added into `func_table.cc` is

```
case 4:  if (equal(p_arg_value[0].i, p_arg_value[1].i))
           b_value.i=1;
         else
           b_value.i=0;
         b_type='i';
         return 1;
```

and the '4' for case statement is obtained from the mosaa.idx file.

Chapter 6

Conclusions

The development environment for MOSAA depends heavily on the Linux(Unix) environment, which includes the following components:

GNU C++ compiler (*gcc*)
lex
bison
X window/Xview

A good development environment also needs some auxiliary software such as an **emacs** editor and the X windows debugger **xxgdb**.

It is relatively easy to add or modify some simple rules once the basic ideas described in this guide are understood and by following the appropriate instructions.

To make a large modification of the rule base or even to modify the inference engine, one must have a good understanding of the inference engine and how it works with the rules.

Bibliography

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 1.1, Dec. 1991, available from anonymous ftp site: scss3.cl.msu.edu/pub/standands/corba.ps.gz.

[2] Free Software Foundation, *bison.info*, comes with slackware-2.3 distribution, available from anonymous ftp site: ftp.cdrom.com/pub/linux/slackware

[3] I.Mukhopadhyay, R.M.Lees, W.Lewis-Bevan, and J.W.C.Johns, "Fourier Transform Spectroscopy of the CO-Stretching Band of C-13 Methanol in the Torsional Ground State",J.Chem.Phys. **102**, pp. 6444-6455, 1995.

[4] R.M.Lees, "Far-Infrared and Infrared Spectroscopy of Methanol Applied to FIR Laser Assignments", *Far-Infrared Science and Technology, ed. J.R.Izatt, Proc.SPIE* Vol. **666**, pp. 158-170, 1986.

[5] John Levine, Tony Mason & Doug Brown, *lex & yacc*, O'Reilly & Associates, Inc. Sebastopol,CA, 1992.

[6] Texas Instruments, *Personal Consultant^{TM} Easy, Reference Guide*, Austin, Texas, 1987.

[7] Li-Hong Xu, High-Resolution Fourier Transform Spectroscopy of $^{13}CD_3OH$ With Far-infrared Laser Analysis, Ph.D. thesis, University of New Brunswick, 1992.

[8] Dan Heller, *XView Programming Manual*, Updated for Xview Version 3 by Thomas Van Rattlte, O'Reilly & Associates, Inc. Sebastopol,CA, 1991.

[9] Saibei Zhao, Infrared Fourier Transform Spectroscopy of O-18 Methanol, Ph.D. thesis, University of New Brunswick, 1993.

[10] S.Zhao, R.M.Lees, J.W.C.Johns, C.P.Chan, and M.C.L.Gerry, "Fourier Transform Spectroscopy of$CH_3^{18}OH$: The In-Plane CH_3-Rocking Band", *J.Mol.Spectrosc.* **172**, pp. 153-175, 1995.

Appendix A

lex file

This appendix gives the complete input_rule.l file which is used for building the lexical analyzer.

```
%{

/*
                    May 24, 1996                          input_rule.l
            lex file of input_rule
*/

#include <stdio.h>
#include "main_func_ptype.h"
#include "const.h"

extern int DEBUG_L;

char y_l_id[YYLMAX];
int y_l_lineno=0;

%}
```

```
%%
"C_R"[0-9]*            { if (DEBUG_L)
                             printf("--%s\n",yytext);
                         strcpy(y_l_id,yytext);
                         return CR;
                       }

"G_R"[0-9]*            { if (DEBUG_L)
                             printf("--%s\n",yytext);
                         strcpy(y_l_id,yytext);
                         return GR;
                       }

"A_R"[0-9]*            { if (DEBUG_L)
                             printf("--%s\n",yytext);
                         strcpy(y_l_id,yytext);
                         return AR;
                       }

"IF"                   { if (DEBUG_L)
                             printf("--%s\n",yytext);
                         strcpy(y_l_id,yytext);
                         return IF;
                       }

"THEN"                 { if (DEBUG_L)
                             printf("--%s\n",yytext);
                         strcpy(y_l_id,yytext);
                         return THEN;
```

```
                    }

"FOR"               { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return FOR;
                    }

"ANYOF"             { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return ANYIF;
                    }

"DO"                { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return DO;
                    }

"~"                 { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return MOSA_NAME;
                    }

"!"                 { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return MOSA_NAME;
```

```
                        }

"!="                    { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return MOSA_NAME;
                        }

"=="                    { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return MOSA_NAME;
                        }


">="                    { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return MOSA_NAME;
                        }

"<="                    { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return MOSA_NAME;
                        }

"<<"                    { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
```

```
    return MOSA_NAME;
  }

">>"                  { if (DEBUG_L)
                          printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return MOSA_NAME;
                      }

"<"                   { if (DEBUG_L)
                          printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return MOSA_NAME;
                      }

">"                   { if (DEBUG_L)
                          printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return MOSA_NAME;
                      }

"++"                  { if (DEBUG_L)
                          printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return MOSA_NAME;
                      }

"--"                  { if (DEBUG_L)
                          printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
```

```
        return MOSA_NAME;
      }

[{}&%$;,:=()#]      { if (DEBUG_L)
                          printf("--%s\n",yytext);
                       strcpy(y_l_id,yytext);
                       return yytext[0];
                    }

[+-/]               { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return yytext[0];
                    }

"*"                 { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return yytext[0];
                    }

"i"                 { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
                      return LOOPI;
                    }

"d"                 { if (DEBUG_L)
                          printf("--%s\n",yytext);
                      strcpy(y_l_id,yytext);
```

```
                        return LOOPD;
                      }

"["                   { if (DEBUG_L)
                            printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return yytext[0];
                      }

"]"                   { if (DEBUG_L)
                            printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return yytext[0];
                      }

\n                    { y_l_lineno++;
                        if (DEBUG_L)
                            printf("line---%d\n",y_l_lineno);
                       }

"."                   { if (DEBUG_L)
                            printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return yytext[0];
                      }

[0-9]+                { if (DEBUG_L)
                            printf("--%s\n",yytext);
                        strcpy(y_l_id,yytext);
                        return INTEGER;
```

```
                              }

[0-9]*"."[0-9]+               { if (DEBUG_L)
                                    printf("--%s\n",yytext);
                                strcpy(y_l_id,yytext);
                                return(DOUBLE);
                              }

[a-zA-Z][a-zA-Z0-9_]*          { if (DEBUG_L)
                                     printf("--%s\n",yytext);
                                 strcpy(y_l_id,yytext);
                                 /*if it is a mosa func name*/
                                 if (is_mosa_name(y_l_id))
                                       return MOSA_NAME;
                                 return IDENTIFIER;
                                /*strcpy(y_l_id,yytext);
                                 return(IDENTIFIER);*/
                               }

"/*"[^\*]*"*/"                { if (DEBUG_L)
                                    printf("****%s\n",yytext);
                                strcpy(y_l_id,yytext);
                                char temp[YYLMAX], *next;
                                                    /*count how many '\n' inside*/
                                strcpy(temp,y_l_id);
                                while ( (next=strchr(temp,'\n'))!=NULL)
                                   { y_l_lineno++;
                                     strcpy(temp,next+1);
                                   }
                                return COMMENT;
```

```
                    }

"'"[a-zA-Z0-9\ ]"'"     { if (DEBUG_L)
                              printf("--%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return CHAR;
                        }

\"[^\"]*\"              { if (DEBUG_L)
                              printf("----%s\n",yytext);
                          strcpy(y_l_id,yytext);
                          return STRING;
                        }

"/%"[^\%]*"%/"          {  /*this is the comment just for user , skip it*/
                          if (DEBUG_L)
                              printf("****%s\n",yytext);

                          strcpy(y_l_id,yytext);
                          char temp[YYLMAX], *next;
                                               /*count how many '\n' inside*/
                          strcpy(temp,y_l_id);
                          while ( (next=strchr(temp,'\n'))!=NULL)
                              { y_l_lineno++;
                                strcpy(temp,next+1);
                              }
                        }

.                       ;
%%
```

Appendix B

Some input files

This appendix gives knowledge_base files used as part of the input files for program `mosaa`.

B.1 Rule Summary

The complete rule file `mar.rule` is not included in this appendix because of the number of pages it requires; about 60 pages not including the explanation part of each rule. However, `mar.rule` can be found in anonymous ftp site

```
physi02.novlab.unb.ca/pub/mosaa/input-file (input files)
```

The numbers of rules are as shown in table B.1.

Table B.1: Numbers of different rule types

Rule type	Number
Total Rules	196
Goal Rules	1
Consequence Rules	110
Antecedent Rules	85
Rule with `TRY` property	3
Rule with `Relative 'A'` property	8
Rule with `Relative 'C'` property	15

The following part shows the id of the rules in `mar.rule`, as well as some associated properties. The superscript "T" with rule id means this rule has the TRY property. The superscript "RA" or "RC" with rule id means this rule has a relative antecedent rule or a relative consequence rule, respectively.

1. **Goal Rule**

   G_R1

2. **Consequence Rule**

   C_R1, C_R2, C_R6 to C_R8, C_R9^{T}, C_R10, C_R12^{T}, C_R13, C_R16 to C_R22, C_R25 to C_R31, C_R32^{T}, C_R33 to C_R43, C_R51 to C_R60, C_R99, C_R100 to C_R106, C_R180 to C_R183, C_R186 to C_R197, C_R200 to C_R203, C_R212 to C_R217, C_R221 to C_R226, C_R230, C_R234 to C_R239, C_R250 to C_R260, C_R270 to C_R275, C_R280.

3. **Antecedent Rule**

   A_R1, A_R3, A_R4^{RA}, A_R6, A_R7, A_R8^{RA}, A_R9 to A_R11, A_R12^{RA}, A_R13, A_R14, A_R15^{RC}, A_R16 to A_R18, A_R19^{RA}, A_R20^{RC}, A_R25^{RA}, A_R26 to A_R28^{RC}, A_R30^{RA}, A_R31^{RC}, A_R32^{RC}, A_R33^{RA}, A_R34 to A_R36, A_R37^{RA}, A_R38 to A_R41, A_R44 to A_R54, A_R59^{RC}, A_R60, A_R61, A_R70^{RC}, A_R71^{RC}, A_R72^{RC}, A_R80 to A_R81, A_R86 to A_R96, A_R100 to A_R109, A_R111, A_R112, A_R114^{RC} to A_R117^{RC}, A_R130 to A_R134.

B.2 _.parm file

```
%                              May24, 1996                                 mar.parm
%
% format
% parm_name   has_index  fields_num
%                                  0  :   value_type  ques
%                                !=0  :   field_name value_type ques, field_name...
%

All_Assign_Done nindex 0 i nques


A_Prerule  nindex 0 i nques


A_Dorule nindex 0  i nques


A_Prefail nindex 0 i nques


Analysis_ok nindex 0 i nques


Assign_Cur_Series nindex 0 i nques


Bottom_L_S_idx  nindex 0  i nques


Two_B nindex 0 d yques


CA_CD_Exist nindex 0 i nques


CA_CD nindex 0 d nques
```

```
%problem, shouldn't have two kind of CD
CD nindex 0 d nques

CD_Exist  nindex 0 i nques

CD  nindex  4 K i yques n i yques t i yques prop s yques

CD_Candidate_Stack nindex 0 k nques

CD_Candidate_Stack_Num  nindex 0 i nques

CLOSE_DEFI  nindex 0 d yques

Candi_Queue  yindex  0 i nques

Cur_Series  nindex  7 branch c yques K i nques n i nques t i nques prop  c  yques
      series_no i nques Symm s nques

Cur_Series_No nindex 0 i nques

Cur_Peaks  yindex  11  J i nques wv d nques intens d nques pkf_idx i yques
       delta1 d nques delta2 d nques series_no i nques bias d nques
                        zone_no i nques confirm i nques from c nques

Cur_Branch nindex 0 c yques

Cur_S_idx  nindex 0 i nques

Cur_CD  yindex   0  d nques
```

```
Cur_CD_series  nindex  4 n i yques t i yques  K i yques prop c yques

Cur_R_Series nindex 8 n i yques t i yques K i yques prop s yques top i nqyes
                 bottom i nques top_of_three i nques bottom_of_three i nques

Cur_Q_Series nindex 8 n i yques t i yques K i yques prop s yques top i nqyes
                 bottom i nques top_of_three i nques bottom_of_three i nques

Cur_P_Series nindex 8 n i yques t i yques K i yques prop s yques top i nqyes
                 bottom i nques top_of_three i nques bottom_of_three i nques

Cur_R_Peaks  yindex 9 pkf_idx i nques wv d nques intens d nques overlap i yques
       delta1 d nques delta2 d nques J int nques confirm int nques from c nques

Cur_Q_Peaks  yindex 9 pkf_idx i nques wv d nques intens d nques overlap i yques
       delta1 d nques delta2 d nques J int nques confirm int nques from c nques

Cur_P_Peaks  yindex 9 pkf_idx i nques wv d nques intens d nques overlap i yques
       delta1 d nques delta2 d nques J int nques confirm int nques from c nques

Cur_Zone  nindex 0 i nques

Cur_PKF_idx nindex 0 int nques

Cur_Found_Line_Num nindex 0 int nques

Cur_Found_Line nindex 3 pkf_idx int nques  from  char nques S_idx int nques

Cur_Found_R_Line nindex 3 pkf_idx int nques  from  char nques S_idx int nques
Cur_Found_Q_Line nindex 3 pkf_idx int nques  from  char nques S_idx int nques
```

```
Cur_Found_P_Line nindex 3 pkf_idx int nques  from  char nques S_idx int nques

RPQ_CONFIRM_ET nindex 0 double  nques

Did_Select nindex 0 i nques

E_T_low  nindex 0  d  nques

E_T_high nindex 0  d  nques

ERROR_TOLERANCE nindex 0 d yques

Exist_Series nindex 0 i nques

Fir_L_S_idx  nindex 0 i yques

Fir_L_Found nindex 0 i nques

FAR_MORE_DEFI nindex 0 d nques

Incorrect_L  nindex 0 i nques

I_SEARCH nindex 0 d yques

I_Low_Range nindex 0 d yques

I_High_Range nindex 0 d yques

I_EXT nindex 0 d yques
```

```
I_EXT_FOR_WEAK_PEAK nindex 0 d yques

INTENS_ADJUST_LIMIT nindex 0 d yques

Last_Zone  nindex 0 i nques

%this parm hiddenly has "S_idx" and "pkf_idx" two fields
Line_TJ_Candidate_Stack  nindex 0 k nques

Line_TJ_Candidate_Stack_Num  nindex 0 i nques

Line_Candidate_Stack  nindex 0 k nques

Line_Candidate_Stack_Num  nindex 0 i nques

Line_TJ_Reserve_Stack nindex 0 k  nques

Line_TJ_Reserve_Stack_Num nindex 0 i nques

Line_Temp_Stack nindex 0 k  nques

Line_Temp_Stack_Num nindex 0 i nques

MAX_DELTA2  nindex 0 d yques

MAX_SERIES_NUM nindex 0 i yques

MAX_BIAS  nindex 0 d yques

MINI_DELTA2 nindex 0 d yques
```

```
MINI_S_L_NUM nindex 0 i yques

MISS_LINE nindex 0 i yques

More_Transfer nindex 0 i nques

Others_Found  nindex 0 i nques

P_Initial_Assign nindex 0 i nques

P_Assignment nindex 0 i nques

PKF  yindex 6  wv  d  nques  intens d  nques overlap i nques adjust_wv i nques
                   adjust_intens i nques assigned i nques

Preprocess nindex 0 i nques

Pro_L  nindex 0 i nques

R_To_Q_Pro_L nindex 0 i nques

P_CONFIRMED  nindex 0 i nques

Pre_P_Series nindex 2 top_wv doub nques bottom_wv doub nques

Pre_R_Series nindex 2 top_wv doub nques bottom_wv doub nques

Q_za_no  nindex 0 i nques
```

```
Q_0  nindex  2  wv d yques  pkf_idx i nques

ROUGH_2B  nindex 0  d  yques

Reserve_TJ nindex 9 fir_S_idx i nques sec_S_idx i nques thi_S_idx i nques
             fir_pkf_idx i nques sec_pkf_idx i nques  thi_pkf_idx i nques
             fir_zone_no i nques sec_zone_no i nques thi_zone_no i nques

R_Initial_Assign nindex 0 i nques

R_P_Correct  nindex 0 i nques

R_P_Initial_Assign_Done nindex 0 i nques

R_P_Comm_Assign_Done nindex 0 i nques

RESL  nindex 0 d yques

R_P_MATCH_ET nindex 0 d yques

R_J_Determined nindex 0 i nques

ReturnI1 nindex 0 i nques
ReturnI2 nindex 0 i nques
ReturnI3 nindex 0 i nques
ReturnI4 nindex 0 i nques

ReturnD1 nindex 0 d nques
ReturnD2 nindex 0 d nques
ReturnD3 nindex 0 d nques
```

```
ReturnD4 nindex 0 d nques

STUCK_AT nindex 0 i nques

Sec_L_S_idx nindex 0 i nques

Sec_L_Found nindex 0 i nques

Search_Dire  nindex 0  c nques

Search_L_Range  nindex  0 d nques

Search_H_Range  nindex  0 d nques

Series  yindex  2  branch  c  yques  K  i  nques

SEARCH_EXT1 nindex 0 i nques

S_RANGE1  nindex  0 d  yques

SEARCH_WV_EXT  nindex 0 d yques

S_Low_Range nindex 0 d yques

S_High_Range nindex 0 d yques

Search_Zone_No nindex 0 int nques

STRONG_PEAK_INTENS nindex 0 double nques
```

```
Stop_Transfer nindex 0 int nques

Range_Ext  nindex 0 d yques

Three_Found nindex 0 i nques

Thi_L_S_idx  nindex 0  i  nques

Thi_L_Found nindex 0 i nques

Top_L_S_idx nindex 0 i nques

Top_of_Three_L nindex 0 i nques

Transfer_FJ_To_P nindex 0 i nques

Transfer_FJ_To_P_Done nindex 0 i nques

Bottom_of_Three_L nindex 0 i nques

Up_And_Down_Extension nindex 0 i nques

Zone_Area  yindex 6 idx_1st i nques  idx_2nd i nques  av_sp d nques
                       line_num i nques  branch c  yques  J i nques

Zone_Analysis_ok  nindex 0 i nques

Zone_Num  nindex 0 i nques
```

```
%temp variables, the scope is only one rule, so there is a class
%Temp_Var_Stack to deal with that

Bias1  nindex 0 d nques
Bias2  nindex 0 d nques
Bias3  nindex 0 d nques

Delta2_Dire nindex 0 doub nques

L0  nindex 0 i  nques
L1  nindex 0 i  nques
L2  nindex 0 i  nques
L3  nindex 0 i  nques
L4  nindex 0 i  nques
L5  nindex 0 i  nques
L6  nindex 0 i  nques

Pkf_idx1 nindex 0 i nques
Pkf_idx2 nindex 0 i nques

TEMPI1 nindex 0 i nques
TEMPI2 nindex 0 i nques
TEMPI3 nindex 0 i nques
TEMPI4 nindex 0 i nques

TEMPD1 nindex 0 d nques
TEMPD2 nindex 0 d nques
TEMPD3 nindex 0 d nques
TEMPD4 nindex 0 d nques
```

```
TEMPC1 nindex 0 c nques
TEMPC2 nindex 0 c nques

i nindex 0 i nques

d nindex 0 d nques

S_idx  nindex 0 i nques
#
```

B.3 _.sub file

```
%                          May 24, 1996                          jin.sub
%
%Format:
% subtoutine_name,  return_type  [arg1_type arg2_type arg3_type ....]
%


CA_CD_get_3J  int


CA_CD_get_P_3J int


CD_assign_P int


adjust_wv int int_pkf_idx
    %adjust pkf_idx's wv by asking user


adjust_intens int int_pkf_idx
    %adjust pkf_idx's intens by asking user


assign_1st_L_P  int


ask_y_n_ques int string
    %ask a questione--string, get y--1, n--0


choose_CA_CD int


choose_CA_CD int int


choose_CD int string int
```

```
choose_CD int

choose_fir_line int

choose_sec_line int

choose_thi_line int

down_search_from_TJ int

final_step int

final_P_process int

final_R_process int

get_Aj_1  doub  doub  doub doub
%case of  bias1+3Aj, bias2-3Aj, bias+Aj in ET

get_Aj  doub  doub

get_2B_quota int

get_click_peak int  string
    %user click to get one peak, return this peak's pkf_idx

get_zone_no int int
    %according to pkf_idx, get the corresponding zone no
```

```
is_overlapped int int_pkf_idx
    %show peak pkf_idx on screen, and ask user if it is overlapped, (if it is
    %already known overlapped, just show it to the user. return 1 if it is
    %overlapped

main_preprocess int

process  int

prompt int string
    %just show message

show_peak int int
    %display the new peak just got for the current series

show_sh   int
    %show the spreadsheet style of the current series

sub_preprocess int
    %the preprocess of assigning each series , return 0 if user doens't want to
    %assign any more

sub_final_step int
    %the final step after assigning each series

up_search_from_TJ int

ws_process int

get_pkf_idx  int  d_wv
```

```
    %get the pkf_idx of the peak with wavenumber d_wv

search_line  int  i_line_no d_s_low_range, d_s_high_range, d_i_low_range,
                  d_i_high_range
    %search line i_line_no in wv and intensity range,  put them into
    %Line_Candidate_Stack, return the number of lines found

move_cur_series_to_R int

move_cur_series_to_P int

move_R_to_cur_series int

move_P_to_cur_series int

clean_stack_except_TJ int k_stack undo i_fir i_sec i_thi
    %remove all the lines from the stack except the three lines

get_R_from_P int d_wv  i_J
get_R_from_P int d_wv  i_J  i_K i_n s_Symm

get_P_from_R int d_wv i_J
get_P_from_R int d_wv i_J i_K i_n s_Symm

show_sh int c_branch

process_after_R int
process_after_P int

up_remove_R_peaks_from int int
```

```
down_remove_R_peaks_from int int

up_remove_P_peaks_from int int
down_remove_P_peaks_from int int

clean_stack in k_stack undo

get_Q_from_R int d_wv i_J i_K i_n s_Symm

select_top_line int  c_branch  i_pkf_idx1 c_from  i_pkf_idx2 c_from
%choose one top line  for c_branch
%c_from shows where it from, c_from='P', means from P branch,
        % else , 'R', ' '


select_bottom_line int  c_branch  i_pkf_idx1  c_from i_pkf_idx2 c_from
%choose one bottom line  for c_branch

another_one_from int i_data1 i_data2  i_data3
%return the data from i_data1, i_data2, which is not equal to i_data3
#
```

B.4 _.mosaa file

```
%                          May24,1996                         mar.mosa
%
% format:
% mosa_function_name  return_type  [arg1_type arg2_type arg3_type ...]
% int-i, double-d, char-c void-v  arbitrary-?


!=  int  i i
!=  int  d d
!=  int  c c
!=  int  s s


==  int  i i
==  int  d d
==  int  c c
==  int  s s


>   int  i i
>   int  d d
>   int  i d
>   int  d i

<   int  i i
<   int  d d
<   int  i d
<   int  d i
```

```
>>  int  i i
>>  int  d d
>>_remove  int  d i
>>_remove  int  i d

<<  int  i i
<<  int  d d
<<_remove  int  i d
<<_remove  int  d i

~   int  i i
~   int  d d
~_remove   int  i i
~_remove   int  d d

!   int  i
!   int  d

++ int tparm modify
++ doub eparm modify

-- int tparm modify
-- doub eparm modify

= int tparm reasoning  i
= int eparm reasoning d
=  int eparm reasoning i
= int rparm reasoning c
= int gparm reasoning s
```

```
ABS  int  int
ABS  double doulbe

Adjust_wv  int i_S_idx
% this is a rule function, means did adjusting line S_idx

Adjust_intens  int i_S_idx

Adjust_remove int int double

Delta2_Smooth int int

EXCHANGE_VALUE int tparm modify  tparm modify
EXCHANGE_VALUE int eparm modify  eparm modify
EXCHANGE_VALUE int rparm modify  rparm modify
EXCHANGE_VALUE int gparm modify  gparm modify

EXCHANGE_PEAK_CONTENTS int int_S_idx1  int_S_idx2

FOUND  int  parm undo

GET_STACK  int  parm reason

IN_RANGE  int  i_var i_low i_high
IN_RANGE  int  d_var d_low d_high
```

```
IN_RANGE  int  i_var d_low d_high
IN_RANGE  int  i_var i_low d_high
IN_RANGE  int  i_var d_low i_high
IN_RANGE  int  d_var i_low i_high
IN_RANGE  int  d_var i_low d_high
IN_RANGE  int  d_var d_low i_high


Intens_Smooth int int


KNOWN  int parm undo


None_Operation void


PUT_STACK int


Perturbation int parm undo


REMOVE int parm undo


Right_Dire  int int


SIGN int  int
SIGN int  double


Set  int  t undo  i
Set  int  e undo  d
Set  int  e undo  i
Set  int  r undo  c
Set  int  g undo  s
```

```
UNKNOWN  int parm undo

UNFOUND int parm  undo

Zone_Dist  doub int int

round int doub

RESERVE int parm undo

Push_Line int  k_stack undo i_S_idx i_pkf_idx

Pop_Line  i_pkf_idx k_stack undo  i_S_idx

Maximum  int  int int
Maximum  doub doub doub
Maximum  doub doub int
Maximum  doub int doub

Minimum int int int
Minimum doub doub doub
Minimum doub doub int
Minimum doub int doub

OR int int int

STOP int int
```

```
>=   int  i i

>=   int  d d

>=   int  i d

>=   int  d i


<=   int  i i

<=   int  d d

<=   int  i d

<=   int  d i



Select int  i  i  d  d


STUCK  int i


INT_FLOOR int doub


INT_CEIL  int doub


DOUB  doub int


RESERVE int parm undo char_invoketype


SHOW_WM  int


Pop_Top_UPT_Candidate int k_stack undo  parm_S_idx undo parm_Pkf_idx undo
%get the top line which is above three line from the k_stack,
%put its S_idx in parm_S_idx, put its Pkf_idx in parm_Pkf_idx
%if can't find any, return 0
```

```
Get_Line_Num  int_num  k_stack undo  i_S_idx
%get the number of the lines which S_idx is i_S_idx from the stack

SHOW_WM_STORE int int

INT int doub

Transfer_Line_R_To_P int int_S_idx
Transfer_Line_P_To_R int int_S_idx



Transfer_R_To_P int
Transfer_P_To_R int

P_Extension int
R_Extension int

Up_Search_Done int
Down_Search_Done int
Up_And_Down_Extension int

R_Delta2_Smooth int int_S_idx
P_Delta2_Smooth int int_S_idx

Push_CD int k_CD_Candidate_Stack undo i_K i_n s_Symm
Pop_CD  int k_CD_Candidate_Stack undo

Transfer_Line_R_To_Q  int int_S_idx
```

```
RPQ_Confirm  int int_S_idx


In_Stack int  k_Line_Candidate_Stack undo i_S_idx  i_Pkf_idx
%if line i_Pkf_idx is already in stack (with i_S_idx),  return 1
#
```

B.5 _.prop file

```
%
% May 24, 1996                    jin.prop
% recording all the legal property
%

VAR  tparm i  i  i
VAR  eparm d  d  d
VAR  eparm d  d  i
VAR  eparm d  i  d
VAR  eparm i  d  d
VAR  eparm i  i  d
VAR  eparm i  d  i
VAR  eparm d  i  i

VAR  tparm i i
VAR  eparm d d
VAR  eparm i i
VAR  eparm i d
VAR  eparm d i

TRY

MATCH parm int

INVOKE

Relative char int
```

B.6 parm.const file

```
% May 24, 1996                                   parm.const
%
% parm_name   type  value

INTENS_ADJUST_LIMIT d 0.1

CLOSE_DEFI d  0.15

ERROR_TOLERANCE d 0.001

FAR_MORE_DEFI d 4

I_EXT d  0.2

I_EXT_FOR_WEAK_PEAK d 0.2

MAX_DELTA2   d   0.03

MAX_SERIES_NUM  i 50

MAX_BIAS  d  0.006

MINI_DELTA2  d   0.01
%before was 0.0001

MINI_S_L_NUM  i  10

RESL  d 0.002
```

```
S_RANGE1 d 0.2

SEARCH_EXT1 i 20

SEARCH_WV_EXT  d 0.01

STRONG_PEAK_INTENS d 0.4

ROUGH_2B  d  1.4

R_P_MATCH_ET d 0.002

RPQ_CONFIRM_ET  d 0.002
#
```

Appendix C

Inference Engine Diagrams

The diagrams shown in this appendix explain the inference engine processes.

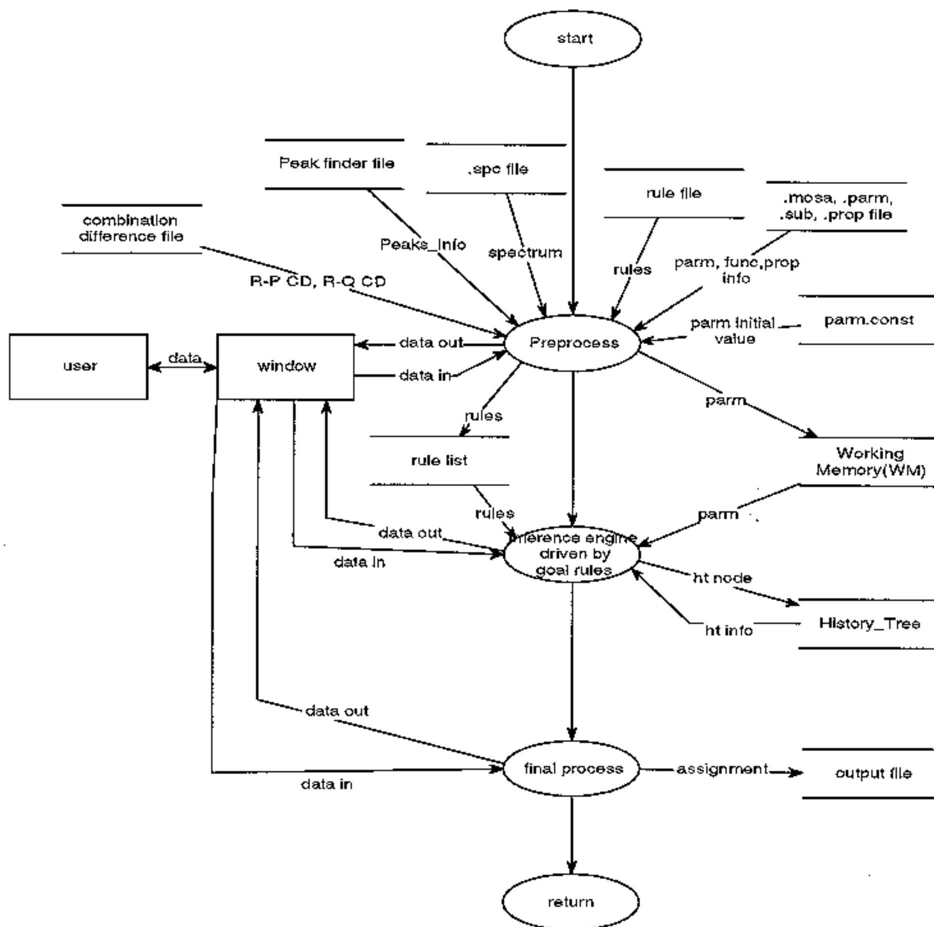

Figure C.1: mosaa Toplevel.

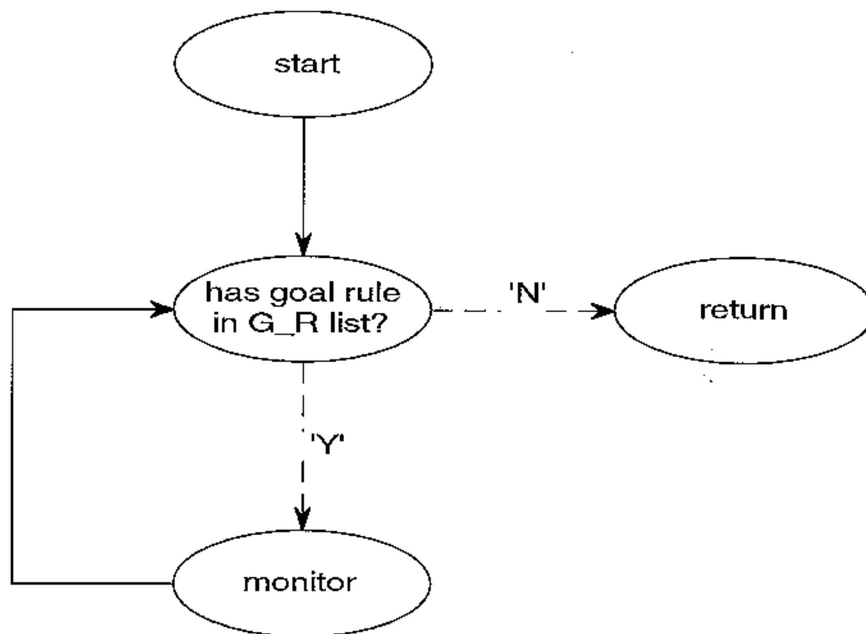

Figure C.2: Inference engine.

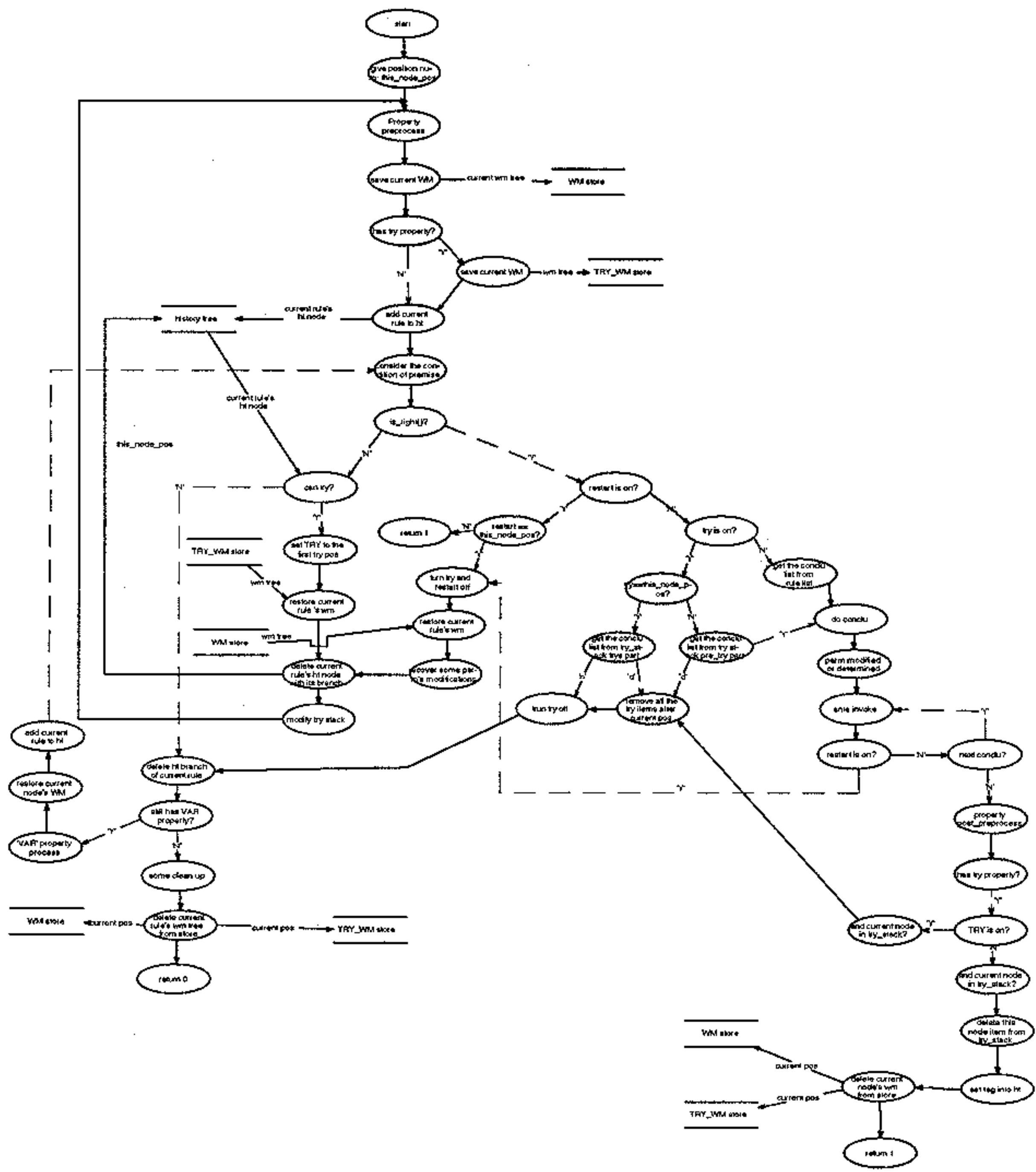

Figure C.3: Monitor mechanism.

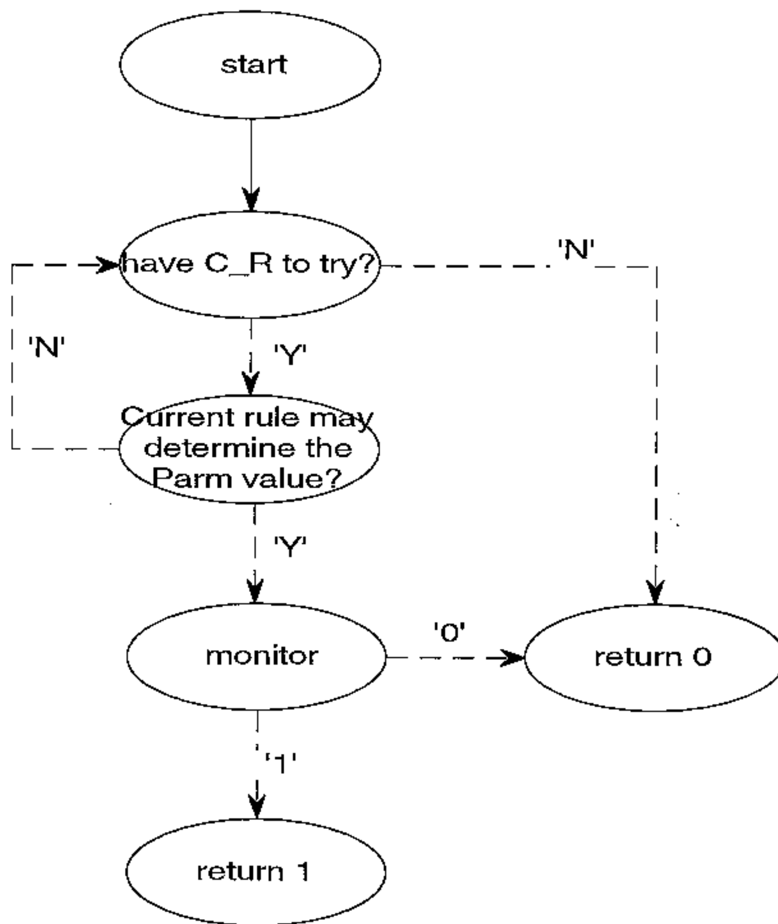

Figure C.4: Findout mechanism.

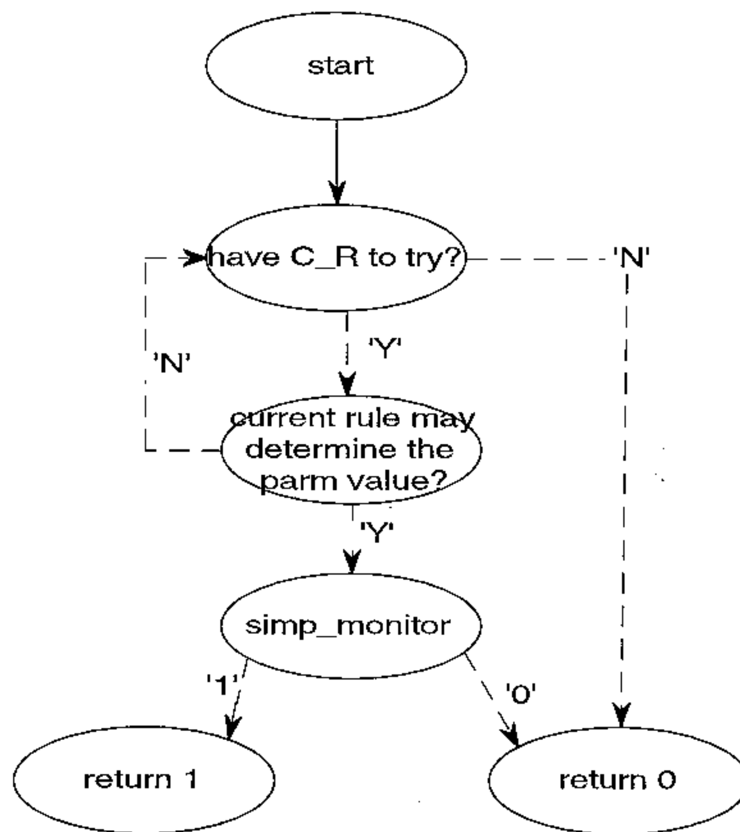

Figure C.5: Simple findout machanism.

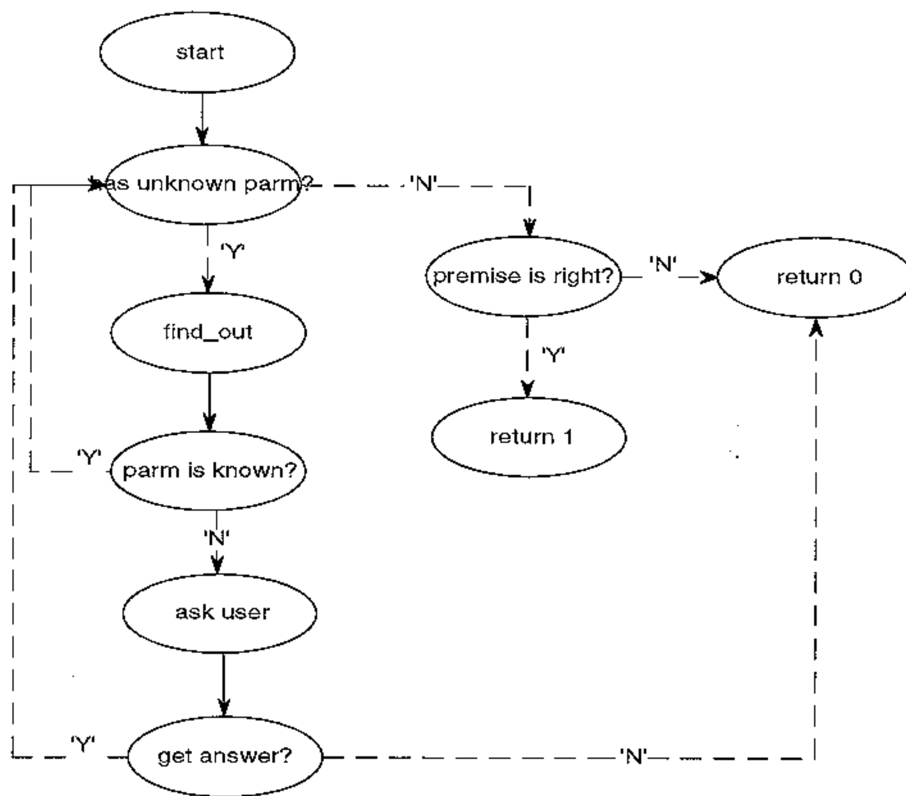

Figure C.6: Procedure 'is_right'-determines if the premise of a consequence rule is right.

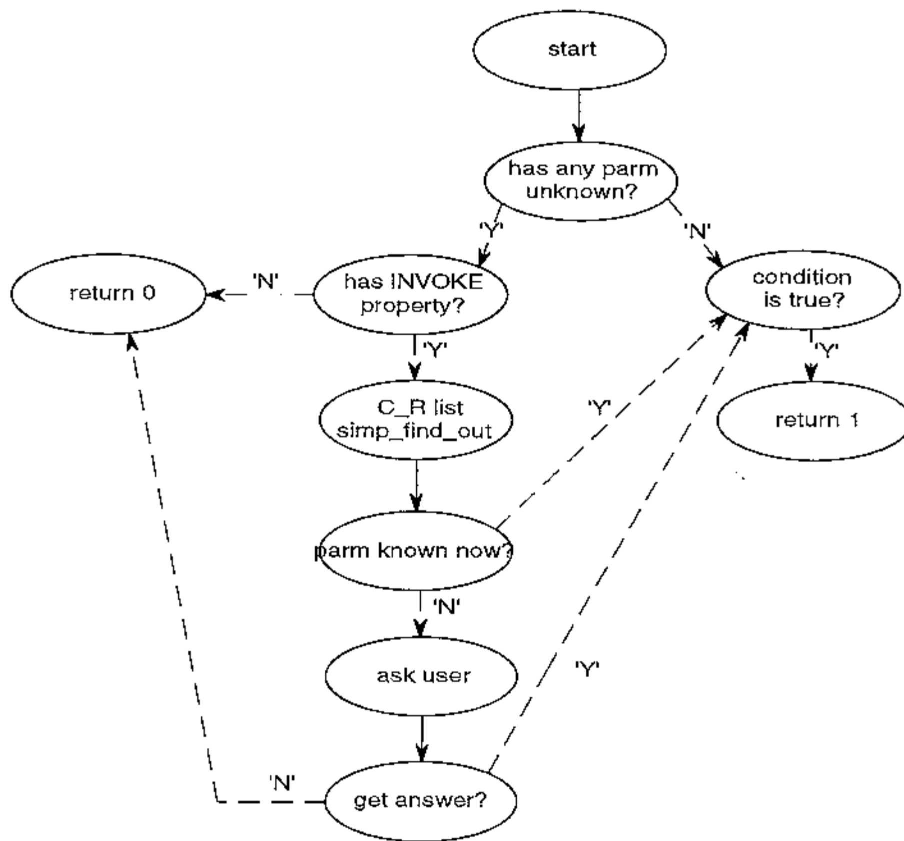

Figure C.7: Procedure 'ante_is_right'-determines if the premise of an antecedent rule is right.

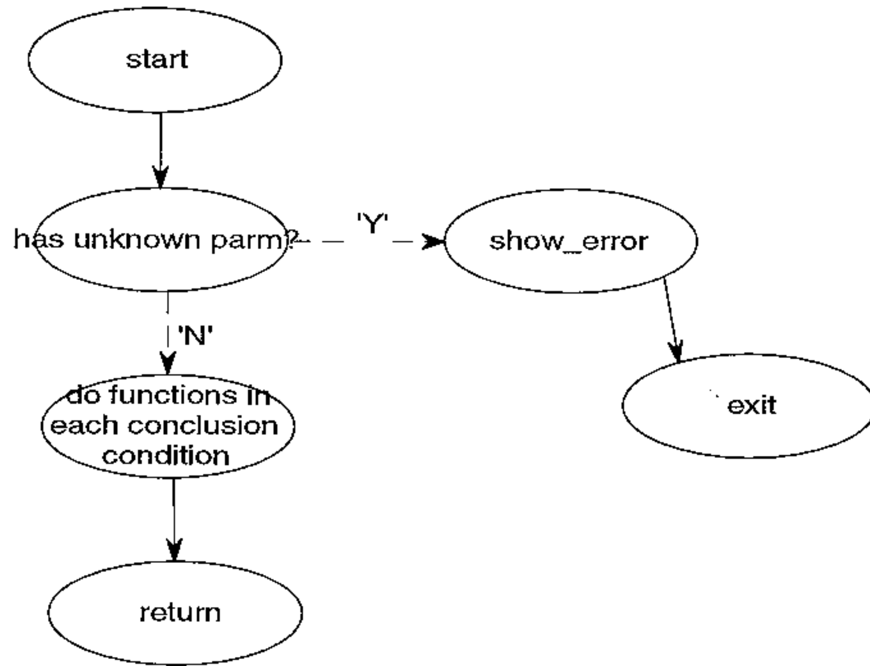

Figure C.8: Do conclusion procedure- 'doconclu'.

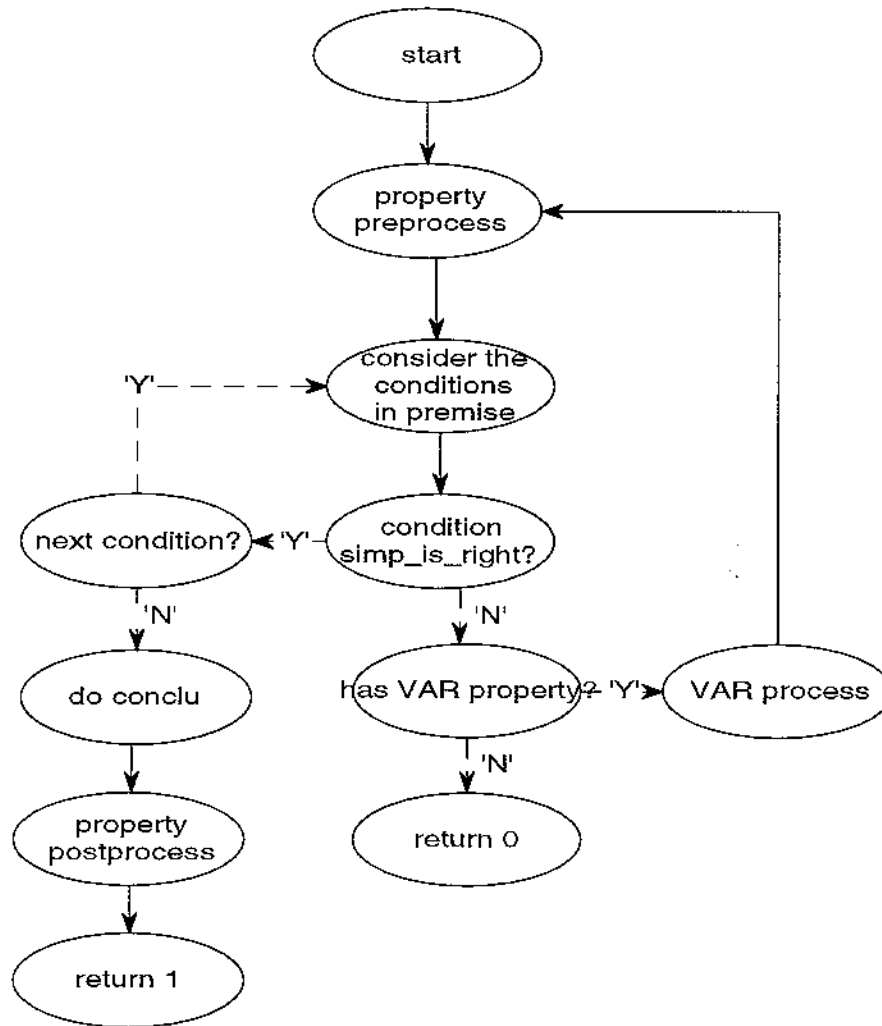

Figure C.9: Simple monitor machanism.

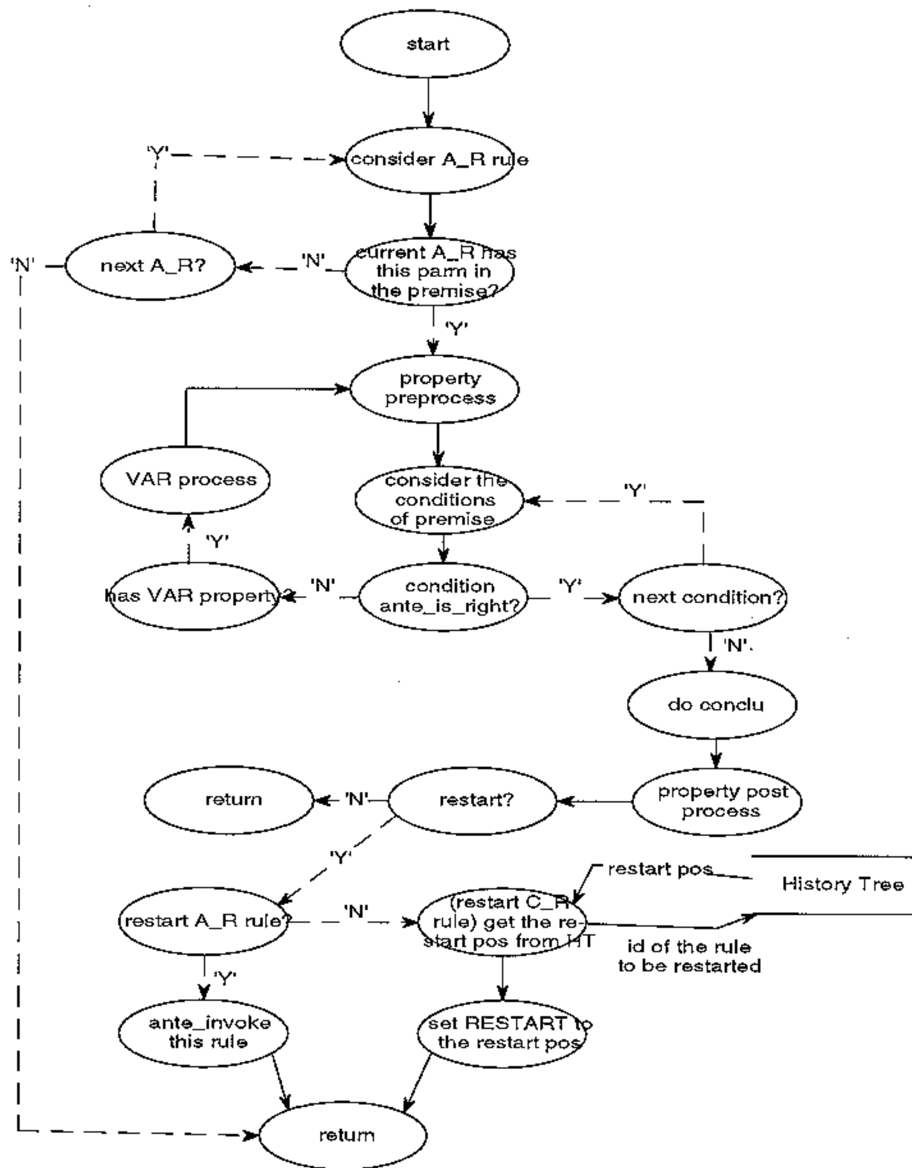

Figure C.10: Antecedent rule invoking procedure-'ante_invoke'.

Appendix D

Output file

This appendix gives some sample output files of `mosaa`, which shows the assigned R, P, Q branches.

K=3 n=0 E2

R_branch

J	wv	delta1	delta2	intens	Comm
3.	1014.063795	1.462042		0.703548	
4.	1015.525837	1.445112	-0.016930	0.549469	
5.	1016.970949	1.428034	-0.017078	0.447692	
6.	1018.398983	1.410948	-0.017086	0.384300	
7.	1019.809931	1.393699	-0.017249	0.095409	
8.	1021.203630	1.376558	-0.017141	0.313295	
9.	1022.580188	1.359246	-0.017312	0.294261	c
10.	1023.939434	1.341988	-0.017258	0.278582	c
11.	1025.281422	1.324591	-0.017397	0.277325	c
12.	1026.606013	1.307208	-0.017383	0.240680	c
13.	1027.913221	1.289959	-0.017249	0.282682	c
14.	1029.203180	1.272000	-0.017959	0.179601	c
15.	1030.475180	1.254359	-0.017641	0.306190	c
16.	1031.729539	1.237189	-0.017170	0.107564	
17.	1032.966728	1.221357	-0.015832	0.103519	
18.	1034.188085	1.199632	-0.021725	0.287611	
19.	1035.387717	1.183786	-0.015846	0.403163	
20.	1036.571503	1.165913	-0.017873	0.440108	
21.	1037.737416	1.148132	-0.017781	0.488663	
22.	1038.885548	1.129947	-0.018185	0.517341	
23.	1040.015495	1.111996	-0.017951	0.546689	
24.	1041.127491	1.093950	-0.018046	0.617380	
25.	1042.221441	1.075936	-0.018014	0.619836	
26.	1043.297377	1.057627	-0.018309	0.652444	
27.	1044.355004	1.038988	-0.018639	0.723110	
28.	1045.393992	1.021708	-0.017280	0.600252	
29.	1046.415700	1.002976	-0.018732	0.805025	
30.	1047.418676	0.984776	-0.018200	0.777478	
31.	1048.403452	0.966106	-0.018670	0.962907	
32.	1049.369558	0.947687	-0.018419	0.886564	
33.	1050.317245	0.930000	-0.017687	0.883800	
34.	1051.247245	0.911566	-0.018434	0.905029	
35.	1052.158811	0.892818	-0.018748	0.935280	
36.	1053.051629	0.875445	-0.017373	0.866443	
37.	1053.927074			0.921684	

P_branch

J	wv	delta1	delta2	intens	Comm
4.	1001.758869	-1.613570		0.625149	
5.	1000.145299	-1.630187	-0.016617	0.544680	
6.	998.515112	-1.646813	-0.016626	0.458841	
7.	996.868299	-1.663408	-0.016595	0.398014	
8.	995.204891	-1.679597	-0.016189	0.345711	
9.	993.525294	-1.696925	-0.017328	0.211162	
10.	991.828369	-1.713028	-0.016103	0.295617	c
11.	990.115341	-1.729529	-0.016501	0.286274	c
12.	988.385812	-1.745999	-0.016470	0.285428	c
13.	986.639813	-1.762406	-0.016407	0.279635	c
14.	984.877407	-1.778985	-0.016579	0.288940	c
15.	983.098422	-1.795268	-0.016283	0.090683	c
16.	981.303154	-1.811660	-0.016392	0.298264	c
17.	979.491494	-1.828076	-0.016416	0.318321	
18.	977.663418	-1.844396	-0.016320	0.358370	
19.	975.819022	-1.860742	-0.016346	0.403479	
20.	973.958280	-1.877103	-0.016361	0.436517	
21.	972.081177	-1.893440	-0.016337	0.457209	
22.	970.187737	-1.909785	-0.016345	0.509444	
23.	968.277952	-1.926099	-0.016314	0.544925	
24.	966.351853	-1.942369	-0.016270	0.602421	
25.	964.409484	-1.958727	-0.016358	0.634518	
26.	962.450757	-1.975040	-0.016313	0.676681	
27.	960.475717	-1.991359	-0.016319	0.719350	
28.	958.484358	-2.007773	-0.016414	0.747885	
29.	956.476585	-2.023982	-0.016209	0.787944	
30.	954.452603	-2.040600	-0.016618	0.822818	
31.	952.412003	-2.057004	-0.016404	0.853332	
32.	950.354999	-2.073563	-0.016559	0.878609	
33.	948.281436	-2.090213	-0.016650	0.903961	
34.	946.191223	-2.106787	-0.016574	0.898987	
35.	944.084436	-2.123929	-0.017142	0.937505	
36.	941.960507	-2.148020	-0.024091	0.944509	
37.	939.812487			0.945380	

Q_branch

J	wv	delta1	delta2	intens	Comm
8.	1007.439937	-0.151527		0.188149	
9.	1007.288410	-0.168356	-0.016829	0.191218	c
10.	1007.120054	-0.184881	-0.016525	0.779114	c
11.	1006.935173	-0.202730	-0.017849	0.620641	c
12.	1006.732443	-0.220074	-0.017344	0.835427	c
13.	1006.512369	-0.235163	-0.015089	0.530766	c
14.	1006.277206	-0.253645	-0.018482	0.878013	c
15.	1006.023561	-0.269397	-0.015752	0.800137	c
16.	1005.754164			0.505736	c