THE ALDAT LANGUAGE

by

Jeff Parkovnick

In Conjunction with

T.H. Merrett

Bob Reckhow

Ragui Kamel

Lily Lam

A

good use of examples

~1976?

yoo 308-621

The limerick packs laughs anatomical
Into space that is quite economical
  But the good ones I've seen
  So seldom are clean,
And the clean ones so seldom are comical.

1 Introduction

This paper is a language manual for ALDAT- An Algebraic Database Language, proposed by T.H. Merritt and designed by the class of Compilier Construction at McGill University. This version is a prototype, to be used as a model for further work in this field.

2 Syntax of ALDAT-1

Backus-Naur Form (BNF)

Note: the following symbols are meta-symbols belonging to the BNF formalism, and not symbols of the language ALDAT.

```
          ::=   |   {   }
```

The curly brackets denote possible repetition of the enclosed symbols zero or more times. In general,

```
          A ::= { B }
```

is a short form for the purely recursive rule:

```
          A ::= <EMPTY> | AB
```

BNF

```
1   <PROGRAM> ::= MAIN <PROGRAM HEADING> <BLOCK> EOF _|_
2   <PROGRAM HEADING> ::= <IDENTIFIER> ;
3                       | <IDENTIFIER> ( <RELATION LIST> { ; <RELATION LIST> } ) ;
4   <RELATION LIST> ::= <RELATION NAME> {, <RELATION NAME> } : <RELATION TYPE>
5   <RELATION NAME> ::= <IDENTIFIER>
6   <RELATION TYPE> ::= INPUT
7                     | OUTPUT
8                     | UPDATE
9   <BLOCK> ::= <MACRO DECLARATION PART> <DOMAIN DECLARATION PART>
10              <VARIABLE DECLARATION PART> <DOMAIN ALGEBRA PART>
11              <STATEMENT PART>
```

```
0   <MACRO DECLARATION PART> ::= <EMPTY>
1                              | MACRO <MACRO DEFINITION> {; <MACRO DEFINITION> } ;
2   <MACRO DEFINITION> ::= <MACRO NAME> = <STRING>
3   <MACRO NAME> ::= <IDENTIFIER>
4   <DOMAIN DECLARATION PART> ::= <EMPTY>
5                               | DOMAIN <DOMAIN DEFINITION> {; <DOMAIN DEFINITION> } ;
6   <DOMAIN DEFINITION> ::= <DOMAIN NAME> {, <DOMAIN NAME> } : <COMPONENT TYPE>
7   <DOMAIN NAME> ::= <IDENTIFIER>
8   <COMPONENT TYPE> ::= <TYPE IDENTIFIER>
9                      | ARRAY ( <SIGN> <UNSIGNED INTEGER> .. <SIGN> <UNSIGNED INTEGER> )
                               OF <TYPE IDENTIFIER>
20  <TYPE IDENTIFIER> ::= INTEGER
21                      | CHARACTER
22                      | BOOLEAN
23  <SIGN> ::= +
24           | -
25           | <EMPTY>
26  <VARIABLE DECLARATION PART> ::= <EMPTY>
27                    | VAR <VARIABLE DEFINITION> {; <VARIABLE DEFINITION> } ;
28  <VARIABLE DEFINITION> ::= <VARIABLE OR RELATION NAME>
                               {, <VARIABLE OR RELATION NAME> } : <TYPE>
29  <VARIABLE OR RELATION NAME> ::= <IDENTIFIER>
30  <TYPE> ::= <COMPONENT TYPE>
31           | RELATION ON ( <DOMAIN NAME> {, <DOMAIN NAME> } ) <RELATION SIZE>
32  <RELATION SIZE> ::= <EMPTY>
33                    | SIZE = <ARITHMETIC EXPRESSION>
34  <DOMAIN ALGEBRA PART> ::= <EMPTY>
35               | LET <DOMAIN ALGEBRA SECTION> {; <DOMAIN ALGEBRA SECTION>} ;
```

```
6   <DOMAIN ALGEBRA SECTION> ::= <DOMAIN NAME> BE <DOMAIN ALGEBRA EXPRESSION>
7   <DOMAIN ALGEBRA EXPRESSION> ::= <BOOLEAN EXPRESSION>
8                  | <ASSOCIATIVE OPERATOR> OF <BOOLEAN EXPRESSION>
9                  | <ASSOCIATIVE OPERATOR> OF <BOOLEAN EXPRESSION> FOR <DOMAIN NAME>
0                  | <BINARY OPERATOR> ON <BOOLEAN EXPRESSION> FOR
                            <ARITHMETIC EXPRESSION> PREV <DOMAIN NAME>
1                  | <BINARY OPERATOR> ON <BOOLEAN EXPRESSION> FOR
                            <ARITHMETIC EXPRESSION> NEXT <DOMAIN NAME>
2                  | <BOOLEAN EXPRESSION> FOR <ARITHMETIC EXPRESSION> PREV <DOMAIN NAME>
3                  | <BOOLEAN EXPRESSION> FOR <ARITHMETIC EXPRESSION> NEXT <DOMAIN NAME>
4   <BOOLEAN EXPRESSION> ::= <LOGICAL TERM>
5                        | <BOOLEAN EXPRESSION> | <LOGICAL TERM>
6   <LOGICAL TERM> ::= <LOGICAL FACTOR>
7                  | <LOGICAL TERM> & <LOGICAL FACTOR>
8   <LOGICAL FACTOR> ::= <LOGICAL PRIMARY>
9                    | ¬ <LOGICAL FACTOR>
50  <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION>
51                    | <ARITHMETIC EXPRESSION> = <ARITHMETIC EXPRESSION>
52                    | <ARITHMETIC EXPRESSION> < <ARITHMETIC EXPRESSION>
53  <ARITHMETIC EXPRESSION> ::= <TERM>
54                          | <ARITHMETIC EXPRESSION> + <TERM>
55                          | <ARITHMETIC EXPRESSION> - <TERM>
56  <TERM> ::= <FACTOR>
57         | <TERM> * <FACTOR>
58         | <TERM> DIV <FACTOR>
59         | <TERM> MOD <FACTOR>
60  <FACTOR> ::= <BASE>
61           | <BASE> EXP <FACTOR>
62  <BASE> ::= <PRIMARY>
63         | ( <DOMAIN ALGEBRA EXPRESSION> )
64         | + <BASE>
65         | - <BASE>
```

```
66   <PRIMARY> ::= <DOMAIN OR VARIABLE NAME>
67                 | <DOMAIN OR VARIABLE NAME> ( <ARITHMETIC EXPRESSION> )
68                 | <UNSIGNED INTEGER>
69                 | <STRING>
70   <DOMAIN OR VARIABLE NAME> ::= <IDENTIFIER>
71   <ASSOCIATIVE OPERATOR> ::= |
72                            | &
73                            | =
74                            | +
75                            | *
76                            | MAX
77                            | MIN
78                            | <FUNCTION NAME>
79   <FUNCTION NAME> ::= <IDENTIFIER>
80   <BINARY OPERATOR> ::= <ASSOCIATIVE OPERATOR>
81                       | -
82                       | DIV
83                       | MOD
84                       | EXP
85   <STATEMENT PART> ::= BEGIN <STATEMENT LIST> END
86   <STATEMENT LIST> ::= <STATEMENT> {; <STATEMENT> }
87   <STATEMENT> ::= <STATEMENT PART>
88                 | <VARIABLE NAME> := <BOOLEAN EXPRESSION>
89                 | <VARIABLE NAME> ( <ARITHMETIC EXPRESSION> ) := <BOOLEAN EXPRESSION>
90                 | <RELATION NAME> :+ <RELATIONAL EXPRESSION>
91                 | IF <BOOLEAN EXPRESSION> THEN <STATEMENT> ELSE <STATEMENT>
92                 | WHILE <BOOLEAN EXPRESSION> DO <STATEMENT>
93   <RELATIONAL EXPRESSION> ::= " <BOOLEAN EXPRESSION> {, <BOOLEAN EXPRESSION> } "
94                             | <RELATIONAL OPERATION>
95   <RELATIONAL OPERATION> ::= <RELATION>
96                            | <RELATION> <| <DOMAIN NAME> {, <DOMAIN NAME> }
                                   <RELATIONAL OPERATOR> <DOMAIN NAME> {, <DOMAIN NAME> }
                                       |> <RELATION>
```

```
97   <RELATION> ::= <RELATION NAME>
98              | <RELATIONAL OPERATION>   ← with #95, this is circular
99              | <RELATION> <| <DOMAIN NAME> {, <DOMAIN NAME> } |>
100             | <RELATION> <* <BOOLEAN EXPRESSION> *>
101  <RELATIONAL OPERATOR> ::= <
102                          | <=
103                          | IJOIN
104                          | UJOIN
105                          | DJOIN
106                          | SJOIN
107                          | LJOIN
108                          | LORAN
109                          | HIRAN
110                          | LTJOIN
111                          | LEJOIN
112                          | DIVREL
```

3 Constants

Numeric constants are restricted to integers at this time. Real numbers will be implemented in a latter version of the language.

```
<UNSIGNED INTEGER> ::= <DIGIT> { <DIGIT> }
<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
```

String constants are any ~~set~~ sequence of valid ~~EPCIDIC~~ EBCDIC characters enclosed in quotes. If the string is to contain a quote mark, then this quote mark is to be written twice.

NOTE: there are no string functions implemented in this version. Functions to be implemented later include:

1) Concatenation

2) Substring

3) Byte

as in BPL.

The length of a string is limited to 256 characters.

EXAMPLE:

'HELP'　　'BUGGER OFF'　　'DON''T'

In this version, though, I thought the type CHARACTER meant single characters, so that in fact the nearest thing to strings we allow is an array of characters. Thus string constants have no use unless they are single characters.

Boolean constants are the predefined variables 'TRUE' and 'FALSE'. They are assigned using production 66 in the grammar.

EXAMPLE:

```
    A := TRUE;
```

Constant RELATIONS are formed by assigning a constant tuple to a RELATION declared in the program.

EXAMPLE:

these identifiers can't be ~~relat~~ domain names, because they are type identifiers

```
     ...
     VAR: A  RELATION ON (INTEGER,INTEGER,CHARACTER);
     ...
       BEGIN
          ...
          A:+ (4888284,6915134,'JEFF');
          ...
       END
```

your grammar says these should be "

4 Identifiers

Identifiers serve to denote constants,variables,domains and relations.

```
<IDENTIFIER> ::= <LETTER> { <CONTINUATION CHARACTER> }

<CONTINUATION CHARACTER> ::= <LETTER>

                           | <DIGIT>

                           | <BREAK CHARACTER>

<LETTER> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<BREAK CHARACTER> ::= _|#
```

RESERVED WORDS

BE
DO
IF
OF
ON
DIV
END
EOF
EXP
FOR
LET
MAX
MOD
MIN
VAR
ELSE
MAIN
NEXT
PREV
SIZE
THEN
TRUE
ARRAY
BEGIN
DJOIN
FALSE
HIRAN
IJOIN
INPUT
LJOIN
LORAN
SJOIN
UJOIN
WHILE
MACRO
DIVREL
DOMAIN
LEJOIN
LTJOIN
OUTPUT
UPDATE
RELATION

5 Comments

Comments, defined by:

```
    << ANY SEQUENCE OF SYMBOLS NOT CONTAINING ">>" >>
```

may be inserted anywhere in a program, and may be removed without altering the meaning of the program.

6 Declarations

6.1 Domain Declarations

Domain declarations, productions 14-22, allow the definition of six types of domains; scalar or array domains in INTEGER,BOOLEAN,OR CHARACTER.

EXAMPLE:

```
    DOMAIN
        K,L,N,P,X,Y : INTEGER;
        Q,R,S,M : CHARACTER;
        T,O : BOOLEAN;
        ZOO : ARRAY (-1 .. 5 ) OF INTEGER;
```

In the future, the ability to define user types (as in PASCAL) will be added, allowing the user to define his own types of domains.

6.2 Variable Declarations

Scalar variables are standard.

EXAMPLE:

```
     VAR
       A,B,C : INTEGER;
       D,E,F : BOOLEAN;
```

ARRAYS are restricted to one-dimensional vectors. Indexes are an improper ? subset of the Integers.

EXAMPLE:

```
     VAR
       A,B,C : ARRAY (-50 .. 50) OF INTEGER;
       D : ARRAY (0 .. 100) OF BOOLEAN ;
```

RELATIONS are divided into three classes:

1) Relations external to the program (ie. on a database).
   NOTE: these relations must be declared with the same domain names as used on the data base.
2) Relations declared in the program and output to the database at program's end.
3) Relations declared in the program and deleted at program's end.

Classes 1 and 2 must be declared in the parameter list of the program as either INPUT , OUTPUT, or UPDATE (productions 1-8).

Relations are declared using previously declared domains. The size (ie number of tuples), or the expected size of the relation must be given. This size can be a constant, or an expression based on the size of a relation on the database.

EXAMPLE:

Let 1) A and B be relations on the database, A with 20 tuples,
B with 40 tuples.

2) C is a relation declared in the program and is to be deleted at the end of execution

3) D is a relation declared in the program and is to be output to the database

4) The domains are as follows:

```
A(X,Y)          B(R,T,S)              C(P,Q)              D(K,L,M,N,O,P)
```

The declarations are as follows:

```
MAIN
     FRODO ( A,B : INPUT ; D : OUTPUT);
     ...
     DOMAIN
          X,Y,P,K,L,N : INTEGER;
          Q,R,S,M : CHARACTER ;
          T,O : BOOLEAN ;
      ...
      VAR
          A : RELATION ON (X,Y) SIZE = A ;
          B : RELATION ON (R,T,S) SIZE = B ;
          C : RELATION ON (P,Q) SIZE = A*2 ;
          D : RELATION ON (K,L,M,N,O,P) SIZE = A*4+2*B+10 ;
      ...
```

Since A already exists, would leaving this size specification out (prod.32) have the same effect?

NOTE: the domain names in a program must be unique.

```
ie  A : RELATION ON (X,...,X)
and C : RELATION ON (Z,...)
    D : RELATION ON (Z,...)
```

I don't see the need to make this restriction.

are illegal.

7 Domain Algebra

"Domain algebra allows the programmer to define a domain to be the result of any operation on other domains or domain-occurrences independently of any relation. The resulting domain is (and the operand domains may be) virtual domains: they exist only as the defined result of some expression in the domain algebra. These virtual domains become actual domains (with explicitly stored values) only when actualized by the programmer and then only if the operand domains are actual or can be actualized for the relation concerned. Most commonly, actualization takes place in a relational operation which generalizes projection, although explicit use of a virtual domain in a join or other operation may actualize the domain.

The operators of domain algebra fall into two categories, which may be termed horizontal and vertical. The horizontal operators work entirely within a given tuple, and produce a virtual domain which can be regarded as an extension of the tuple. Virtual operators combine values of the same domain-occurrence for many tuples."

T.H.Merrett
MRDS-An Algebraic Relational
Data Base System

The form of the domain algebra expression choosen would depend on the usage (productions 34-43).

Domains to be used in examples are as follows:

```
        DOMAIN
              TEST1,TEST2,TEST3,GRADE,STUD_NO : INTEGER;
              QUES1,QUES2,QUES3 : BOOLEAN;
              CLASS : INTEGER;
```

Values of these domains are:

TEST1	TEST2	TEST3	QUES1	QUES2	QUES3	CLASS
90	85	87	TRUE	FALSE	TRUE	1
65	70	73	FALSE	TRUE	TRUE	0
80	75	77-	TRUE	TRUE	FALSE	0
75	77	80	TRUE	TRUE	FALSE	1
80	77	83	FALSE	FALSE	TRUE	1

GRADE: 9 8 9 10 9 9 9 10 8 8 10 7

STUD_NO: 1 2 3 1 2 3 1 2 3 1 2 3

Form 1

```
LET <DOMAIN NAME> BE <BOOLEAN EXPRESSION>
```

<BOOLEAN EXPRESSION> reduces to a new domain which is then assigned to <DOMAIN NAME>.

EXAMPLE:

1) domains test1,test2,test3 contain the marks for the class for three mid terms. Find the average mark of each student.

```
LET AVERAGE BE ( TEST1+TEST2+TEST3 ) DIV 3 ;
```

2) a questonare was given the class, with the answers stored in ques1,ques2,ques3. Find all students who answered question 1 and 2 TRUE ~~and~~ or question 3 FALSE.

```
LET ANS BE QUES1 & QUES2 | ¬ QUES3 ;
```

```
AVERAGE: ( 87 69 77 77 80 )
ANS: ( FALSE FALSE TRUE TRUE FALSE )
```

For the next two forms, the operator is associative, since the tuples in a relation are in no particular order, and any other operation will have order-depentent results.

Form 2

```
LET <DOMAIN NAME> BE <ASSOCIATIVE OPERATOR> OF
   <BOOLEAN EXPRESSION> {  FOR <DOMAIN NAME> }
```

A domain is produced, which is then acted upon by the operator with or without the use of a control domain.

EXAMPLE:

1) find the maximum mark from the previos example

```
LET MAXMARK BE MAX OF AVERAGE ;
```

2) assume the class was divided into two sections, as given by class. Find the maximum mark in each section.

```
LET SECMAX BE MAX OF ( TEST1+TEST2+TEST3 ) DIV 3 FOR CLASS ;
```

3) in the questionare, did all students answer the questions in the desired manner

```
LET ALLANS BE & OF ANS ;
```

4) by section

```
LET SECANS BE & OF ANS FOR CLASS ;
```

```
MAXMARK: (87 87 87 87 87)
SECMAX: (87 77 77 87 87)
ALLANS: (FALSE FALSE FALSE FALSE FALSE)
SECANS: (FALSE FALSE FALSE FALSE FALSE)
```

Form 3

```
LET <DOMAIN NAME> BE <BINARY OPERATOR> ON <BOOLEAN EXPRESSION>
           <arithmetic expression>
    & FOR ^ PREV <DOMAIN NAME>
           ,
          ____
          NEXT
```

In the next two forms, the operator class has been expanded since in historical reduction, the tuples are assumed to have an order.

This form allows operations on n tuples forwards for backwards in the domain.

EXAMPLE:

1) a class has weekly tests, with marks put in a relation with domains grade and stud_no. Find the average for two consectutive test thru the term

```
LET PROG BE ( + ON GRADE FOR 2 PREV STUD_NO ) DIV 2 ;

PROG: ( 9 8 9 9 8 9 9 9 8 8 10 7 )
```

see note on the next page

Form 4

```
LET <DOMAIN NAME> BE <BOOLEAN EXPRESSION> FOR
 <ARITHMETIC EXPRESSION>  PREV  <DOMAIN NAME>
                          NEXT
```

The domain formed is shifted up or down n tuples after being sorted by the control domain.

EXAMPLE:

1) sort the domain grade so that the first n tuples are the first students marks,etc

```
 LET MARKS BE GRADE FOR 0 PREV STUD_NO ;


MARKS: ( 9 10 9 8 - 8 9 10 10 - 9 9 8 7 )
```

this does nothing, but set MARKS ≡ GRADE. Remember: the order of tuples in a relation is irrelevant.

Space precludes further discussion here.

I believe it was always Merretts assumption that the controlling domain (STUD_NO in these examples) in historical reduction must be a key. That is, values of this domain must be unique, so that the tuples can be unambiguously sorted. To handle your examples, ~~a new~~ another domain TEST_DATE must be included, and the STUD_NO ~~must~~ would have to be combined with it somehow to get a unique key for each ~~mar~~ GRADE.

8 . Expressions

There are two levels of expressions implemented in ALDAT. The first level is standard arithmetic expressions, similar to those in PASCAL or BPL. The second level is that of relational expressions, the "raison d'etre" for ALDAT.

8.1 Arithmetic Expressions

As mentioned before, string functions have not be included in this version of the language. In addition, only integer values are valid.The priority of existing operators is as follows:

```
Unary + and -
EXP     (exponentiation)
*  AND  DIV  (integer division)
+  and  -
<  and  =
¬
&
1
```

EXAMPLES:

```
A + B * (A DIV 3 )
A & B & - C
(A EXP 2) + (B EXP 2) - 20
```

8.2 Relational Expressions

Relational expressions are used to preform bulk operations on files in a database. Possible operations are:

A) Projection

Projection is the act of assigning domains from one relation to another relation. In this transfer, duplicate tuples are eleminated. (production 99)

EXAMPLE:

```
A <| X,Y |>
```

if X = (2 2 3 4)

and Y = (2 2 4 6)

then the resulting relation will have two domains with values (2 3 4) and (2 4 6)

B RESTRICTION

In restriction a boolean virtual domain is used to eliminate unwanted tuples from the relation. If element (i) is true, the corresponding tuple is kept, otherwise it is ignored.

EXAMPLE:

Using test1 and test2 defined on page 15, and Q, a virtual domain defined as:

```
LET Q BE (TEST1+TEST2)<80;
```

If MARKS is the relation containing test1 and test2 then

```
MARKS <* Q *>
```

will result in a relation containing students whose marks are less than 80 (ie tuples 2 3 4 5).

C) CONSTANT TUPLE

Used to add a tuple to the end of a relation. See example on page 8.

D) Joins

A list of possible joins is given on page 6. The syntax of such an expression is given on page 5, production 96. Further joins will be implemented in future versions of ALDAT¬

NOTE: in a relation3l expression, a relation can appear only once. Also domain names in the program must be unique.

9 Statements

9.1 Assignment statements

Assignment statements serve to replace the current value of a variable by a new value specified by an expression. Relational assignment statements use a different syntax (:+ instead of :=), and they specify addition, rather than replacement, of tuples.

EXAMPLE:

```
A :=B;
A( 5 ):= X EXP 2 ;
H :+ G <| D,E |> ;
```

9.2 IF - Statement

The if statement specifies that a statement be executed only if a certain condition is true.

EXAMPLE:

```
IF A=1 then B:=2
        ELSE B:= 1
```

9.3 While statement

The while statement specifies that a statement be executed until a certain condition is found to be false.

EXAMPLE:

```
WHILE A<10 do A:=A+1
```

There was a young lady called Gloria
Who was had by a Major Pretoria.
She was had by more men
And then had again
By the Band of the Waldorf Astoria