A Short Summary of the Evaluate Modelling Language

William S. Page
Structural Analysis Division
Statistics Canada
January 2, 1987.

Introduction

Evaluate is a vector-oriented modelling language. It is based on the algebraic properties of the category of ordinary sets and set-functions. For more details on the design of the language see "Extended Vector Algebra Language EVALUATE", Structural Analysis Division Working paper, William S. Page, June 28, 1983.

This version of Evaluate is written in "C" and is implemented under Eunice (a UNIX shell for VMS) on a VAX 8600 and under the MSDOS operating system on a Compaq 286 micro-computer. It consists of two levels. The upper level handles the high-level language including variable definitions and scenario (variable value) management. High-level language statements are translated to low-level shell command sequences. These command sequences consist of a number of filters which implement the low-level Evaluate data manipulations. The following is a short summary of the Evaluate high-level language. A brief tutorial with examples will be available shortly.

Vectors are defined in terms of domains. Domains represent sets. In programming languages such as Pascal, domains would be referred to as data types.

A Short Summary of the Evaluate Modelling Language

Table of Contents

A Short Summary of the Evaluate Modelling Language

1) Domain Constants

Name	Syntax	Description
real numbers	REAL	real (floating point) numbers
titles	TITLE	character strings
zero	0	empty set
unit	1	unnamed singleton set

2) Domain Expressions

For any domains A B C etc.

Name	Syntax	Description
pairing	(A,B,C ...)	cross-product, set of tuples
	(a:A,b:B,c:C ...)	labelled pairing
glueing	(A;B;C ...)	disjoint union
	(a:A;b:B;c:C ...)	labelled glueing
vector space	(A->B)	set of vectors
title set	{name}	subset (e.g. name:50->TITLE)
grouping set	[name]	partition (e.g. name:SIC->50)
enumeration	2	= (1;1)
etc.	3	= (1;1;1) etc.

3) Elements

The elements of domain A belong to vector space (1->A).

4) Data types

A pairing is similar to the notion of a record data type in programming languages such as Pascal. A glueing is similar to a variant record data type. A vector space is similar to an array data type.

5) Vectors and arrays

Vectors are strongly-typed total functional mappings from a subject domain to an object domain. Vectors are a generalization of the notion of an array in programming languages such as Pascal. The subject and object of a vector are given by a notation called a signature:

```
name:subject->object
```

6) **Signature**

The signature of a vector defines the vector space to which the vector belongs. In Pascal this would correspond to the data type:

```
var name:array [subject] of object;
```

7) **Matrix**

A vector whose subject is a pairing and whose object is REAL, corresponds to the usual notion of a matrix. For example,

```
matrix:(row,column)->REAL
```

8) **Time series**

A time series is a vector whose subject is a time title set, i.e. a vector with signature n->REAL and whose object is REAL. For example, suppose

```
t81t90:10->REAL
```

is a vector containing 1981;1982; ... 1990, then

```
tpop:{t81t90}->REAL
```

might be a time series representing the total population from 1981 to 1990 in one year increments.

9) **Vector constants**

Examples	Signature	Description
3.141592	1->REAL	real number
"Iron Ore"	1->TITLE	title

9.1) **Projections and Injections**

For any domains A B etc., associated with the pairing or glueing of A and B, we have projection and injection vectors denoted by <0> <1> etc. and |0| |1| etc., respectively.

Examples	Signature	Description
<0>	(A,B)->A	projection vector
<1>	(A,B)->B	projection vector
|0|	A->(A;B)	injection vector
|1|	B->(A;B)	injection vector
|1| |2| |3| etc.	1->number	element of enumeration
<a>	(a:A,b:B)->A	labelled projection
|a|	A->(a:A;b:B)	labelled injection

Projections and injections are used with composition to form selection and restriction operators. See below.

10) **Vector expressions**

New vectors can be formed from known vectors by writing vector expressions. Each vector expression gives rise to domain expressions in the signature of the resulting vector.

11) **Composition**

For any domains A, B, C and vectors f:A->B g:B->C the expression

    g f

(read g of f) denotes a vector with signature A->C. Composition is associative, i.e. for any domain D and vector h:C->D

  h (g f) = (h g) f

Composition has left and right idenities.

12) **Right identity**

For any domains A B and vector f:A->B, the expression

    sbj f

denotes the right composition identity with signature A->A.

    f(sbj f) = f

13) **Left identity**

For any domains A B and vector f:A->B, the expression

    obj f

denotes the left composition identity with signature B->B.

    (obj f)f = f

13) **Pairing**

For any domains A B C and vectors f:A->B g:A->C the expression

    f,g

denotes a vector with signature A->(B,C).

14) **Tupleing**

More generally, for domains A B1 B2 B3 etc and vectors f1:A->B1 f2:A->B2 f3:A->B3 ...

```
f1,f2,f3, ...
```

denotes a vector with signature A->(B1,B2,B3, ...)

15) **Selection**

Composition with a project vector is called selection. For a vector x:A->(B,C) the composition

```
<0>x
```

denotes a vector with signature A->B. <0> denotes the projection vector whose signature depends on the context. In this case its signature is <0>:(B,C)->B.

Similarily

```
<1>x
```

denotes a vector with signature A->C, i.e. the composition with projection <1>:(B,C)->C.

16) **Empty projection**

Composition with the empty projection

```
<>x
```

denotes the unique vector A->1.

Pairing and selection are related by

```
<0>(f0,f1) = f0
<1>(f0,f1) = f1
```

These equations extend to tuples.

17) **General pairing (type coercion)**

For any domains A, B, C, and D where A and B represent different domains and vectors f:A->C and g:B->D, the expression

```
f,g
```

denotes a vector with signature (A,B)->(C,D) defined by

```
(f id<0>),(g id<1>)
```

where id = obj((sbj f)||,(sbj g)||).

18) **Labelled pairing**

For domains A, B, C and vectors f:A->B and g:A->C the expression

```
a:f,b:g
```

denotes the vector with signature A->(a:B,b:C)

Similarily, for domains A, B, C, and D with A not the same as B and vectors f:A->C and g:B->D

```
a:f,b:g
```

denotes a vector with signature (a:A,b:B)->(a:C,b:D) defined as above.

19) **Labelled selection**

<a> denotes a projection vector from a labelled pairing. Thus,

```
<a>(a:f,b:g)=f
<b>(a:f,b:g)=g
```

In programming languages with record structures, for example in Pascal,

```
var f: record a:A; b:B; c:C; end;
```

labelled selection corresponds to the 'dot' notation

```
f.a
```

20) **Glueing**

Glueing and restriction are analogous to projection and selection. For domains A, B, C, D with C and D different domains and vectors f:A->C g:B->C and h:B->D we have following glueing expressions

Expression	Signature	Definition
f;g	(A;B)->C	
g1;g2;g3	(B1;B2;B3)->C	
a:f;b:g	(a:A;b:B)->C	
f;h	(A;B)->(C;D)	(<0>id g);(<1>id e)
a:f;h:e	(a:A;b:B)->(a:C;b:D)	

where id = sbj(<>(obj g);<>(obj e))

21) **Restriction**

For domains A, B, C and vector z:(A;B)->C, we have the following restriction expressions

Expression	Signature	Definition
z|0|	A->C	composition with injection A->(A;B)
z|1|	B->C	composition with injection B->(A;B)
z||	0->C	empty injection

Glueing and restriction are related by

```
(g0;g1)|0| = g0
(g0;g1)|1| = g1
```

or with labels

```
(a:g0;b:g1 ... )|a|=g0
etc.
```

22) **Nesting**

For domains A B C and vector x:(A,B)->C the expression

```
x<0>
```

denotes a vector with signature A->(B->C) i.e. a vector whose elements are themselves vectors (elements of a vector space).

Similarily, the expression

```
x<1>
```

denotes a vector with signature B->(A->C).

23) **Flattening**

For any vector w:A->(B->C)

```
flat w
```

denotes the vector with signature (A,B)->C.

Nesting and flattening are related by

```
(flat w)<0> = w
flat(x<0>) = x
```

24) **Naming**

For any vector f:A->B, the expression

```
f<>
```

denotes a vector with signature 1->(A->B) which names the vector as an element of vector space.

25) **Transpose**

Thinking of the vector x:(A,B)->C as a matrix, we may form the transpose of x by composition with paired projections. The expression

```
x<1,0>
```

denotes the vector with signature (B,A)->C.

25.1) **Interchange**

Given a vector whose object is a glueing, x:A->(B;C) we may interchange the order of the components of the object by compostion with glued injections. The expression

```
|1,0|x
```

denotes the vector with signature A->(C;B)

26) **Equalizer**

For any domains A and B and vectors f:A->B and g:A->B the expression

```
f=g
```

denotes a vector with signature n->A which points out the largest subset of A over which f and g agree.

27) **Range**

For any domains A B and vector f:A->B, the expression

```
{f}
```

denotes a boolean vector with signature B->2 such that for each element of B

```
y:1->B
```

which occurs in the range of f, i.e. for which there exists a

```
x:1->A
```

with f x = y then

```
{f} y = |1|
```

28) **Equality**

For any domain A there is an operator called equality with signature

```
eql:(A,A)->2
```

For any domains A B and vectors f:A->B g:A->B, equalizer, equality and range are related by

```
eql(f,g) = {f=g}
```

29) **Boolean vectors**

The enumeration domain 2 has two elements. These elements also serve as truth values.

Element	Signature	Description
|0|	1->2	false
|1|	1->2	true

Vectors of the form x:A->2 where A stands for any domain expression, are called boolean vectors. For any boolean vector x:A->2, the expression

```
true x
```

denotes a vector with signature n->A where n is an enumeration domain. True is related to range by

```
{true x} = x
```

True can be defined in terms of equalizer as

```
x=|1|
```

30) **Co-equalizer**

For any vectors f:A->B and g:A->B, the expression

```
coe(f,g)
```

denotes a vector with signature B->n. coe(f,g) represents the finest partition of B which makes f and g agree.

31) **Title sets**

Name	Expression	Signature	Definition
subject titles	$f	{f}->A	subject titleset constructor
object titles	@f	A->{f}	object titleset constructor

These vectors are used together with composition to associate title sets with untitled vectors.

32) **Groupings**

Name	Expression	Signature	Definition
subject grouping	f$	[f]->A	subject group constructor
object grouping	f@	A->[f]	object group constructor

These vectors are used together with composition to assocate groupings with untitled vectors.

33) **Real Arithmetic**

For domain A and vectors m:A->REAL and n:A->REAL, we have the following element-wise arithmetic expressions

Name	Infix	Expression	Signature
addition	m+n	add(m,n)	A->REAL
multiplication	m*n	mul(m,n)	A->REAL
subtraction	m-n	sub(m,n)	A->REAL
division	m/n	div(m,n)	A->REAL

33.1) **Real Vector Space Arithmetic**

The above element-wise expressions are extended to vectors whose objects are real vector spaces. A real vector space is a vector space of the form (A->REAL) for any domain A. For any domains A B C with B and C different and vectors m:A->REAL n:A->REAL p:A->(B->REAL) q:A->(B->REAL) r:A->(C->REAL) we have the following cases:

Expression	Signature	Description
m+p	A->(B->REAL)	Scalar by vector
p+q	A->(B->REAL)	Element-wise vector by vector
q+r	A->((B,C)->REAL)	Outter vector by vector

34) **Summation, Aggregation and Linear Interpolation**

For domains A B and vectors x:A->(B->REAL) f:A->B m:A->REAL n:A->REAL ts:{m}->REAL

Name	Expression	Signature	Definition
summation	sum x	A->REAL	sum over domain B
aggregation	agg(f,m)	B->REAL	sum m(i) for f(i)=j, i in A, j in B
linear interp.	line(ts;n)	A->REAL	step-wise linear function line(ts;_):REAL->REAL

35) **New Operators**

Evaluate is easily extended since any name which is not defined as a variable is treated as an operator. Thus if XXX is not defined, then the use of XXX in an expression. e.g.

```
XXX ( ... )
```

is translated as an operator. XXX is expected to be the name of a low-level filter.

36) **Operator Expressions**

When operators appear at the lowest level in an expression, rather than vectors, the result in called an operator expression. For example, the expression

```
(sbj,obj)
```

denotes the operation evaluating the subject and object of a vector then pairing the result. A special operator _ (underscore) called the null operator is available to form operator expressions such as

```
(_+_)
```

This operator can be applied to a vector whose object is REAL. The result is a vector whose elements are twice the original value.

37) Using Evaluate

To enter Eunice from VMS type:

```
$ csh
```

Programs are in directory /u/page/bin. This should be added to your PATH.

Three sub-directories are expected in the working directory in which Evaluate is run:

inp	contains input vector values
out	contains output vector values
tmp	contains temporary vector values
scn	contains views

Evaluate is initiated by the command

```
eva [-l] [viewname]
```

-l option causes low-level shell code to be output to stdout instead of being executed. This allows ev to be used as a "compiler" for the low-level language. If viewname is included, variable definitions and values are loaded from the specified file in sub-directory scn.

38) Evaluate commands:

exit	Exit Evaluate
clear [varname]	Deletes all variables or a specified variable definition and value
load viewname	Retrieve variable definitions and values from a file
save viewname	Save variable definitions and values on a file
reset [varname]	Delete all or a specified temporary variable value(s)
list [varname]	List all variable definitions and values or a cross-reference of a specific variable.
varname:filename	Assign a value to a variable
varname=expression	Define a variable
edit varname	Full screen edit of value of variable
expression	Print the value of the expression

expression stands for any vector or operator expression defined above.

39) Variable names

Variable names must begin with a letter. Upper and lower case letters are treated as distinct. The list of variables with their definitions and values is maintained in alphabetical order by variable name.

40) **Values**

Values of variables are stored as files. The save command, in addition to saving the definitions of all variables as a view file, causes the temporary values of variables to be saved as permenant files in the either the inp or out sub-directories. inp is used if the variable has no definition. out is used if it has both a definition and a value. The permanent file name is formed by appending a sequential numeric extension to the variable name.

41) **File names**

Note that paths part of file names in Evaluate use the \ delimiter, not / as in Eunice.

42) **Interface with TERF**

To retrieve a vector from the TERF inventory:

While outside of evaluate (in the shell) issue the command:

```
getinv matrixname
```

A file called matrixname and any associated titlesets will be written to the inp sub-directory in your working directory.

When in Evaluate assign matrixname as the value of a variable, e.g.

```
matrix:matrixname
```

Now matrix may be used in expressions.