MoNet

Complex Experiment Simulation Platform

Content

.

0. Overview

Monet is a computerized platform that helps describe complex systems. Monet is developed using conventional programming techniques. Nevertheless, these programming techniques are used novelty and rely on custom-made script languages that endow Monet with remarkable power and flexibility to model multidimensional complex systems successfully. The idea of modeling systems by describing interconnected elements' structure and time-changing phenotypes arose in 1996 with the Monitor system that preceded Monet. However, Monet's formal development began in 2011 as a tool needed to perform several experiments included in the doctoral work by Gerardo Febres at the Universidad Simón Bolívar.

In Monet, calculations and simulations are based on routines or functions that perform any required task. A new role is added if the existing roles do not cover a task. Thus, the Monet platform increases its capabilities. A system description in Monet is in parts organized like a tree. At the file level, each tree node is registered and described as an independent file on the computer. The leaves of the tree are the most detailed representation of the system, and descriptions of higher-level components are "added" on that level, which in turn can interact with each other.

The visual representation of a system's component is a tree-like structure shown in a grid displaying the properties of each of the child nodes of the component described. In this way, MoNet represents each system component in the grid rows. One row for each child of the described component. The component displayed on the grid that corresponds to a node "knows" who and where its parent is. Navigation is possible through the different levels of detail of the model.

The impact of these system representation criteria goes beyond the merely descriptive. With the ability to add routines responsible for performing functions, the platform is capable of executing calculation operations between complex structures while keeping the results organized. They are like tensor operations assignable to cells, and therefore, the order of huge system descriptions is not lost.

1. Comprehensive System Descriptions

A Monet's system description is a collection of interacting elements. Monet's descriptions involve two aspects regarding each element comprising the system: the phenotypic description and the structural description.

(i) Phenotypic description: depiction of each element by listing its attributes with their corresponding values. The attributes' values may be represented by texts or by numerical expressions.

(ii) Structural description: This aspect of the description includes the set of contained elements that constitute an element. The structural description is a related-element network forming a tree-like structure deepening into an increasing detail description level.

An attribute value is inherited from another element's attribute value.

MoNet simulates systems described in sets of inter-connected files. Each file comprises the description of an object that may be a compound entity. However, the file of a compound object does not contain the description of the contained objects. Instead, a container object has the necessary information to point to the files where the contained objects' files are. Thus, an object's description may consist of the values of the attributes that characterize the type of object — which would be the description at its scale —or maybe the collection of descriptions of the objects forming it -this would be the description at a more detailed scale. Consistently, the descriptions of the more detailed components can contain even more detailed elements, allowing for increasingly detailed descriptions until indivisible elementary objects are reached.

This abstract representation of a system leads to a hierarchical web of files capable of representing complex systems with descriptions at different scales

Monet's file structure

MoNet simulates systems described in sets of inter-connected files. Each file comprises the description of an object that may be a compound entity. However, the file of a compound object does not contain the description of the contained objects. Instead, a container object has the necessary information to point to the files where the contained objects' files are. Thus, an object's description may consist of the values of the attributes that characterize the type of object —which would be the description at its own scale— or maybe the collection of descriptions of the objects forming it -this would be the description at a more detailed scale. Consistently, the descriptions of the more detailed components can contain even more detailed elements, allowing for increasingly detaigled descriptions until indivisible elementary objects are reached.

This abstract representation of a system leads to a hierarchical web of files capable of representing complex systems with descriptions at different scales.

Graphical user interphase. The environment

MoNet uses three panels to represent model descriptions. The main panel contains a grid that contains detailed descriptions of the components of the system being observed at a certain scale.

Tailored Data and Command Representations

MoNet uses a tailored script languages to represents complex structures and to allow defining mathematical operations among them, and automatic commands:

The Data Autonomous representation (DAR)

The Command Script language (CSL)

2. Multi-dimensional environment

The requirements for locating, selecting and handling information elements in a multidimensional logical environment stress the need for a specialized syntax script language. MoNet employs systems of rules and structures which make the system of script languages devoted to achieving this objective.

2.1 The Data Autonomous Representation. DAR Structures

Any multidimensional structure of values can be represented as a combination of three prime types of structures: ORTHOs, TREEs, and RINGs. Complex interconnected structures are foreseeably within the scope of MoNet's representing capabilities. However, the prime types of structures are first explained in this document.

As a rule that applies to all structures, MoNet uses special symbols to split elementary components that form a compound multidimensional structure. The splitting symbols are of the form ']d['. The opening and closing brackets (in that order) suggest the sides where the elements are being separated. The letter 'd' is the number of the dimension the splitting symbol ']d['refers to. The splitting dimension tag 'd' starts with the number zero (0).

ORTHOs: To this type belongs any structure being formed by the same number of elements counted within any of the structure dimensions. Thus, ORTHOs are a regular set of elements showing structural symmetry around any plane oriented perpendicular to the direction of each dimension. To describe

Figure S.1 shows examples of ORTHO structures in one, two, and three dimensions (a figurative version). The corresponding color description of these structures is as follows:

One-dimensional structure, in Figure 2.1.a:

K]0[B]0[V]0[Y

Two-dimensional structure, in Figure 2.1.b:

K]0[B]0[V]0[Y]1[B]0[C]0[O]0[G]1[O]0[B]0[C]0[K

Three-dimensional structure, in Figure 2.1.c:

K]0[B]0[V]0[Y]1[B]0[C]0[O]0[G]1[O]0[B]0[C]0[K]2[V]0[E]0[Y]0[W]1[K]0[D]0[R]0[Y]1[D]0[E]0[D]0[D

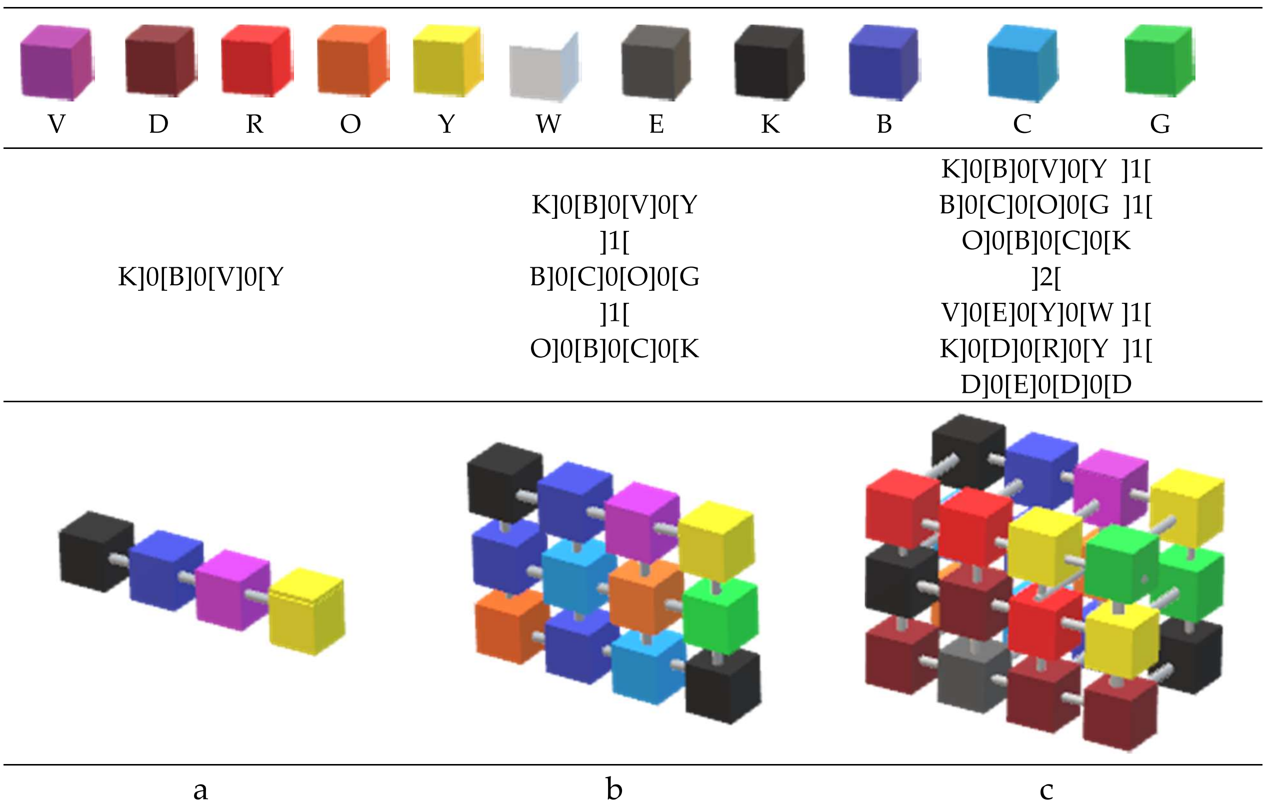


Figure 2.1. The color-property description of ORTH-structures of objects of one, two and three dimensions. The particular case of one-dimension-ORTH can also be considered a LIST.

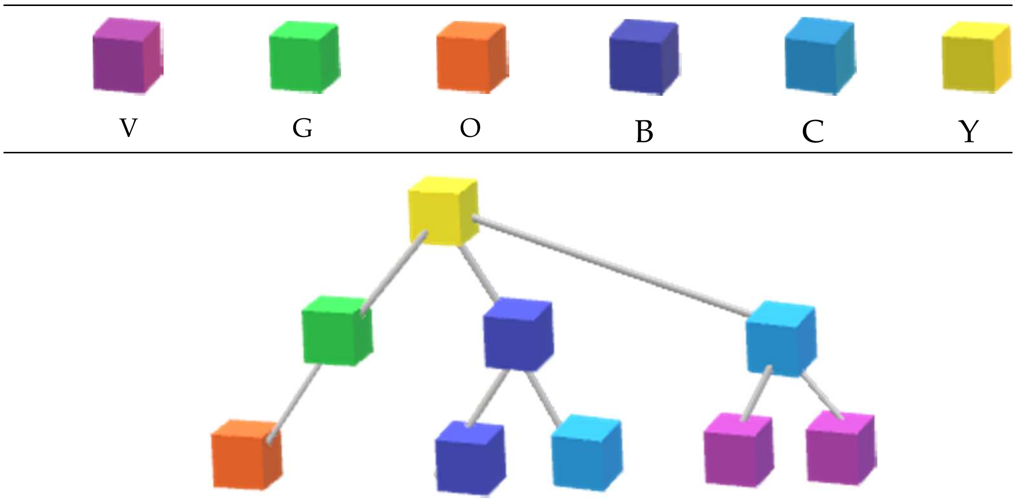
TREEs: Trees are fractal-like structures. This means they do not necessarily live in an integer number of dimensions. Three-structures appear when an property description is divided into more detailed parameters or components, each of which can be described in more and more detail by successive divisions.

Therefore, trees do not completely fill any ortho-space where we may pretend to insert the tree. Trees do not fit into orthogonal shapes as the tables or matrixes are. Thus, in conventional data records, representing tree-shaped data usually leads to important, and undesirable, amount of redundancy. When using the Autonomous Representation trees are depicted by nesting the divisions of each branch of the tree into the paired symbols "{"and "}".

A tree of property-values depicted in an increasing detail-level is presented in Figure 2.2. Tree structure, in Figure 2.2 shows an example of a single three level tree with the following description:

Y]1[E]2[O]1[DB]2[B]0[B]1[V]2[V

The root is the object's yellow property which is represented with the tag 'Y' located at the start of the expression. The depth associated with the root is considered to be zero. Going deeper from the root, the depth of any tree's component is determined by reading from left to right the number of open-curly-brackets ({) minus the number of close- curly-brackets (}).



Y]1[E]2[O]1[DB]2[B]0[B]1[V]2[V

Figure 2.2. An example of description of a TREE-structure.

Tree structure, in Figure 2.3 shows LIST of TREE structures. Three structures, each one being a TREE, are connected by an ORTHogonal splitter forming a complex structure with the following coded description:

G{O}]0[B{B]2[C}]0[C{V]2[V}

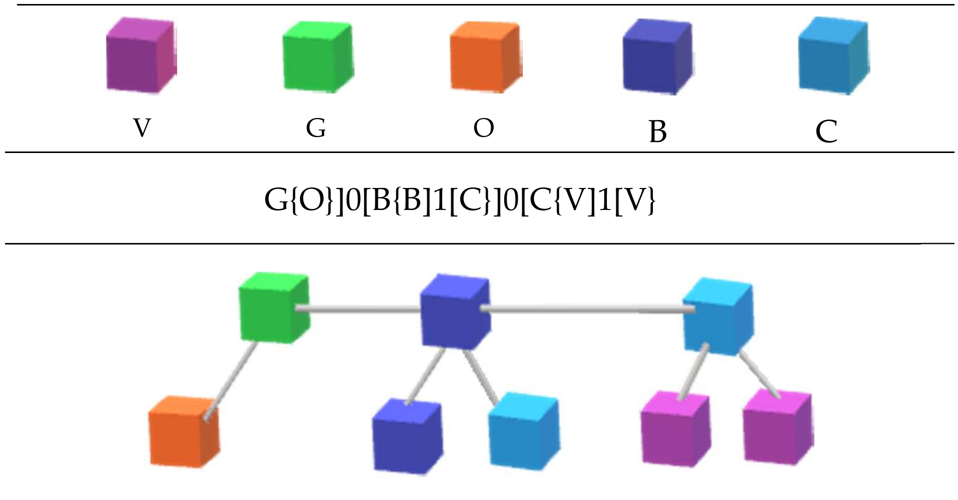


Figure 2.3. An example of a description of a LIST of three TREE structures.

Representing TREE structures is one of uttermost capabilities of DAR since it provides ways of modeling commonly found structures in nature.

RINGs: Rings are cyclic structures. MoNet support for RING-structures is currently being developed.

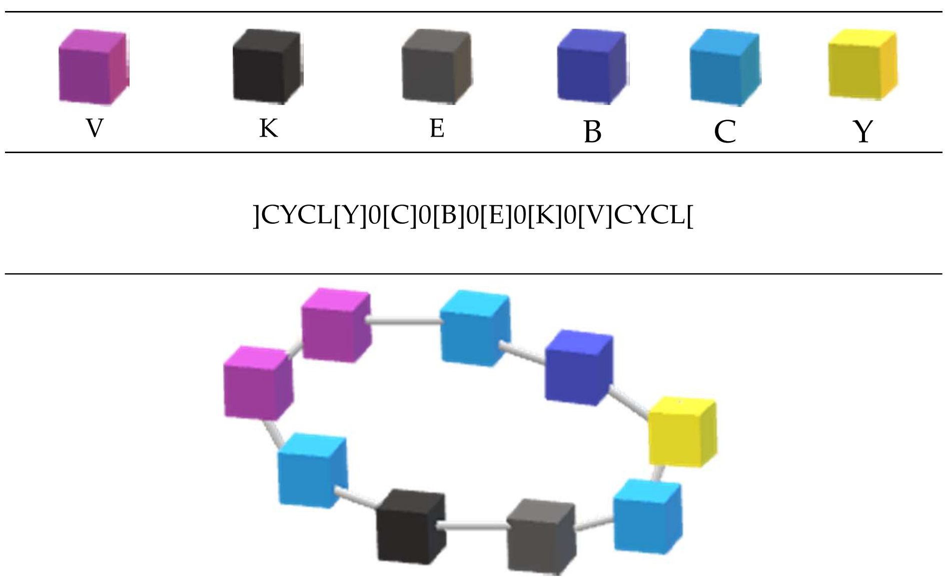


Figure 2.3. An example of description of RING-structures.

2.2 Attributes and property names

Description attributes are identified by a text with a capital-letter written extension that specifies the kind of value the attribute normally takes. Thus, for example, ‘color.STRN’ means the attribute ‘color.STRN’ takes string values. The following are the currently supported types of attribute values:

`.STRN`	String
`.BOOL`	Boolean
`.INTG`	Integer
`.FLOT`	Floating point
`.LIST`	Compound List of values of any Elementary type
`.STRC`	Compound structure of values of any Elementary kind
`.LINK`	Link to an executable file
`.EXEC`	Action to be performed by MoNet
`.TASK`	
`.AUTO`	

In general the name of an attribute is:

```
TheAttribNameWithType = TheAttributeName.TYPE
```

2.3 Attributes and property values

To refer to the value of an attribute, the identifying text is surrounded with paired symbol ‘<’ and >’. Thus, the value of attribute ‘color.STRN’ is retrieved with the expression ‘<color.STRN>’. Thus, taking the structure in Figure S.1.a as an example, its name could be ‘Figure.S.1.a.LIST’ and its value, as it is represented in the figure, is DR]0[R]0[O]0[Y . Then we can write <Figure.S.1.a.LIST> = DR]0[R]0[O]0[Y. In general the syntax is:

```
TheAttribValueWithType = <TheAttributeName.TYPE>
```

2.4 Localizer

Localizer is the term to refer to the sublanguage used to locate and retrieve attribute values and subsystem descriptions within the MoNet’s environment.

A value exiting within the model net is signaled by setting the value of three coordinates:

a. COORD.Agent.AttribName: the agent’s attribute which value is the one being searched.

b. COORD.Agent.Name: the agent’s ID or Tag name , and
c. COORD.PATH: the agent’s file path,

The coordinate COORD.Agent.AttribName signaling the attribute whose value is of interest must always be specified. Such a coordinate statement follows the syntax:

`<SomeAttribute.TYPE>` .

The coordinate COORD.Agent.Name needs to be specified when the referenced value belongs to an agent (or element) that is different from the one holding the localizer expression. This specification is done by using a conditional statement that locates the element the attribute-value is referred to:

`<@><ConditionAttribute.TYPE> = ConditionValue</@>` .

The coordinate COORD.PATH needs to be specified when the referenced value is in a file that is different from the one holding the localizer expression. This specification is done wih the following syntax that indicates the path where the sought element is:

`<~>Literaly written agent’s File Path</~>` ,or
`<~><PathAttrib.LINK></~>` .

In general, a localizer statement can look as any of the following sentences:

`<TheAttributeName.TYPE>` , or
`<TheAttributeName.TYPE><@><ConditionAttrib.TYPE> = CondValue</@>` , or
`<~><PathAttrib.LINK></~><TheAttributeName.TYPE><@><ConditionAttrib.TYPE> = CondValue</@>` .

2.5 Extracting substructures (Using Extractors)

Extractor is the name of the syntax used to retrieve the value of subsets of an attribute-structure. A single Extractor-phrase retrieves a connected portion of the subject structure. The Extractor-phrase must be surrounded by the char ‘!‘, and located after the closing angled-bracket ‘>’ of the attribute’s expression, or before the attribute’s tag closing angled-bracket ‘>’.

A general substructure extraction from the structure `AttributeName.STRC` is specified as `<AttributeName.STRC!ExtractorPhrase!>` . The shape of the `!ExtractorPhrase!` varies upon the type of structure it is applied to.

Extracting sub-ORTHs: The extraction of substructures from ORTHs is specified by indicating the smallest coordinate value in all dimensions and the largest coordinate value in all dimensions. Thus, if the subject structure is orthogonal-three-dimensional, the substructure is set by signaling the vertexes located at the closest-left-upper corner and the farthest-rightlowest corner of the subject structure. Each coordinate value is separated from its neighbor by a two-dot sign (:).

Specifying an element within an OTRH structure is achieved by indicating the values of each coordinate where the element is located. The coordinate values are spitted by the two-dot symbol (:). For example, retrieving the color property value of the (black) element at the close-lower-left corner of Figure S.1.c is done with the following syntax:

```
<Figure.S.1.c.TYPE!0:0:0!> = N.
```

The same property for the element located in the top of the second column of the closer plane of elements would be:

```
<Figure.S.1.c.TYPE!1:2:0!> = G.
```

Extracting sub-ORTH structures is done using the Range-Limit-Splitting symbol ‘`]…[`’. The Range-Limit-Splitting symbol indicates that all elements located within the range limits indicated at the start and the end of the splitting symbol are included in the selection. Thus, the extractor-phrase `{L]…[U}` means that all elements located above (or equal) the lower limit L, and below (or equal) the upper limit U are included in the structure extraction. The coordinate's limits L and U refer to the corresponding dimension of the subject structure. The limits of multidimensional ORTH structures are also specified using the two-dot dimension-splitting symbol (:). The general extracting phrase `{L0:L1:L2]…[U0:U1:U2}` means that elements within the limits specified for dimensions zero, one and two respectively, are to be selected. An example helps to understand the syntax. Extracting the eight elements of the close-plane lower two rows of Figure S.1.c implies the following syntax:

```
<Figure.2.1.c.TYPE!0:0:0]…[3:1:0!> = K]0[B]0[V]0[Y]1[B]0[C]0[O]0[G
```

The extracted structure is shown in Figure A.2.

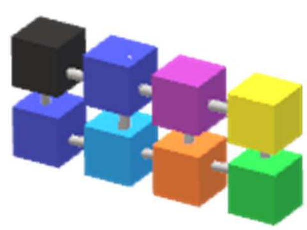


Figure 2.4. A substructure extracted from the structure shown in Figure 2.1.c

This syntax for specifying sub-ORTHs of any ORTH is applicable to ORTHs of any number of dimensions. The number of splitting dimension symbols ‘:’ in the extracting phrase indicates the number of dimensions (minus 1) of the subject structure, and therefore, these two numbers must match.

Extracting sub-TREEs: To the type **TREE** belong structures formed by components connected in a tree-like topology; each element may contain several hierarchy lower components, which in turn, may be divided into 'lower' components. The extraction of substructures from TREEs is specified by indicating the elements that will represent the roots of the new selected trees.

Specifying an element within a TREE structure is achieved by indicating each coordinate where the element is located. In the case of trees these coordinates are specified by using nested curly brackets. For example, retrieving the color property value of the second sub-tree Figure 2.3 is done with the following syntax:

```
<Figure.2.3.TYPE!1!> = B{B]0[C}
```

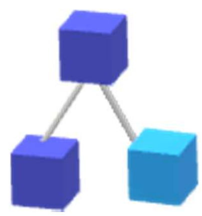


Figure 2.5. A substructure extracted from the TREE structure shown in Figure 2.3.

A list of sub-trees can be selected using the LIST Range operator (]…[). Thus, selecting the two ending trees from the list of tree shown in Figure 2.3 would be as follows:

```
<Figure.2.3.TYPE!1]…[2!> = B{B]0[C}]0[C{V]0[V}
```

Extracting sub-RINGs: This section is reserved for sub-RING's extraction. However, at this point it is foreseeable a RING has not sub-RINGs. Thus, this command may never be necessary.

2.6 Special tags for specifying dimension data-range

There are several special tags that are useful for specifying generic limits in the coordinate range of any dimension of the elements to be extracted from a structure.

<Last>	Refers to the last coordinate value in the dimension is at.
]…[	Retrieves all data defined by phrases before and after the symbol.

].R0.p:R1.q:R2.r.[	Retrieves the substructure with resolutions p, q and r for dimensions 0, 1 and 2 respectively.
`].:R1.q:R2.r.[`	Retrieves the substructure with full resolution for dimension 0 and resolution q and r for dimensions 1 and 2 respectively.
`].::R2.r.[`	Retrieves the substructure with full resolution for dimensions 0 and 1, and resolution r for dimension 2.
`].::.[`	Retrieves the substructure with full resolutions for dimensions 0, 1 and 2 respectively. Equivalent to using]…[.

2.7 Assembling structures (Using Integrators)

Integrator is the name of the syntax used to link two or more structures to form a joint structure. An Integrator-operator details how two structures are to be combined to form a resulting structure. Assembling structures is a recently devised functionality. The need for forming joint structures was detected during March of 2023, when G. Febres, while studying ways to detect patterns [reference], had some language descriptions represented by TREE-structures, needed to be chunk, selected, and then, the resulting sub-TREEs joint into a new language descriptive structure.

An Integrator operand has the following form:

`{*}` .

2.8 Graphs

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order

2.9 Control window

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order

3. System's structure modeling

DAR handles three primitive shapes: ORTH, TREE and CYCL. These four-letter shape names stand for orthogonal, tree and cycle, respectively. Any structure can be described as a connected set of sub-structures of any of these primitive types of data arrays.

3.1 The main grid

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

3.2 The graphic's panel and the graphic control panel

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

3.3 Commands menu

Figure 4.1 illustrates Monet's main command menu bar. Beside buttons devoted for direct operations, most commands are included in these menus.

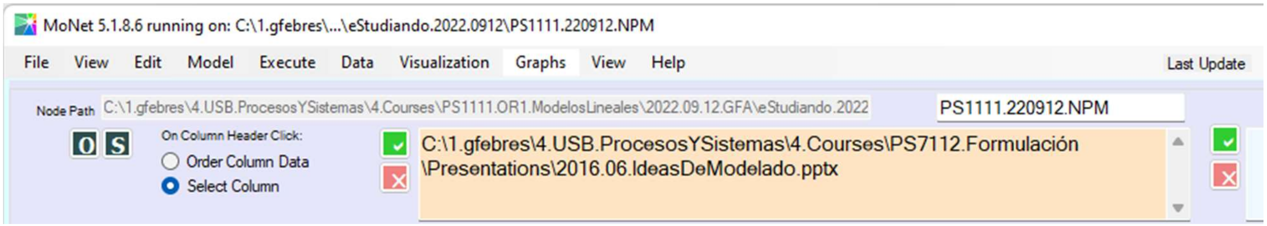


Figure 3.1. Other Monet's menu bar grouping most frequently used coStructure viewmmands.

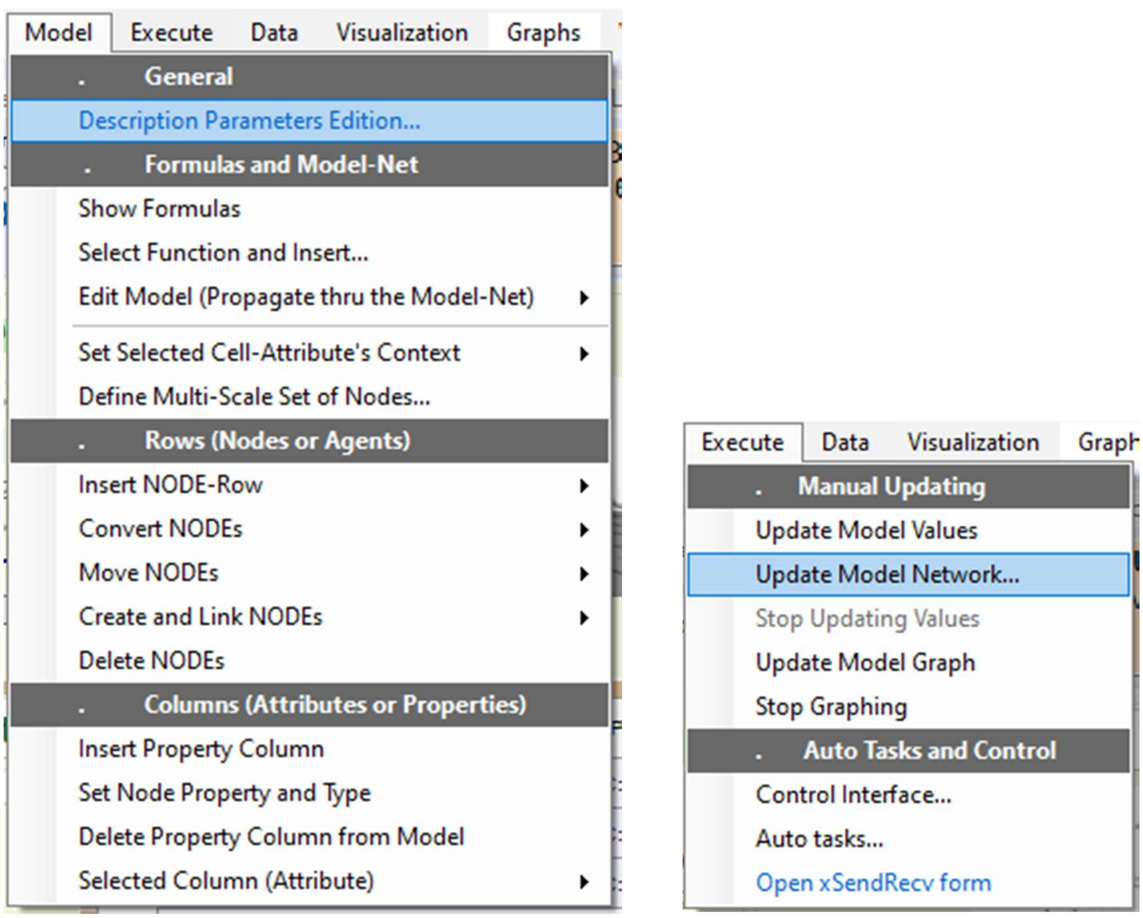


Figure 3.2. Monet's Model Structure.

4. Syntax and functions

The shape of structures are described in a synthetic fashion. This capability is important to describe the shape of compound structures, yet keeping a short description length.

DAR handles three primitive shapes: ORTH, TREE and CYCL. These four-letter shape names stand for orthogonal, tree and cycle, respectively. Any structure can be described as a connected set of sub-structures of any of these primitive types of data arrays.

4.1 Arithmetic operations

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

Symbol	Operator action
^	Exponentiation
*	product
/	division
%	Integer division
+	Sum
−	Subtraction

Operator's symbols must be separated from the rest of the expression by including spaces before and after the symbol. For example: The expression A * B is interpreted as the product of A times B. The expression A*B is interpreted as the text 'A*B'.

Parenthesis '(' and ')' are used to group operations.

4.2 Transcendental functions

Arithmetic operations are represented with the same operators and syntaxes

4.3 Operating with structures

The shape of structures are described in a synthetic fashion. This capability is important to describe the shape of compound structures, yet keeping a short description length.

DAR handles three primitive shapes: ORTH, TREE and CYCL. These four-letter shape names stand for orthogonal, tree and cycle, respectively. Any structure can be described as a connected set of sub-structures of any of these primitive types of data arrays.

A number located before the name of the structure type indicates de number of dimensions of the structure

Integer numbers located after the name of the structure type indicate de size of each dimension of the space where the structure lives. These integer number are separated by the two-dot character (:). Therefore, for ORTHs the number of two-dots reveals the number of dimensions of the orthogonal structure. For instance, the shape of a cube with each edge of size seven is described as ORTH{7:7:7}.

A list of four cubes with diminishing sizes seven, six, five and three respectively:

ORTH{7:7:7}]0[ORTH{6:6:6}]0[ORTH{5:5:5}]0[ORTH{3:3:3}

7LIST{7LIST{7LIST{Y}}}

LIST{LIST]1[LIST]0[LIST]0[LIST }

LIST{LIST]0[LIST{LIST}]0[LIST{LIST]0[LIST}}

0-ORTH = SCLR

1-ORTH = LIST

2-ORTH es una matriz

3-ORTH(i:j:k) es un paralelepipedo de tamaño i j k

4.4 Special functions

Monet comprises specially configured functions to treat specific operations. Following there are several examples of special functions with their parameters.

Entropy: Computes the symbolic entropy of a set of symbols listed in an autonomous list of symbol Tuples separated by `"]0["`.

ArgOrder	Type | Ref. Name	Description
Arg0	string Arg0Expression	Returns the entropy associated with the distribution of numbers included in the Argument STRC

Example 1: Entropy(<FiltredHist.LIST)

LanguageEntropy: Returns the entropy [0,1] of a Language described with a LanguageStruct as: Symb1]1[Freq1]2[Pos11]2[Pos12]2[...Pos1N]0[...]0[SymbLast]1[FreqLast]2[PosLast1]2[PosLast2]2[...]2[PosLastM

ArgOrder	Type | Ref. Name	Description
Arg0	string Arg0Expression	Returns the entropy [0,1] of a Language described with a STRC

Example 1: LanguageEntropy(<FiltredHist.LIST>)

FundamentalScale: Retrieves ….

ArgOrder	Type | Ref. Name	Description
Arg0	string `MultidimValueSTRC`	Converts the numerical (multidimensional) description MultidimValueSTRC into a description formed with elementary symbols of the same dimensionality
Arg1	string `ScaleTypeLIST`	Type of scale specifying the non-linearity of the scale
Arg3	String `ScaleMaxValLIST`	LIST with the Max Value for each scale's dimension
Arg4	string `ScaleResLIST`	LIST with the Resolution for each scale's dimension
Arg6	string `ScaleParamsSTRC`	STRC with Scale specific parameters for each scale's dimension
Arg7	string `FirstASCIICode`	(Optional) ASCII number of the `first character representing the elementary symbols`

Example 1: ConfigureSymbolicScale(<FiltredHist.LIST>, Hyperbolic, MinElem(<FiltredHist.LIST>), MaxElem(<FiltredHist.LIST>), <Resolution.FLOT>, <Inflection.INTG>, <ScaleParam.FLOT>)

4.5 Control functions. Meta-functions

Arithmetic operations are represented with the same operators and syntaxes

STRCTgrow: Builds a one-dimensional structure with elements whose values are computed as indicated in the function's arguments.

ArgOrder	Type | Ref. Name	Description
Arg0	string TheSTRC	The Growing STRCT Attribute name. The component whose value is used to feed the Growing structure
Arg1	string GrowthDimSplitter	The Dimension where the STRCT will Grow. The splitter symbol to split the growing structure elements
Arg2	Int GrowthNumSteps	The number of the grow-steps
Arg3	`String ResetSTRNG`	Statement to signal whether or not the computation should erase the previous computations. <Reset> = True or <Reset> = False.
Arg4	string GrowthCompXpressn	The Expression that explains how to compute the new structure values
Arg5	string GeneralityOfXpressn	refers to the type of Expression evaluated

Example 1: STRCTgrow(<ProcessValue.STRC.Last><IC><InitialCond.STRC></>,]0[, 1, 1, <ProcessParams.STRC> * <ProcessValue.STRC><IC><InitialCond.STRC></>{<Last>} * (1 - <ProcessValue.STRC><IC><InitialCond.STRC></>{<Last>}), Compact)

5. User interphase

DAR handles three primitive shapes: ORTH, TREE and CYCL. These four-letter shape names stand for orthogonal, tree and cycle, respectively. Any structure can be described as a connected set of sub-structures of any of these primitive types of data arrays.

5.1 The main grid

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

5.2 The graphic's panel and the graphic control panel

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

5.3 Commands menu

Figure 4.1 illustrates Monet's main command menu bar. Beside buttons devoted for direct operations, most commands are included in these menus.

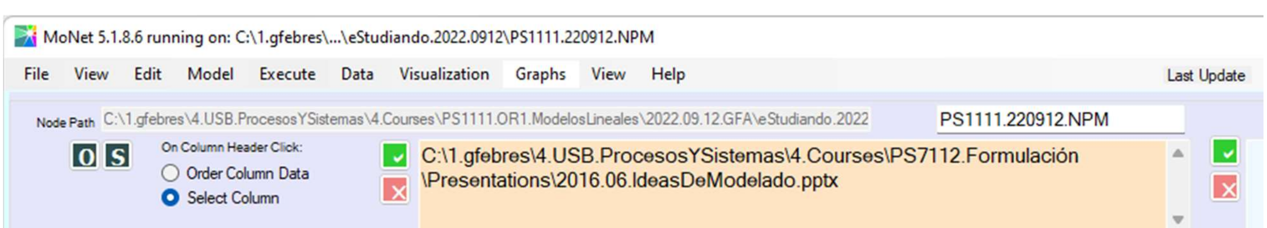


Figure 4.1. Monet's menu bar grouping most frequently used commands.

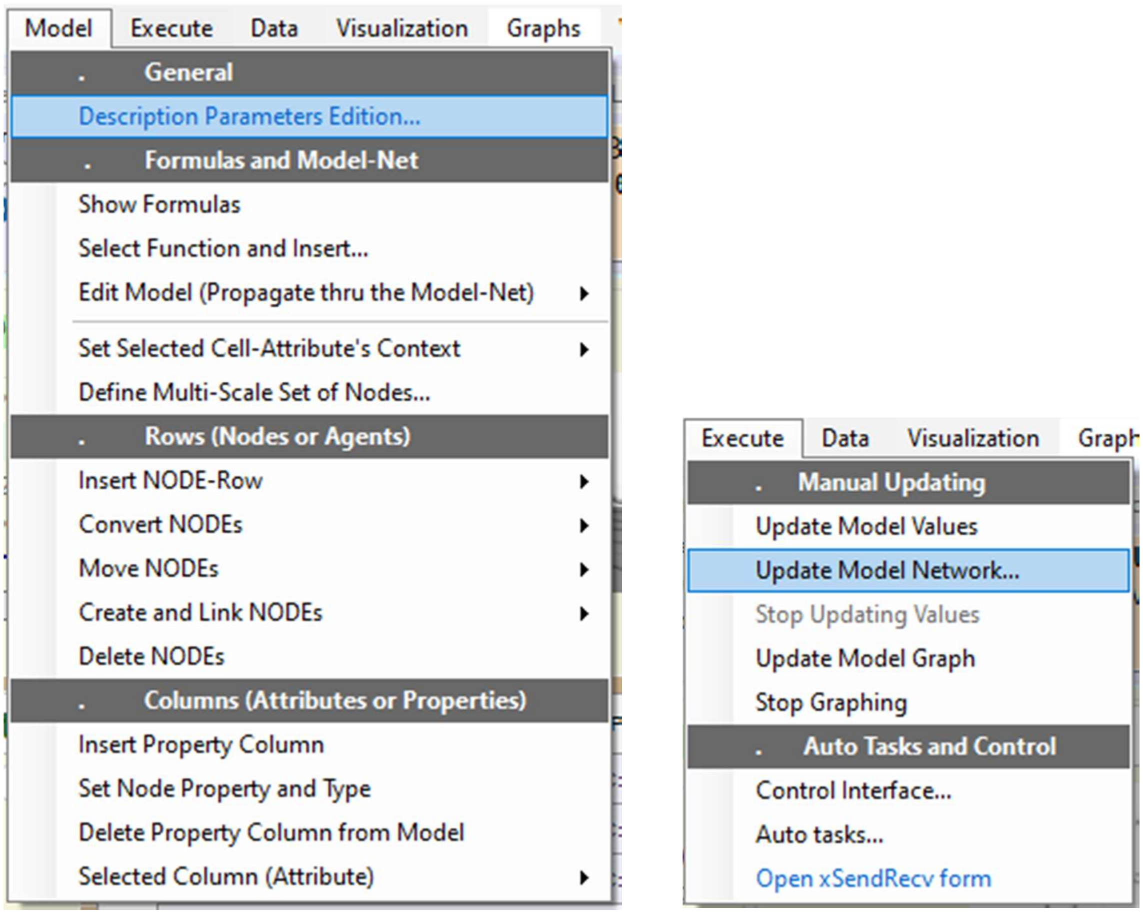


Figure 5.2. Monet's Model and Execute menus.

6. Graphics

6.1 Visualization Technique

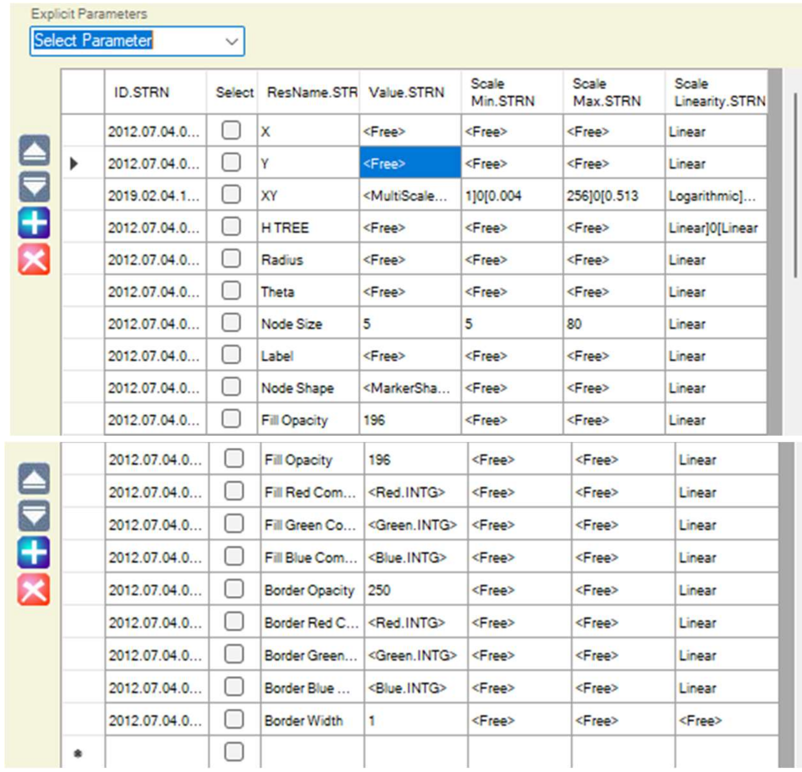
The rationale behind MoNet ‘s graphics module is that a graphic is a projection of the phenomenon studied on a 2D surface representing the selected attribute values as the dimensions and other properties of the graph elements to achieve an effective visual description of an aspect of the system.

Naturally, the position over the graph’s x and y axes represents the values of two attributes linked with the axes. But market size, shape, border width, label, and other marker properties can be associated with values of the system depicted. Therefore, it shows more than just two dimensions in each graph.

6.2 Graphic resources

Figure 6.1 shows a grid offering the graphic resources that can be associated with the model attribute values. This grid's panel is usually hidden to save the user interface space —the View menu offers a submenus for opening and closing the graphic resources panel.

A set of attributes linked to a 2D graph defines a 2D graph capable of representing up to 6 dimensions of the subjects selected as the graph subject. After naming this graphic subject with its set of attributes, there is a parametric description of a graphic projection that may be applied to any similar subject living within the simulation scene.



Explicit Parameters

Select Parameter

	ID.STRN	Select	ResName.STR	Value.STRN	Scale Min.STRN	Scale Max.STRN	Scale Linearity.STRN
	2012.07.04.0...	☐	X	<Free>	<Free>	<Free>	Linear
▶	2012.07.04.0...	☐	Y	<Free>	<Free>	<Free>	Linear
	2019.02.04.1...	☐	XY	<MultiScale...	1]0[0.004	256]0[0.513	Logarithmic]...
	2012.07.04.0...	☐	H TREE	<Free>	<Free>	<Free>	Linear]0[Linear
	2012.07.04.0...	☐	Radius	<Free>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Theta	<Free>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Node Size	5	5	80	Linear
	2012.07.04.0...	☐	Label	<Free>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Node Shape	<MarkerSha...	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Fill Opacity	196	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Fill Opacity	196	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Fill Red Com...	<Red.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Fill Green Co...	<Green.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Fill Blue Com...	<Blue.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Border Opacity	250	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Border Red C...	<Red.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Border Green...	<Green.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Border Blue ...	<Blue.INTG>	<Free>	<Free>	Linear
	2012.07.04.0...	☐	Border Width	1	<Free>	<Free>	<Free>
*		☐					

Figure 6.1. The grid connects model attributes with 2D graphic properties.

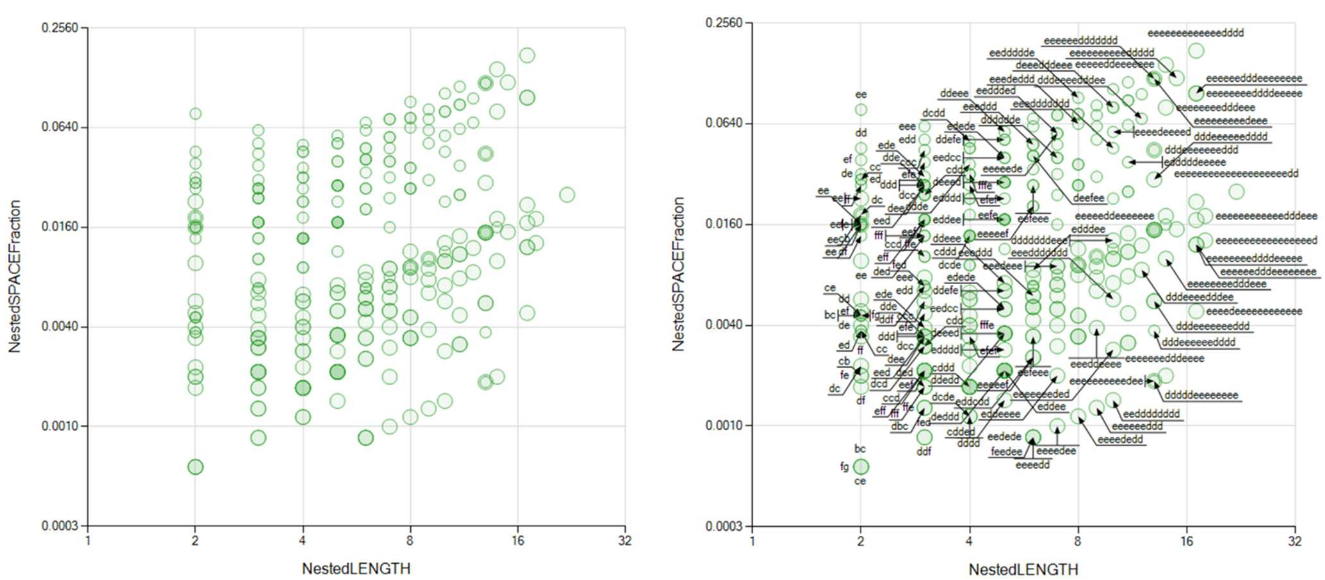


Figure 6.2. Illustration of a 2D graph showing three or more object dimensions.

7. Applications and examples of specific models

Monet is a continuously developed platform. Initially, in year 2012, Monet served as the basis to conceive and develop algorithms to perform complex classifying tasks. As more studies set additional requirements, Monet has evolved to comply with the new challenges that studying complex systems frequently presents. Therefore, Monet is now an agile modeling platform capable of incorporating procedures and functions into its capacity for modeling and visualizing systems. Following, there is a list of studies where Monet's use was crucial. Most of these studies resulted in publications that are referenced below.

Entropy-based classifying models

Information entropy is a property of probability distributions. Since the entropy of a probability distribution can be quantified, evaluating entropy is a powerful to method characterize complex systems by recognizing the symbols used in the description of the system, and then counting the frequency of their appearance within the description.

In 2014, Febres, Jaffe, and Gershenson presented a comparison between Spanish and English [1]. The study relied on Monet's platform and needed to extract the symbols, which in this case were words, from more than 400 famous speeches by notorious authors. Two tasks proved difficult to comply with to fulfill the objectives of the study: 1) the capacity to split and record all symbols (words) from each speech, and 2) the capacity to register and control the set of symbols of more than 400 speeches and to share the properties of the distributions of these set of symbols among all speeches, for quantitative comparison purposes. Separating a natural language text into words is a straightforward task; any word is preceded and followed by either a space or a punctuation sign. However, recording the characteristic set of words with their frequencies for each speech, is not an obvious procedure, especially when there are hundreds of different words in each speech and there are several hundred speech to keep track of.

To reach our objective, we created the Data Autonomous Representation (DAR), and incorporated it into Monet. Originally, Monet was conceived to represent and to study network properties and performance. Under this conception Monet would represent a speech as a network with as many nodes as different words in the speech, and as many arcs connecting nodes as the number of times a word (node) precedes or follows another word (node). Using conventional computing variables and structures to represent the topology of these more than 400 networks seemed unmanageable. Thus, after creating DAR, Monet was capable of embedding the network of a whole speech into a single grid's cell or a single field of a database, if the system were based on a database.

Besides the DAR, the function `SplitStruct()`, created to split a large text in a set of words with their corresponding frequencies, was essential to achieve the goal. The set of symbols (words) obtained for each speech is then fed into Monet's function `Entropy()` to obtain a measure of the speech's information complexity for Spanish and English.

Space decomposition linear optimization models

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

Educational games

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

Integrating differential equations

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

Empirical probability models

Arithmetic operations are represented with the same operators and syntaxes conventionally used. Thus, valid operators, presented in their precedence order, are:

8. Function dictionary

Special functions

Monet comprises specially configured functions to treat specific operations. Following there are several examples of special functions with their parameters.

Entropy: Computes the symbolic entropy of a set of symbols listed in an autonomous list of symbol Tuples separated by `"]0["`.

ArgOrder	Type | Ref. Name	Description
Arg0	string Arg0Expression	Returns the entropy associated with the distribution of numbers included in the Argument STRC

Example 1: Entropy(<FiltredHist.LIST)

LanguageEntropy: Returns the entropy [0,1] of a Language described with a LanguageStruct as: Symb1]1[Freq1]2[Pos11]2[Pos12]2[...Pos1N]0[...]0[SymbLast]1[FreqLast]2[PosLast1]2[PosLast2]2[...]2[PosLastM

ArgOrder	Type | Ref. Name	Description
Arg0	string Arg0Expression	Returns the entropy [0,1] of a Language described with a STRC

Example 1: LanguageEntropy(<FiltredHist.LIST>)

FundamentalScale: Retrieves ....

ArgOrder	Type | Ref. Name	Description
Arg0	string `MultidimValueSTRC`	Converts the numerical (multidimensional) description MultidimValueSTRC into a description formed with elementary symbols of the same dimensionality
Arg1	string `ScaleTypeLIST`	Type of scale specifying the non-linearity of the scale
Arg3	String `ScaleMaxValLIST`	LIST with the Max Value for each scale's dimension
Arg4	string `ScaleResLIST`	LIST with the Resolution for each scale's dimension
Arg6	string `ScaleParamsSTRC`	STRC with Scale specific parameters for each scale's dimension
Arg7	string `FirstASCIICode`	(Optional) ASCII number of the `first character representing the elementary symbols`

Example 1: ConfigureSymbolicScale(<FiltredHist.LIST>, Hyperbolic, MinElem(<FiltredHist.LIST>), MaxElem(<FiltredHist.LIST>), <Resolution.FLOT>, <Inflection.INTG>, <ScaleParam.FLOT>)

SpaceProb2D: Creates and populates a structure containing the empirical probabilities of a process modeled as a bi-variate status registered history.

ArgOrder	Type | Ref. Name	Description
Arg0	string `MultidimValueSTRC`	Converts the numerical (multidimensional) description MultidimValueSTRC into a description formed with elementary symbols of the same dimensionality
Arg1	string `ScaleTypeLIST`	Type of scale specifying the non-linearity of the scale
Arg3	String `ScaleMaxValLIST`	LIST with the Max Value for each scale's dimension
Arg4	string `ScaleResLIST`	LIST with the Resolution for each scale's dimension
Arg6	string `ScaleParamsSTRC`	STRC with Scale specific parameters for each scale's dimension
Arg7	string `FirstASCIICode`	(Optional) ASCII number of the `first character representing the elementary symbols`

Example 1: SpaceProb2D (<DomainVar1.FLOT>]0[<DomainVar2.FLOT>, <MinValueVar1>]0[<MinValueVar2>]1[<MaxValueVar1>]0[<MaxValueVar2>, <ResolutionVar1>]0[<ResolutionVar2>, <ProcessHistoricValue.LIST>, ProcessHistoricMinValue]0[ProcessHistoricMaxValue, ProcessHistoricResolution, PastHorizonTime, ProjectionTime)

ConfigureSymbolicScale: Retrieves the set of repeated symbols within `TheText`.

ArgOrder	Type | Ref. Name	Description
Arg0	string `MultidimValueSTRC`	Converts the numerical (multidimensional) description MultidimValueSTRC into a description formed with elementary symbols of the same dimensionality
Arg1	string `ScaleTypeLIST`	Type of scale specifying the non-linearity of the scale
Arg2	string `ScaleMinValLIST`	LIST with the Min Value for each scale's dimension
Arg3	String `ScaleMaxValLIST`	LIST with the Max Value for each scale's dimension
Arg4	string `ScaleResLIST`	LIST with the Resolution for each scale's dimension
Arg5	string `ScaleInflecLIST`	LIST with the Inflection point for each scale's dimension
Arg6	string `ScaleParamsSTRC`	STRC with Scale specific parameters for each scale's dimension
Arg7	string `FirstASCIICode`	(Optional) ASCII number of the `first character representing the elementary symbols`

Example 1: ConfigureSymbolicScale(<FiltredHist.LIST>, Hyperbolic, MinElem(<FiltredHist.LIST>), MaxElem(<FiltredHist.LIST>), <Resolution.FLOT>, <Inflection.INTG>, <ScaleParam.FLOT>)

Lang1DimRepetitiveSymbols: Retrieves the set of repeated symbols within `TheText`.

ArgOrder	Type | Ref. Name	Description
Arg0	string `TheText`	Retrieves the set of most repeated symbols within TheText Format: Symb1]1[Freq1]2[Pos11]2[Pos12]2[...Pos1N]0[...]0[SymbLast]1[FreqLast]2[PosLast1]2[PosLast2]2[...]2[PosLastM
Arg1	int `MinRepetitions`	Minimal symbol appearances to be considered a repeated symbol
	string `Criterion`	Criterion used to select symbol-sequences: Length, Entropy, Space.
Arg2	string `PresentSymbolOrder`	Keyword indicating the Order in which the repeated symbols are presented: ByFreqRank, BySymbolSize, ByPosition, ByShowUpOrder
Arg3	string `WithSymmetry`	True to include Symmetric Symbol Sequences in the repetitions account. False otherwise
Arg4	string `Pattern1DimAttrib`	(Optional) Attribute's name to store the Pattern representation with elementary symbols

Example 1: = Lang1DimRepetitiveSymbols(<SymbolicSeries.STRC>, <RepetitionsRequired.INTG>, ByFreqRank, True, Pattern1Dim.STRN)

FilterPastAvg: Retrieves the weighted average of a list of values according to given parameters.

ArgOrder	Type | Ref. Name	Description
Arg0	string ProcessValue	The LIST of values to be filtered
Arg1	int `PastElements`	Number of past elements to be accounted in the average
Arg2	String `FiltParameters`	`FormatString`]1[LIST of conditions to exclude values from the average sepatated by]0[. `FormatString`: Use repeated char '#' to indicate the number of decimal places

Example 1: = FilterPastAvg(<ProcessValue.STRC>, 5, #.#####]1[NaN]0[Infinity]0[< -5]0[>= 5)

SymbolRelevance: `Assigns a relevance to a symbol according to the selected criterion`.

ArgOrder	Type | Ref. Name	Description
Arg0	string `TheSymbolOrSymmetric`	The character sequence to be evaluated
Arg1	`bool IsReversedOrder`	True if characters in `TheSymbolOrSymmetric are provided in reversed order`
Arg2	`string[] SymbolSequence`	`Array containing all the symbol-sequences describing TheSymbolOrSymmetric. Only used when criterion = ent (entropy)`
Arg3	`string Criterion`	`The criterion used to evaluate the symbol's relevance. Values are: len (symbol length), fill (Fill fraction, ent (entropy), lenfreq (length times frequency).`

Example 1: = FilterPastAvg(<ProcessValue.STRC>, 5, #.#####]1[NaN]0[Infinity]0[< -5]0[>= 5)

Control functions. Meta-functions

Arithmetic operations are represented with the same operators and syntaxes

STRCTgrow: Builds a one-dimensional structure with elements whose values are computed as indicated in the function's arguments.

ArgOrder	Type | Ref. Name	Description
Arg0	string TheSTRC	The Growing STRCT Attribute name. The component whose value is used to feed the Growing structure
Arg1	string GrowthDimSplitter	The Dimension where the STRCT will Grow. The splitter symbol to split the growing structure elements
Arg2	Int GrowthNumSteps	The number of the grow-steps
Arg3	String ResetSTRNG	Statement to signal whether or not the computation should erase the previous computations. <Reset> = True or <Reset> = False.
Arg4	string GrowthCompXpressn	The Expression that explains how to compute the new structure values
Arg5	string GeneralityOfXpressn	refers to the type of Expression evaluated

Example 1: STRCTgrow(<ProcessValue.STRC.Last><IC><InitialCond.STRC></>,]0[, 1, 1, <ProcessParams.STRC> * <ProcessValue.STRC><IC><InitialCond.STRC></>{<Last>} * (1 - <ProcessValue.STRC><IC><InitialCond.STRC></>{<Last>}), Compact)

Example 2: dSdt.LIST = STRCTgrow(<dSdt.LIST<IC><dSdt.Init.FLOT></>>,]0[, 1, 1, -1 * <e.Permisness.TREE{<Last>{0<RelDepth>0</>}}> * <r.InfctRate.FLOT{<Last>{0<RelDepth>0</>}}> * <S.LIST{<Last>}> * <Daily New Cases.LIST{<Last>}>, Compact)

DO: Repeats a computation a specified number of times.

ArgOrder	Type | Ref. Name	Description
Arg0	string ToRepeatXpressnsLIST	The Expression in which computation is to be repeated. Enclose ToRepeatXpressnsLIST in colons (‘The Expression’) to avoid misinterpreting the commas that may be part of The Expression with the commas used to separate arguments of the function DO.
Arg1	string XprssnsIndexOrderLIST	A LIST of integer numbers indicating the order in which multiple indexes are applied for nested loops.
Arg2	string SplitterDimTag	Splitter Tag that will separate the Computed Results obtained when the expression is applied. i.e.]1[.
Arg3		Symbolic Tag signaling the indexes used in de DO loop. i.e. IDX0 for a single loop or IDX0]0[IDX1 for two nested loops.
Arg4	string FirstIndxLIST	The Last index value for the loop steps. Enclose FirstIndxLIST in colons (‘FirstIndxLIST’) to avoid misinterpreting the commas that may be part of FirstIndxLIST with the commas used to separate arguments.
Arg5	string LasttIndxLIST	The Last index value for the loop steps. Enclose LasttIndxLIST in colons (‘LasttIndxLIST’) to avoid misinterpreting the commas that may be part of LasttIndxLIST with the commas used to separate arguments.
Arg6	string StepSizeLIST	The size of the loop steps
Arg7	string StoreResultInAttrib	The name of the Attribute where the result is to be stored. If the Attribute does not exist in the model, the result is stored in the attribute where the Function DO is.

Example 1: Status.2DProb.Hist.STRC = DO('SpaceProb(<Lambda.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>]0[<Permissiveness.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <ProbSpaceScale.STRC>, <ProbSpaceRes.LIST>, <Lambda.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <HistEventScale.LIST>, <HistEventRes.INTG>, <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>, <DataInfectedWithLag.LIST><@><Tag.STRN> = GenParams</@>){0:<Permissiveness.Status.Hist.LIST>{<DO.IDX0>}:<Lambda.Status.Hist.LIST>{<DO.IDX0>}]...[<HistEventRes.INTG><@><Tag.STRN> = GenParams</@>:<Permissiveness.Status.Hist.LIST>{<DO.IDX0>}:<Lambda.Status.Hist.LIST>{<DO.IDX0>}} / STRCElmValueSum(SpaceProb(<Lambda.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>]0[<Permissiveness.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <ProbSpaceScale.STRC>, <ProbSpaceRes.LIST>, <Lambda.LIST{<DO.IDX0> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <HistEventScale.LIST>, <HistEventRes.INTG>, <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>, <DataInfectedWithLag.LIST><@><Tag.STRN> = GenParams</@>){0:<Permissiveness.Status.Hist.LIST>{<DO.IDX0>}:<Lambda.Status.Hist.LIST>{<DO.IDX0>}]...[<HistEventRes.INTG><@><Tag.STRN> = GenParams</@>:<Permissiveness.Status.Hist.LIST>{<DO.IDX0>}:<Lambda.Status.Hist.LIST>{<DO.IDX0>}})', 3,]1[, IDX0, '(<LastDay.INTG> - <2DVariate.ProbMap.STRC><@><Tag.STRN> = GenParams</@>) - <DataInfectedWithLag.LIST><@><Tag.STRN> = GenParams</@>', '<LastDay.INTG> - <DataInfectedWithLag.LIST><@><Tag.STRN> = GenParams</@>', 1, Matrix2D.Hist.Pronostic.LIST)

Example 2: = Sim2DHistProbVar.STRC = DO('SpaceProb(<Var1.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>]0[<Var2.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <Var1.FLOT<@><Tag.STRN> = GenParams</@>{0}>]0[<Var2.FLOT<@><Tag.STRN> = GenParams</@>{0}>]1[<Var1.FLOT<@><Tag.STRN> = GenParams</@>{1}>]0[<Var2.FLOT<@><Tag.STRN> = GenParams</@>{1}>, <Var1Stts.INTG><@><Tag.STRN> = GenParams</@>]0[<Var2Stts.INTG><@><Tag.STRN> = GenParams</@>, <Var1.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <Prob.VarCumm.STRC><@><Tag.STRN> = GenParams</@>, <HistCount.ProjVar.INTG><@><Tag.STRN> = GenParams</@>, <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>, <ProjectTime.INTG>){0:<Var2Sttus.LIST{<DO.IDX0>}>:<Var1Sttus.LIST{<DO.IDX0>}>]...[<HistCount.ProjVar.INTG><@><Tag.STRN> = GenParams</@>:<Var2Sttus.LIST{<DO.IDX0>}>:<Var1Sttus.LIST{<DO.IDX0>}>} / STRCElmValueSum(SpaceProb(<Var1.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>]0[<Var2.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <Var1.FLOT<@><Tag.STRN> = GenParams</@>{0}>]0[<Var2.FLOT<@><Tag.STRN> = GenParams</@>{0}>]1[<Var1.FLOT<@><Tag.STRN> = GenParams</@>{1}>]0[<Var2.FLOT<@><Tag.STRN> = GenParams</@>{1}>, <Var1Stts.INTG><@><Tag.STRN> = GenParams</@>]0[<Var2Stts.INTG><@><Tag.STRN> = GenParams</@>, <Var1.FLOT{<DO.IDX0> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>]...[<DO.IDX0>}>, <Prob.VarCumm.STRC><@><Tag.STRN> = GenParams</@>, <HistCount.ProjVar.INTG><@><Tag.STRN> = GenParams</@>, <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>, <ProjectTime.INTG>){0:<Var2Sttus.LIST{<DO.IDX0>}>:<Var1Sttus.LIST{<DO.IDX0>}>]...[<HistCount.ProjVar.INTG><@><Tag.STRN> = GenParams</@>:<Var2Sttus.LIST{<DO.IDX0>}>:<Var1Sttus.LIST{<DO.IDX0>}>})', 3,]1[, IDX0, '(<HistDays.INTG> - <Sim2DHistProbVar.STRC><@><Tag.STRN> = GenParams</@>) - <ProjectTime.INTG>', '<HistDays.INTG> - <ProjectTime.INTG>', 1, Matrix2D.Hist.Pronostic.LIST)

SWC: HyperFunction that executes a series of other functions.

ArgOrder	Type | Ref. Name	Description
Arg0	string `Argument0STRNG`	LIST of Attributes to be computed.
Arg1	String `StepSize`	The size of the step for the change of indexes.
Arg2	Int `Iterations`	The number of iterations to be computed.
Arg3	string `ResetSTRNG`	Sentene indicating to reset or not the starting index. <Reset> = True or <Reset> = False
Arg4	string `LastProcessed`	The attribute where the last processed index is recorded.

ArgOrder	Type | Ref. Name	Description
Arg0	string AttribArgument0STRNG	`LIST of Attributes with Expressions to be SWC Computed.` `i.e.: t.LIST]...[R.LIST`
Arg1	Int `StepSize`	Idle Argument for future use.
Arg2	int `Iterations`	The number of times the computation of the `LIST of Attributes` is performed.
Arg3	`String` `ResetSTRNG`	Statement to signal whether or not the computation should erase the previous computations. <Reset> = True or <Reset> = False.
Arg4	string `AttribLastProcessed`	The name of the attribute to record the last iteration processed.

`= SWC(Argument0STRNG, StepSize, Iterations, ResetSTRNG, LastProcessed)`

Example: SWC.EXEC = SWC(t.LIST]...[R.LIST, 1, <Days.INTG> - <LastDay.INTG>, <Reset> = False, LastDay.INTG)

References

1. Febres G, Jaffe K, Gershenson C. Complexity measurement of natural and artificial languages. Complexity. 2015;20: 429–453. doi:10.1002/cplx.21529