INScore
Math Expressions Reference
v.1.0

D. Fober
GRAME
Centre national de création musicale

September 2, 2016

Contents

Chapter 1

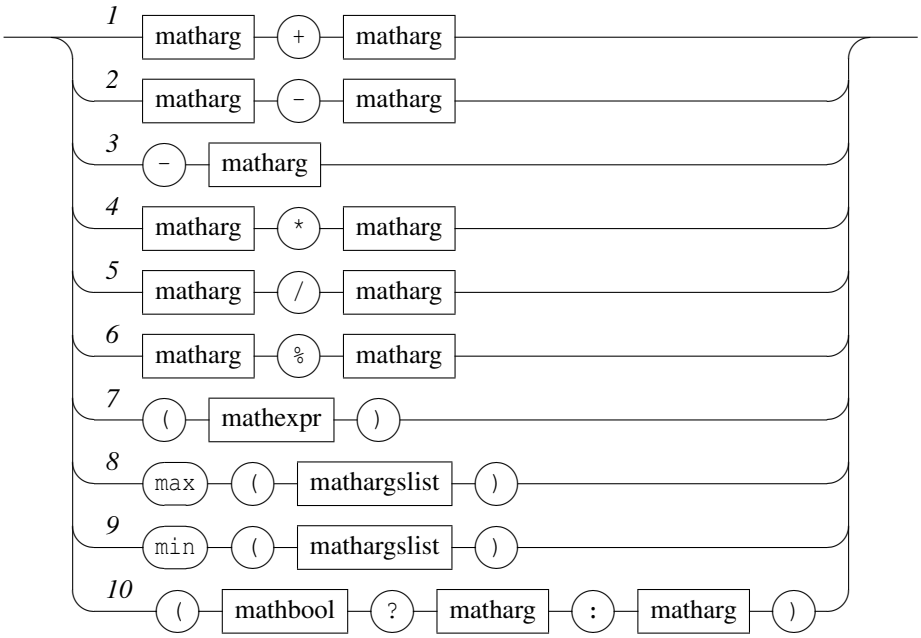
Mathematical expressions

Since INScore version 1.20, mathematical expressions have been introduced as messages arguments. These expressions allows to compute values at parsing time.

1.1 Operators

Basic mathematical expressions are supported. They are listed below.

mathexpr



- **1**: addition.
- **2**: substraction.
- **3**: negation.
- **4**: multiplication.
- **5**: division.
- **6**: modulo.

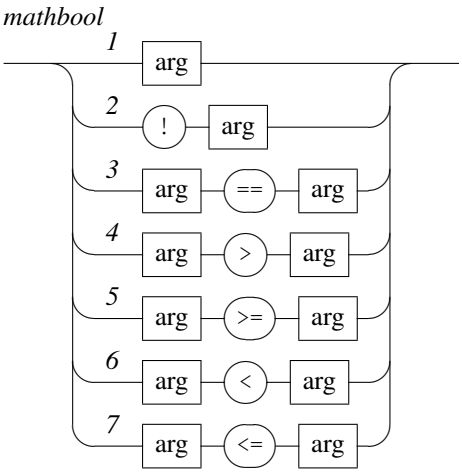
- **7**: parenthesis, to be used for evaluation order.
- **8**: gives the maximum value.
- **9**: gives the minimum value.
- **10**: conditional form: returns the first `matharg` if `mathbool` is true, otherwise returns the second one.

EXAMPLE

Some simple mathematical expressions used as message parameters:

```
/ITL/scene/myObject x 0.5 * 0.2;
/ITL/scene/myObject y ($var ?  1 :  -1);
```

Boolean operations are the following:



- **1**: evaluate the argument as a boolean value.
- **2**: evaluate the argument as a boolean value and negates the result.
- **3**: check if the arguments are equal.
- **4**: check if the first argument is greater than the second one.
- **5**: check if the first argument is greater or equel to the second one.
- **6**: check if the first argument is less than the second one.
- **7**: check if the first argument is less or equal to the second one.

EXAMPLE

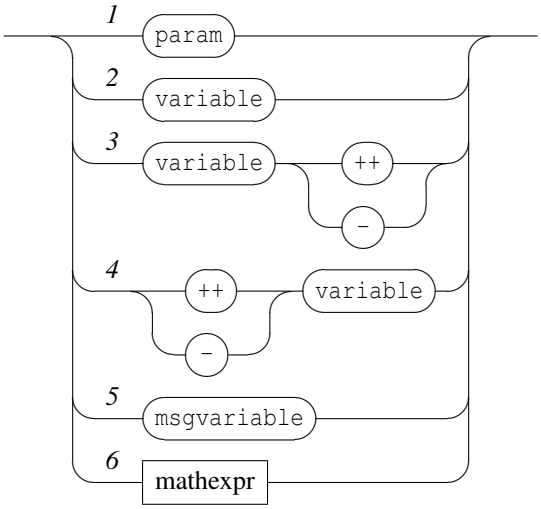
Compare two variables:

```
/ITL/scene/myObject x ($var1 > $var2 ?  1 :  -1);
```

1.2 Arguments

Arguments of mathematical operations are the following:

matharg



- **1**: any message parameter.
- **2**: a variable value.
- **3**: a variable that is post incremented or post decremented.
- **4**: a variable that is pre incremented or pre decremented.
- **5**: a message based variable.
- **6**: a mathematical expression.

1.3 Polymorphism

Since INScore's parameters are polymorphic, the semantic of the operations are to be defined notably when applied to non numeric arguments. Actually, a basic OSC message parameter type is between `int32`, `float32` and `string`. However, due to the extension of the scripting language, parameters could also be arrays of any type, including mixed types (e.g. resulting from variable declarations).

1.3.1 Numeric values

For numeric arguments, automatic type conversion is applied with a precedence of `float32` i.e. when one of the argument's type is `float32`, the result is also `float32` (see Table 1.1).

arg1	arg2	output
int32	int32	int32
float32	int32	float32
int32	float32	float32
float32	float32	float32

Table 1.1: Numeric operations output

1.3.2 Strings

For `string` parameters, we introduce a conversion operation *length*(`string`) that evaluates to `int32`, and that is applied depending on the operation type (see Tables 1.2 and 1.3). Note that we'll refer to numeric

values as `num` to represent `int32` or `float32` that are handled similarly in this context.

operation	evaluates to	comment
`string` + `string`	`string`	concatenate the two strings
`string` *op* `string`	*length*(`string`) *op* *length*(`string`) where *op* is in [- * / %]	operates on strings length
-`string`	`string`	gives the string unchanged
@max(`string string`)	`string`	select the largest string
@min(`string string`)	`string`	select the smallest string
prefixed or postfixed `string`	`string`	gives the string unchanged

Table 1.2: Operations on strings

operation	evaluates to	comment
`string` + `num`	`string` + *string*(`num`)	`num` is converted to string
`num` + `string`	*string*(`num`) + `string`	"
`string` *op* `num`	*length*(`string`) *op* `num`	operates on numeric values
`num` *op* `string`	`num` *op* *length*(`string`) where *op* is in [- * / %]	"
@max(`string num`)	@max(*length*(`string`) `num`)	"
@min(`string num`)	@min(*length*(`string`) `num`)	"

Table 1.3: Operations on strings and numeric values: apart the '+' operator, strings are converted to numeric values using their length.

And finally, a boolean operation on `string` is evaluates as a boolean operation on *length*(`string`).

1.3.3 Arrays

For arrays, the operation is distributed inside the array elements:

$$arg\ op\ [v_1\ ...\ v_n] := [arg\ op\ v_1\ ...\ arg\ op\ v_n]$$

or

$$[v_1\ ...\ v_n]\ op\ arg := [v_1\ op\ arg\ ...\ v_n\ op\ arg]$$

When both parameters are arrays, the operation is distributed from one array elements to the other array elements, up to the smaller array size:

$$i < j \quad \wedge \quad [a_1\ ...\ a_i]\ op\ [b_1\ ...\ b_j] := [a_1\ op\ b_1\ ...\ v_i\ op\ b_i\ b_{i+1}\ ...\ b_j]$$

$$i > j \quad \wedge \quad [a_1\ ...\ a_i]\ op\ [b_1\ ...\ b_j] := [a_1\ op\ b_1\ ...\ v_i\ op\ b_j\ a_{j+1}\ ...\ a_i]$$

Boolean operations on arrays are evaluated as the logical *and* of it's element's boolean values and returns false when the arrays sizes differs.