















Ptolemy 库	名称	作用	BMC 编码
Source s Library		生成一个稳定递增或递减的数值序列.	<p>设左侧 init 和 step 输入端口所连接 relation 分别为 rIn 和 rSIn, 右侧输出端口所连接 relation 为 rOut, 状态为 ST_Ramp, 内部参数分别为 init, step, limit 则</p> <p>情形一：当 limit≤0 时</p> <p>①没有输入连接时</p> $(ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1)$ $\vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ <p>②init 连接, step 未连接时</p> $(rIn[i]! = nil \wedge rOut[i] = rIn[i])$ $\vee \left( rIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right)$ <p>③init 未连接, step 连接时</p> $\left( rSIn[i]! = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + rSIn[i-1] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right)$ $\vee \left( rSIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right)$ <p>④init 连接, step 连接时</p> $(rIn[i]! = nil \wedge rOut[i] = rIn[i])$ $\vee \left( rIn[i] = nil \wedge \left( rSIn[i]! = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + rSIn[i-1] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \vee \left( rSIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \right)$ <p>情形二：当 limit&gt;0 时</p> <p>①没有输入连接时</p> $(ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1)$ $\vee (limit > ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ $\vee (ST\_Ramp[i-1] \geq limit \wedge rOut[i] = nil \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ <p>②init 连接, step 未连接时</p> $\left( (ST\_Ramp[i-1] < limit) \wedge \left( (rIn[i]! = nil \wedge rOut[i] = rIn[i] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \vee (rIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right)) \right) \right)$ $\vee (ST\_Ramp[i-1] \geq limit \wedge rOut[i] = nil \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ <p>③init 未连接, step 连接时</p> $\left( (ST\_Ramp[i-1] < limit) \wedge \left( rSIn[i]! = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + rSIn[i-1] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \vee \left( rSIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \right)$ $\vee (ST\_Ramp[i-1] \geq limit \wedge rOut[i] = nil \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ <p>④init 连接, step 连接时.</p> $\left( (ST\_Ramp[i-1] < limit) \wedge \left( rIn[i]! = nil \wedge rOut[i] = rIn[i] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1 \right) \vee \left( rIn[i] = nil \wedge \left( rSIn[i]! = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + rSIn[i-1] \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \vee \left( rSIn[i] = nil \wedge \left( (ST\_Ramp[i-1] = 0 \wedge rOut[i] = init \wedge ST\_Ramp[i] = 1) \vee (ST\_Ramp[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1) \right) \right) \right) \right)$ $\vee (ST\_Ramp[i-1] \geq limit \wedge rOut[i] = nil \wedge ST\_Ramp[i] = ST\_Ramp[i-1] + 1)$ <p>注：参数 init 和 step 默认情况下分别是 0 和 1, 如果更改, 则可从 Ramp</p>




			actor 的 property 中读取得到.
		<p>生成一个由参数所设定的值或者表达式（这里是 1）.</p>	<p>设右侧输出端口所连接 relation 为 rOut, 内部参数分别为 parameter, limit 则</p> <p>情形一: limit≤0 时</p> $rOut[i] = parameter$ <p>情形二: limit&gt;0 时</p> $c[i] = c[i-1] + 1 \wedge \begin{pmatrix} (c[i] \leq limit \wedge rOut[i] = parameter) \\ \vee (c[i] > limit \wedge rOut[i] = nil) \end{pmatrix}$ $c[0] = 0$ <p>注: 参数的值默认情况下是 1, 如果更改, 可从模型的 MoML xml 文件中 Const actor 的 property 中读取得到.</p>
		<p>生成一个由指定值构成的序列（这里是生成 1 构成的序列）.</p>	<p>情形一: 设左侧端口所连接 relation 为 rIn, 右侧输出端口所连接 relation 为 rOut, 数组 s 存储 Sequence 中的数据则</p> <p>①repeat 为 true (repeat 优先级最高) 时</p> $((rIn[i] = false \vee rIn[i] = nil) \wedge rOut[i] = nil)$ $\vee \left( rIn[i] = true \wedge \begin{pmatrix} (index[i] \geq 0 \wedge index[i] < SeqNum) \wedge ((rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)) \\ \vee ((index[i] \geq SeqNum) \wedge (rOut[i] = s_1) \wedge (index[i+1] = 2)) \end{pmatrix} \right)$ <p>②holdLastPut 为 false, repeat 为 false 时</p> $((rIn[i] = false \vee rIn[i] = nil) \wedge rOut[i] = nil)$ $\vee \left( rIn[i] = true \wedge \begin{pmatrix} (index[i] \geq 0 \wedge index[i] < SeqNum) \wedge ((rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)) \\ \vee ((index[i] \geq SeqNum) \wedge (rOut[i] = nil) \wedge (index[i+1] = index[i])) \end{pmatrix} \right)$ <p>③holdLastPut 为 true, repeat 为 false 时</p> $((rIn[i] = false \vee rIn[i] = nil) \wedge rOut[i] = nil)$ $\vee \left( rIn[i] = true \wedge \begin{pmatrix} (index[i] \geq 0 \wedge index[i] < SeqNum) \wedge ((rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)) \\ \vee ((index[i] \geq SeqNum) \wedge (rOut[i] = s_{SeqNum}) \wedge (index[i+1] = index[i])) \end{pmatrix} \right)$ <p>注: 参数 index 表示序列的索引, 参数 vindex 对应于 Sequence 所给出的序列中索引为 index 的元素 s1, s2, sv3..., 且这些值可从模型的 MoML xml 文件中 Sequence actor 的 property 中读取得到</p> <p>情形二: 设左侧端口没有连接 relation, 右侧输出端口所连接 relation 为 rOut, 则</p> <p>①repeat 为 true (repeat 优先级最高) 时</p> $((index[i] \geq 0 \wedge index[i] < SeqNum) \wedge rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)$ $\vee ((index[i] \geq SeqNum) \wedge (rOut[i] = s_1) \wedge (index[i+1] = 2))$ <p>②holdLastPut 为 false, repeat 为 false 时</p> $((index[i] \geq 0 \wedge index[i] < SeqNum) \wedge rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)$ $\vee ((index[i] \geq SeqNum) \wedge (rOut[i] = nil) \wedge (index[i+1] = index[i]))$ <p>③holdLastPut 为 true, repeat 为 false 时</p> $((index[i] \geq 0 \wedge index[i] < SeqNum) \wedge rOut[i] = s_{index} \wedge index[i+1] = index[i]+1)$ $\vee ((index[i] \geq SeqNum) \wedge (rOut[i] = s_{SeqNum}) \wedge (index[i+1] = index[i]))$ <p>注: 参数 index 表示序列的索引, 参数 vindex 对应于 Sequence 所给出的序列中索引为 index 的元素, 且该值可从模型的 MoML xml 文件中 Sequence actor 的 property 中读取得到.</p>
		<p>生成一个常字符串序列.</p>	<p>该 actor 属于 Const 的子类 actor, 因此编码方式和 Const 相同</p>





Math Library		加上+输入端口处的输入数据，并减去-输入端口处的输入数据	<p>情形一：假设+端口没有 relation 连接，而-端口所连接的 relation 为 rMin，而右侧 out 端口的 relation 为 rOut，创建数组 sM 和 nilNumberM 来辅助编码则：</p> $\bigwedge_{j=1}^{inMinusPort.width} \left( (rMin_j[i]! = nil \wedge sM_j[i] = rMin_j \wedge nilNumberM_j[i] = 0) \vee (rMin_j[i] = nil \wedge sM_j[i] = 0 \wedge nilNumberM_j[i] = 1) \right) \wedge \left( \left( rOut[i] = - \sum_{j=1}^{inMinusPort.width} sM_j[i] \wedge \sum_{j=1}^{inMinusPort.width} nilNumberM_j[i]! = inMinusPort.width \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inMinusPort.width} nilNumberM_j[i]! = inMinusPort.width \right) \right)$ <p>情形二：假设-端口没有 relation 连接，而+端口所连接的 relation 为 rAIn，而右侧 out 端口的 relation 为 rOut，创建数组 sA 和 nilNumberA 来辅助编码则：</p> $\bigwedge_{j=1}^{inAddPort.width} \left( (rAIn_j[i]! = nil \wedge sA_j[i] = rAIn_j \wedge nilNumberA_j[i] = 0) \vee (rAIn_j[i] = nil \wedge sA_j[i] = 0 \wedge nilNumberA_j[i] = 1) \right) \wedge \left( \left( rOut[i] = \sum_{j=1}^{inAddPort.width} sA_j[i] \wedge \sum_{j=1}^{inAddPort.width} nilNumberA_j[i]! = inAddPort.width \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inAddPort.width} nilNumberA_j[i]! = inAddPort.width \right) \right)$ <p>情形三：假设+端口所连接的 relation 为 rAIn，而-端口所连接的 relation 为 rMIn，而右侧 out 端口的 relation 为 rOut，创建数组 sA, sM 和 nilNumberA, nilNumberM 来辅助编码则：</p> $\bigwedge_{j=1}^{inAddPort.width} \left( (rAIn_j[i]! = nil \wedge sA_j[i] = rAIn_j \wedge nilNumberA_j[i] = 0) \vee (rAIn_j[i] = nil \wedge sA_j[i] = 0 \wedge nilNumberA_j[i] = 1) \right) \wedge \bigwedge_{j=1}^{inMinusPort.width} \left( (rMIn_j[i]! = nil \wedge sM_j[i] = rMIn_j \wedge nilNumberM_j[i] = 0) \vee (rMIn_j[i] = nil \wedge sM_j[i] = 0 \wedge nilNumberM_j[i] = 1) \right) \wedge \left( \left( rOut[i] = \sum_{j=1}^{inAddPort.width} sA_j[i] - \sum_{j=1}^{inMinusPort.width} sM_j[i] \wedge \left( \sum_{j=1}^{inAddPort.width} nilNumberA_j[i]! = inAddPort.width \vee \sum_{j=1}^{inMinusPort.width} nilNumberM_j[i]! = inMinusPort.width \right) \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inAddPort.width} nilNumberA_j[i]! = inAddPort.width \wedge \sum_{j=1}^{inMinusPort.width} nilNumberM_j[i]! = inMinusPort.width \right) \right)$ <p>注：inPort.width 可通过看 BooleanSwitch actor 的 input 连接几条 relation 获知。</p>
		给输入乘以一个由参数所设定的常数 c(这里乘以-1)	<p>设左、右两侧端口所对应的 relation 分别是 rIn 和 rOut, Scale 中的常数为 c, 则</p> $(rOut[i] = rIn[i] * c \wedge rIn[i]! = nil) \vee (rIn[i] = nil \wedge rOut[i] = nil)$ <p>注：参数 c 默认情况下是 1，如果更改可从模型的 MoMLxml 文件中 Scale actor 的 Property 中读取得到</p>
		计算输入值的绝对值。	<p>设左、右两侧端口所对应的 relation 分别是 rIn 和 rOut, 则</p> $(rOut[i] = abs(rIn[i]) \wedge rIn[i]! = nil) \vee (rIn[i] = nil \wedge rOut[i] = nil)$
		乘以×输入端口处的输入数据，并除以÷输入端口处的输入数据。	<p>情形一：假设*端口没有 relation 连接，而÷端口所连接的 relation 为 rDIn，而右侧 out 端口的 relation 为 rOut，创建数组 sD 和 nilNumberD 来辅助编码，则：</p> $\bigwedge_{j=1}^{inDividePort.width} \left( (rDIn_j[i]! = nil \wedge sD_j[i] = rDIn_j \wedge nilNumberD_j[i] = 0) \vee (rDIn_j[i] = nil \wedge sD_j[i] = 0 \wedge nilNumberD_j[i] = 1) \right) \wedge \left( \left( rOut[i] = \prod_{j=1}^{inDividePort.width} (1 / sD_j[i]) \wedge \sum_{j=1}^{inDividePort.width} nilNumberD_j[i]! = inDividePort.width \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inDividePort.width} nilNumberD_j[i]! = inDividePort.width \right) \right)$ <p>情形二：假设÷端口没有 relation 连接，而*端口所连接的 relation 为 rMIn，而右侧 out 端口的 relation 为 rOut，创建数组 sM 和 nilNumberM 来辅助编码，则：</p>

		$\bigwedge_{j=1}^{inMultiPort.width} \left( (rMin_j[i]! = nil \wedge sM_j[i] = rMin_j \wedge nilNumberM_j[i] = 0) \vee (rMin_j[i] = nil \wedge sM_j[i] = 0 \wedge nilNumberM_j[i] = 1) \right)$ $\wedge \left( \left( rOut[i] = \prod_{j=1}^{inMultiPort.width} sM_j[i] \wedge \sum_{j=1}^{inMultiPort.width} nilNumberM_j[i]! = inMultiPort.width \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inMultiPort.width} nilNumberM_j[i] = inMultiPort.width \right) \right)$ <p>情形三：假设*端口所连接的 relation 为 rMin，而÷端口所连接的 relation 为 rDIn，而右侧 out 端口的 relation 为 rOut，创建数组 sM，sD 和 nilNumberM，nilNumberD 来辅助编码，则：</p> $\bigwedge_{j=1}^{inMultiPort.width} \left( (rMin_j[i]! = nil \wedge sM_j[i] = rMin_j \wedge nilNumberM_j[i] = 0) \vee (rMin_j[i] = nil \wedge sM_j[i] = 0 \wedge nilNumberM_j[i] = 1) \right)$ $\wedge \bigwedge_{j=1}^{inDividePort.width} \left( (rDIn_j[i]! = nil \wedge sD_j[i] = rDIn_j \wedge nilNumberD_j[i] = 0) \vee (rDIn_j[i] = nil \wedge sD_j[i] = 0 \wedge nilNumberD_j[i] = 1) \right)$ $\wedge \left( \left( rOut[i] = \prod_{j=1}^{inMultiPort.width} sM_j[i] * \prod_{j=1}^{inDividePort.width} (1 / sD_j[i]) \wedge \left( \sum_{j=1}^{inMultiPort.width} nilNumberM_j[i]! = inMultiPort.width \vee \sum_{j=1}^{inDividePort.width} nilNumberD_j[i]! = inDividePort.width \right) \right) \vee \left( rOut[i] = nil \wedge \sum_{j=1}^{inDividePort.width} nilNumberD_j[i] = inDividePort.width \wedge \sum_{j=1}^{inMultiPort.width} nilNumberM_j[i] = inMultiPort.width \right) \right)$ <p>注：inPort.width 可通过看 MultiplyDivide actor 的 input 连接几条 relation 获知。</p>
	<p>输出从上次 reset 端口处接收到 true 值时到当前 tick，左侧端口所接收到的所有输入的和。</p>	<p>情形一：假设 reset 端口处没有连接，而左侧端口处所连接的 relation 为 rIn，右侧输出端口处所连接的 relation 为 rOut，创建数组 s，n，变量 sum 来辅助编码，则</p> $\bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i]! = nil \wedge s_j[i] = rIn_j \wedge n_j[i] = 1) \vee (rIn_j[i] = nil \wedge s_j[i] = 0 \wedge n_j[i] = 0) \right)$ $\wedge \left( \left( \sum_{j=1}^{inPort.width} n_j[i]! = 0 \wedge rOut = sum[i] \wedge sum[i] = sum[i-1] + \sum_{j=1}^{inPort.width} s_j[i] \right) \vee \left( \sum_{j=1}^{inPort.width} n_j[i] = 0 \wedge rOut = sum[i] \wedge sum[i] = sum[i-1] \right) \right)$ <p>注：sum[0] = init 原模型有初始值</p> <p>情形二：假设 reset 端口处有连接 rRIn，而左侧端口处所连接的 relation 为 rIn，右侧输出端口处所连接的 relation 为 rOut，创建数组 s，n，变量 sum 来辅助编码，则</p> $\left( rRIn[i] = true \wedge \bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i]! = nil \wedge s_j[i] = rIn_j \wedge n_j[i] = 1) \vee (rIn_j[i] = nil \wedge s_j[i] = 0 \wedge n_j[i] = 0) \right) \right)$ $\wedge \left( \left( \sum_{j=1}^{inPort.width} n_j[i]! = 0 \wedge rOut = sum[i] \wedge sum[i] = init + \sum_{j=1}^{inPort.width} s_j[i] \right) \vee \left( \sum_{j=1}^{inPort.width} n_j[i] = 0 \wedge rOut = sum[i] \wedge sum[i] = init \right) \right)$ $\vee \left( \left( rRIn[i] = false \vee rRIn[i] = nil \right) \wedge \bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i]! = nil \wedge s_j[i] = rIn_j \wedge n_j[i] = 1) \vee (rIn_j[i] = nil \wedge s_j[i] = 0 \wedge n_j[i] = 0) \right) \right)$ $\wedge \left( \left( \sum_{j=1}^{inPort.width} n_j[i]! = 0 \wedge rOut = sum[i] \wedge sum[i] = sum[i-1] + \sum_{j=1}^{inPort.width} s_j[i] \right) \vee \left( \sum_{j=1}^{inPort.width} n_j[i] = 0 \wedge rOut = sum[i] \wedge sum[i] = sum[i-1] \right) \right)$ <p>注：sum[0] = init</p> <p>注：inPort.width 可通过看 Accumulator actor 的 input 连接几条 relation 获知。</p>
	<p>输出从上次 reset 端口处接收到 true 值时到当前 tick，左端口所接收到的所有输入的平均值。</p>	<p>情形一：假设 reset 端口处没有连接，而左侧端口处所连接的 relation 为 rIn，右侧端口处所连接的 relation 为 rOut，创建变量 sum，count 来辅助编码，则</p>








		<p>①输入为Int型时:</p> $\left( \begin{aligned} &rIn[i] \neq nil \wedge \left( \begin{aligned} &(rOut[i] = sum[i]/count[i] \wedge sum[i] \geq 0) \\ &\vee (rOut[i] = -to\_int((-sum[i])/count[i]) \wedge sum[i] < 0) \end{aligned} \right) \right) \\ &\wedge sum[i] = sum[i-1] + rIn[i] \wedge count[i+1] = count[i] + 1 \end{aligned} \right)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil \wedge sum[i] = sum_{i-1} \wedge count_{i+1} = count[i])$ <p>②输入非Int型:</p> $(rIn[i] \neq nil \wedge rOut[i] = sum[i]/count[i] \wedge sum[i] = sum[i-1] + rIn[i] \wedge count[i+1] = count[i] + 1)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil \wedge sum[i] = sum[i-1] \wedge count[i+1] = count[i])$ <p>注: <math>sum[0] = 0, count[1] = 1</math></p> <p>情形二: 假设 reset 端口处有连接 rRIn,左侧端口处所连接的 relation 为 rIn, 右侧端口处所连接的 relation 为 rOut, 创建变量 sum, count 来辅助编码, 则</p> <p>①输入为Int型时:</p> $\left( \begin{aligned} &\left( \begin{aligned} &(rRIn[i] = true \wedge rOut[i] = rIn[i] \wedge sum[i] = rIn[i] \wedge count[i] = 1) \\ &(rRIn[i] = false \vee rRIn[i] = nil) \end{aligned} \right) \\ &\wedge \left( \begin{aligned} &(rOut[i] = sum[i]/count[i] \wedge sum[i] \geq 0) \\ &\vee (rOut[i] = -to\_int((-sum[i])/count[i]) \wedge sum[i] < 0) \end{aligned} \right) \\ &\wedge sum[i] = sum[i-1] + rIn[i] \wedge count[i+1] = count[i] + 1 \end{aligned} \right)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil \wedge sum[i] = sum[i-1] \wedge count[i+1] = count[i])$ <p>②输入非Int型:</p> $\left( \begin{aligned} &(rIn[i] \neq nil) \wedge \left( \begin{aligned} &(rRIn[i] = true \wedge rOut[i] = rIn[i] \wedge sum[i] = rIn[i] \wedge count[i] = 1) \\ &\vee (rRIn[i] = false \vee rRIn[i] = nil) \wedge rOut[i] = sum[i]/count[i] \end{aligned} \right) \\ &\wedge sum[i] = sum[i-1] + rIn[i] \wedge count[i+1] = count[i] + 1 \end{aligned} \right)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil \wedge sum[i] = sum[i-1] \wedge count[i+1] = count[i])$ <p>注: <math>sum[0] = 0, count[1] = 1</math></p>
	<p>将输入值限制在由 top 和 bottom 所指定的范围内. 如果输入值介于 top 和 bottom 之间, 则输出该输入值; 如果输入值比 top 大, 则输出 top; 如果输入值比 bottom 小, 则输出 bottom.</p>	<p>设左、右两侧连接的 relation 分别为 rIn 和 rOut, Limiter 中内部参数 top, bottom, 则</p> $\left( \begin{aligned} &rIn[i] \neq nil \wedge \left( \begin{aligned} &(rIn[i] \geq bottom \wedge rIn[i] \leq top \wedge rOut[i] = rIn[i]) \\ &\vee (rIn[i] > top \wedge rOut[i] = top) \vee (rIn[i] < bottom \wedge rOut[i] = bottom) \end{aligned} \right) \end{aligned} \right)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil)$
	<p>输出所给出的表达式所计算得到的值(这里给出的表达式是 x-2*y).</p>	<p>设左、右两侧连接的 relation 分别为 rxIn, ryIn 和 rOut, 则</p> $\left( (rOut[i] = rxIn[i] - 2 * ryIn[i]) \wedge (rxIn[i] \neq nil \wedge ryIn[i] \neq nil) \right)$ $\vee ((rOut[i] = nil) \wedge (rxIn[i] = nil \vee ryIn[i] = nil))$ <p>(注: 表达式可以从模型的 xml 文件中读取得到, 具体是: Expression actor--StringAttribute 类型---名为“expression”的 property 对象的 value.)</p>
	<p>输出当前 tick 的输入 token 中的最大值.</p>	<p>情形一: 设左侧连接的 relation 为 rIn, 而右侧连接的 relation 只有 rMax, 且在 channelNumber 端口处没有 relation 连接, 创建数组 s 辅助编码, 则</p>





		$\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMax[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \geq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] < rIn_j[i] \wedge s_j[i] = rIn_j[i] \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \right) \right) \right)$ $\wedge rMax_i = s_{inPort.width}[i]$
		<p>情形二：设左侧连接的 relation 为 rIn，而右侧连接的 relation 只有 rMaxCN，且在 maximumValue 端口处没有 relation 连接，创建数组 s，c 辅助编码，则</p> $\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMaxCN[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \geq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] < rIn_j[i] \wedge s_j[i] = rIn_j[i] \wedge c_j[i] = j \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \right)$ $\wedge rMaxCN[i] = c_{inPort.width}[i]$
		<p>情形三：设左侧连接的 relation 为 rIn，而右侧连接的 relation 为 rMax，rMaxCN，创建数组 s，c 辅助编码，则</p> $\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMaxCN[i] = nil \wedge rMax[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \geq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] < rIn_j[i] \wedge s_j[i] = rIn_j[i] \wedge c_j[i] = j \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \right)$ $\wedge rMaxCN[i] = c_{inPort.width}[i] \wedge rMax[i] = s_{inPort.width}[i]$
		<p>输出当前 tick 的输入 token 中的最小值.</p>
		<p>情形一：设左侧连接的 relation 为 rIn 而右侧连接的 relation 只有 rMin，且在 channelNumber 端口处没有 relation 连接，创建数组 s 辅助编码，则</p>






		$\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMin[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \leq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] > rIn_j[i] \wedge s_j[i] = rIn_j[i] \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \right) \right) \right)$ $\wedge rMin[i] = s_{inPort.width}[i]$	<p>情形二：设左侧连接的 relation 为 rIn，而右侧连接的 relation 只有 rMinCN，且在 minimumValue 端口处没有 relation 连接，创建数组 s，c 辅助编码，则</p> $\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMinCN[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \leq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] > rIn_j[i] \wedge s_j[i] = rIn_j[i] \wedge c_j[i] = j \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \right)$ $\wedge rMinCN[i] = c_{inPort.width}[i]$	<p>情形三：设左侧连接的 relation 为 rIn，而右侧连接的 relation 为 rMin，rMinCN，创建数组 s，c 辅助编码，则</p> $\left( \bigwedge_{j=1}^{inPort.width} rIn_j[i] = nil \wedge rMinCN[i] = nil \wedge rMin[i] = nil \right)$ $\left( \bigvee_{j=1}^{inPort.width} rIn_j[i] \neq nil \right)$ $\vee \bigwedge_{j=1}^{inPort.width} \left( \left( rIn_j[i] \neq nil \wedge \left( \left( not(s_{j-1}[i] = s_0[i]) \wedge \left( \left( s_{j-1}[i] \leq rIn_j[i] \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \vee \left( s_{j-1}[i] > rIn_j[i] \wedge s_j[i] = rIn_j[i] \wedge c_j[i] = j \right) \right) \right) \right) \vee \left( rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i] \wedge c_j[i] = c_{j-1}[i] \right) \right) \right)$ $\wedge rMinCN[i] = c_{inPort.width}[i] \wedge rMin[i] = s_{inPort.width}[i]$
		输出从模型开始运行到当前 tick 在输入端口处可看到的最大值。	设左、右两侧所连接的 relation 分别为 rIn 和 rOut，创建变量 index，max 辅助编码，则： $index[i+1] = index[i] + 1 \wedge \left( \left( rIn[i] \neq nil \wedge \left( \left( (index[i] = 1 \wedge rOut[i] = \max[i+1] \wedge \max[i+1] = rIn[i]) \vee \left( (index[i] > 1) \wedge \left( \left( rIn[i] > \max[i] \wedge rOut[i] = \max[i+1] \wedge \max[i+1] = rIn[i] \right) \vee \left( rIn[i] \leq \max[i] \wedge rOut[i] = \max[i+1] \wedge \max[i+1] = \max[i] \right) \right) \right) \right) \vee \left( rIn[i] = nil \wedge rOut[i] = nil \wedge \max[i+1] = \max[i] \right) \right) \right)$	
		输出从模型开始运行到当前 tick 在输入端口处可看到的最小值。	设左、右两侧所连接的 relation 分别为 rIn 和 rOut，创建变量 index，max 辅助编码，则： $index_{i+1} = index_i + 1 \wedge \left( \left( rIn[i] \neq nil \wedge \left( \left( (index[i] = 1 \wedge rOut[i] = \min[i+1] \wedge \min[i+1] = rIn[i]) \vee \left( (index[i] > 1) \wedge \left( \left( rIn[i] < \min[i] \wedge rOut[i] = \min[i+1] \wedge \min[i+1] = rIn[i] \right) \vee \left( rIn[i] \geq \min[i] \wedge rOut[i] = \min[i+1] \wedge \min[i+1] = \min[i] \right) \right) \right) \right) \vee \left( rIn[i] = nil \wedge rOut[i] = nil \wedge \min[i+1] = \min[i] \right) \right) \right)$	
Flow Control 1		消费来自于左侧输入端口的 token 及底端控制端口的整数	设左侧端口所连接 relation 为 rIn，底端端口所连接 relation 为 rCtrl，右侧端口所连接 relation 为 rOut，创建变量 s 辅助编码，则：	


		<p>值 token，并将输入 token 的路线确定为控制 token 所指定的输出管道，而其它管道在此次激励中不生成 token.</p>	$\bigvee_{j=1}^{inPort.width} \left( \left( rCtrl[i] = j \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = j \wedge rOut_j[i] = rIn[i] \wedge \bigwedge_{h=1}^{j-1} rOut_h[i] = nil \wedge \bigwedge_{h=j+1}^{inPort.width} rOut_h[i] = nil \right) \vee \left( (rCtrl[i] = value \wedge s[i] = rCtrl[i]) \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge (s[i] < 0 \vee s[i] \geq width) \wedge \bigwedge_{h=1}^{inPort.width} rOut_h[i] = nil \right)$ <p><math>s[0] = 0</math></p> <p>注：inPort.width 可通过看 Switch actor 的 input 连接几条 relation 获知.</p>
		<p>是 Switch 角色的一个变体，其中输出仅限为两个，且控制 token 是布尔值，因此作用为：消费来自于左侧输入端口的 token 及底端控制端口的布尔值 token，并将输入 token 的路线确定为控制 token 所指定的输出管道，而另一个输出管道在此次激励中不生成 token.</p>	<p>设左侧端口所连接 relation 为 rIn，底端端口所连接 relation 为 rCtrl，右侧 trueOutput 端口所连接 relation 为 rtOut，且右侧 falseOutput 端口所连接 relation 为 rfOut，创建变量 s 辅助编码，则：</p> $\left( \left( rCtrl[i] = true \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = true \wedge \bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] \neq nil \wedge rtOut_j[i] = rIn_j[i]) \vee (rIn_j[i] = nil \wedge rtOut_j[i] = nil) \right) \wedge \bigwedge_{j=1}^{falsePort.width} rfOut_j[i] = nil \right) \vee \left( \left( rCtrl[i] = false \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = false \wedge \bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] \neq nil \wedge rfOut_j[i] = rIn_j[i]) \vee (rIn_j[i] = nil \wedge rfOut_j[i] = nil) \right) \wedge \bigwedge_{j=1}^{truePort.width} rtOut_j[i] = nil \right)$ <p><math>s[0] = false</math></p> <p>rtOut和rfOut可以不连接</p> <p>注：inPort.width 可通过看 BooleanSwitch actor 的 input 连接几条 relation 获知.</p>
		<p>消费由底端控制端口的整数值 token 所指定的管道中的单个 token，并将之发送给输出，其它输入端口处不消费 token.</p>	<p>设左侧端口所连接 relation 依次为 rIn1, rIn2, rIn3，底端端口所连接 relation 为 rCtrl，右侧端口所连接 relation 为 rOut，创建变量 s 辅助编码，则：</p> $\bigvee_{j=1}^{inPort.width} \left( \left( rCtrl[i] = j \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = j \wedge \left( (rIn_j[i] \neq nil \wedge rOut[i] = rIn_j[i]) \vee (rIn_j[i] = nil \wedge rOut[i] = nil) \right) \right) \vee \left( (rCtrl[i] = value \wedge s[i] = rCtrl[i]) \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge (s[i] < 1 \vee s[i] \geq inPort.width) \wedge rOut[i] = nil$ <p><math>s[0] = 0</math></p> <p>注：inPort.width 可通过看 Select actor 的 input 连接几条 relation 获知.</p>
		<p>将底端 select 端口指定的输入管道中的 token，复制到输出端口处，而其它输入管道中的 token 全部舍弃.</p>	<p>设左侧端口所连接 relation 为 rIn，底端端口所连接 relation 为 rSec，右侧端口所连接 relation 为 rOut，创建变量 s 辅助编码，则：</p> $\bigvee_{j=1}^{inPort.width} \left( \left( rSec[i] = j \vee (rSec[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = 0 \wedge \left( (rIn_j[i] \neq nil \wedge rOut[i] = rIn_j[i]) \vee (rIn_j[i] = nil \wedge rOut[i] = nil) \right) \right) \vee \left( (rCtrl[i] = value \wedge s[i] = rCtrl[i]) \vee (rCtrl[i] = nil \wedge s[i] = s[i-1]) \right) \wedge (s[i] < 1 \vee s[i] \geq inPort.width) \wedge rOut[i] = nil$ <p><math>s[0] = -1</math></p> <p>注：inPort.width 可通过看 Multiplexor actor 的 input 连接几条 relation 获知.</p>
		<p>输出底端控制端口指定</p>	<p>设左侧 trueInput, falseInput 端口所连接 relation 分别为 rtIn, rIn，底端端口所连接 relation 为 rSec，右侧端口所连接 relation 为 rOut，则：</p>










		<p>的输入管道中的 token, 而另一个输入管道中的 token 全部舍弃.</p>	$\left( \begin{aligned} & \left( rSec[i] = true \vee (rSec[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = "true" \\ & \wedge \left( (rIn[i]! = nil \wedge rOut[i] = rIn[i]) \vee (rIn[i] = nil \wedge rOut[i] = nil) \right) \end{aligned} \right) \\ \vee \left( \begin{aligned} & \left( rSec[i] = false \vee (rSec[i] = nil \wedge s[i] = s[i-1]) \right) \wedge s[i] = "false" \\ & \wedge \left( (rIn[i]! = nil \wedge rOut[i] = rIn[i]) \vee (rIn[i] = nil \wedge rOut[i] = nil) \right) \end{aligned} \right) \\ \vee (rSec[i] = nil \wedge s[i] = s[i-1] \wedge s[i] = "nil" \wedge rOut[i] = nil) \\ \text{这里的s是String类型, } s[0] = "nil"$
		<p>输出一个 record, 且对于每个输入端口, 该 record 会包含有一个与该输入端口同名的域.</p>	<p>令左侧输入端口名称依次为 fieldName<sub>1</sub>, fieldName<sub>2</sub>, fieldName<sub>3</sub>, ... fieldName<sub>n</sub>, 且令这些端口所连接 relation 依次为: rIn_fieldName<sub>1</sub>, rIn_fieldName<sub>2</sub>, rIn_fieldName<sub>3</sub>, ... rIn_fieldName<sub>n</sub>, 另令右侧端口所连接 relation 为 rOut, 则</p> $\left( \begin{aligned} & \left( \bigvee_{j=1}^n (rIn\_fieldName_j[i] = nil) \right) \wedge (rOut[i] = nil) \\ & \vee \left( \neg \left( \bigvee_{j=1}^n (rIn\_fieldName_j[i] = nil) \right) \right) \\ & \wedge \left( rOut[i] = (valueRecord(rIn\_fieldName_1[i], rIn\_fieldName_2[i], rIn\_fieldName_3[i], \dots, rIn\_fieldName_n[i])) \right) \end{aligned} \right)$
		<p>从输入的 record 中提取域, 且输出端口的名字必须与域名相匹配.</p>	<p>令左侧输入端口所连接 relation 为 rIn, 令右侧输出端口名称依次为 fieldName<sub>1</sub>, fieldName<sub>2</sub>, fieldName<sub>3</sub>, ... fieldName<sub>n</sub>, 且令这些端口所连接 relation 依次为: rOut_fieldName<sub>1</sub>, rOut_fieldName<sub>2</sub>, rOut_fieldName<sub>3</sub>, ... rOut_fieldName<sub>n</sub>, 则:</p> $\bigwedge_{j=1}^n rOut\_fieldName_j[i] = fieldName_j(rIn[i])$
		<p>增加或修改所输入 record 中的 field, 并输出更新后的 record.</p>	<p>令左侧内置输入端口所连接 relation 为 rIn, 其它输入端口名称依次为 fieldName<sub>1</sub>, fieldName<sub>2</sub>, fieldName<sub>3</sub>, ... fieldName<sub>n</sub>, 令这些端口所连接 relation 依次为: rIn_fieldName<sub>1</sub>, rIn_fieldName<sub>2</sub>, rIn_fieldName<sub>3</sub>, ... rIn_fieldName<sub>n</sub>, 另令右侧输出端口所连接 relation 为 rOut, 则:</p> $\left( \begin{aligned} & \neg \left( \bigvee_{j=1}^n (rIn\_fieldName_j[i] = nil) \right) \\ & \wedge \left( (is - valueRecord(rOut[i]) \wedge \left( \bigwedge_{j=1}^n (fieldName_j(rOut[i]) = rIn\_fieldName_j[i]) \right) \wedge \left( \bigwedge_{j \in \{n, n-1, n-2, \dots, 1\}} (fieldName_j(rOut[i]) = fieldName_j(rIn[i])) \right) \right) \right) \\ & \vee \left( \bigvee_{j=1}^n (rIn\_fieldName_j[i] = nil) \wedge (rOut[i] = nil) \right) \end{aligned} \right)$
Conversions		<p>将一个 double 类型的数值转换为 int 型.</p>	<p>设左、右两侧端口所连接的 relation 分别为 rIn, rOut, Round 中参数 functionName, 则</p> $\left( \begin{aligned} & rIn[i]! = nil \wedge rIn[i]! = to\_int(rIn[i]) \wedge \\ & \left( \begin{aligned} & \left( \begin{aligned} & (functionName == "ceil" \wedge rOut_i = to\_int(rIn[i]) + 1) \\ & \vee (functionName == "floor" \wedge rOut_i = to\_int(rIn[i])) \end{aligned} \right) \\ & \vee \left( \begin{aligned} & (functionName == "round" \wedge \left( (rIn[i] - to\_int(rIn[i]) \geq 0.5 \wedge rOut[i] = to\_int(rIn[i]) + 1) \right) \\ & \vee \left( (rIn[i] - to\_int(rIn[i]) < 0.5 \wedge rOut[i] = to\_int(rIn[i])) \right) \end{aligned} \right) \\ & \vee (functionName == "truncate" \wedge ((rIn[i] \geq 0 \wedge rOut[i] = to\_int(rIn[i])) \vee (rIn[i] < 0 \wedge rOut[i] = to\_int(rIn[i]) + 1))) \end{aligned} \right) \end{aligned} \right) \\ & \vee (rIn[i] = nil \wedge rOut[i] = nil) \\ & \vee (rIn[i]! = nil \wedge rIn[i] = to\_int(rIn[i]) \wedge rOut[i] = rIn[i]) \end{aligned} \right)$
		<p>将布尔值转化为两个任意的值.</p>	<p>设左、右两侧端口所连接的 relation 分别为 rIn, rOut, 则</p> $\left( \begin{aligned} & rIn[i]! = nil \wedge \left( \begin{aligned} & (rIn[i] = true \wedge rOut[i] = trueValue) \\ & \vee (rIn[i] = false \wedge rOut[i] = falseValue) \end{aligned} \right) \\ & \vee (rIn[i] = nil \wedge rOut[i] = nil) \end{aligned} \right)$ <p>注: 默认情况下, trueValue 为 1, falseValue 为 0, 如果更改的话, 可从模型的 xml 文件中 BooleanToAnything actor 的 property 获得.</p>
Random Number Generators		<p>随机地输出布尔值 T 和 F.</p>	<p>设右侧端口所连接 relation 为 rOut, 则</p> $rOut[i] = true \vee rOut[i] = false$
Array		<p>判定数组中是否包含一</p>	<p>情形一: 设左侧 element 端口未连接 relation, array 端口所连接 relation 为</p>






Library		个指定元素.	<p>rIn, 右侧端口所连接 relation 为 rOut, ArrayContains 中参数 elementParameter, 则</p> $\left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge \left( \left( rIn[i].contains(elementParameter) = true \wedge rOut[i] = true \right) \vee \left( rIn[i].contains(elementParameter) = false \wedge rOut[i] = false \right) \right) \right)$ <p>情形二: 设左侧 element 端口连接 relation 为 reIn, array 端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, ArrayContains 中参数 elementParameter, 创建变量 ST_ArrayContains, element 辅助编码, 则</p> $\left( ST\_ArrayContains[i-1] = 0 \wedge \left( \left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge \left( \left( reIn[i] = nil \wedge element[i] = elementParameter \right) \vee \left( reIn[i] = value \wedge element[i] = reIn[i] \right) \right) \right) \right) \right) \wedge ST\_ArrayContains[i] = 1$ $\vee \left( ST\_ArrayContains[i-1] = 1 \wedge \left( \left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge \left( \left( reIn[i] = nil \wedge element[i] = element[i-1] \right) \vee \left( reIn[i] = value \wedge element[i] = reIn[i] \right) \right) \right) \right) \right) \wedge ST\_ArrayContains[i] = 1$
		提取出数组中的一个元素.	<p>情形一: 设左侧 element 端口未连接 relation, array 端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, ArrayElement 中参数 indexParameter, 则</p> $\left( rIn[i] = value \wedge rOut[i] = select(rIn[i], indexParameter) \right) \vee \left( rIn[i] = nil \wedge rOut[i] = nil \right)$ <p>情形二: 设左侧 element 端口连接 relation 为 rIndex, array 端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, 创建变量 ST_ArrayElement, index 辅助编码, 则</p> $\left( ST\_ArrayElement[i-1] = 0 \wedge \left( \left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge rOut[i] = select(rIn[i], index[i]) \right) \vee \left( rIndex[i] = nil \wedge index[i] = indexParameter \right) \vee \left( rIndex[i] = value \wedge index[i] = rIndex[i] \right) \right) \right) \wedge ST\_ArrayElement[i] = 1$ $\vee \left( ST\_ArrayElement[i-1] = 1 \wedge \left( \left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge rOut[i] = select(rIn[i], index[i]) \right) \vee \left( rIndex[i] = nil \wedge index[i] = index[i-1] \right) \vee \left( rIndex[i] = value \wedge index[i] = rIndex[i] \right) \right) \right) \wedge ST\_ArrayElement[i] = 1$
		输出所输入数组的长度.	<p>设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, 则:</p> $\left( rIn[i] = nil \wedge rOut[i] = nil \right) \vee \left( rIn[i] = value \wedge rOut[i] = rIn[i].length \right)$
		查找数组中的最大元素.	<p>设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, rIndex, 则</p> $\left( rIn[i] = nil \wedge rOut[i] = nil \wedge rIndex[i] = nil \right) \vee \left( rIn[i] = value \wedge rOut[i] = rIn[i].max.data \wedge rIndex[i] = rIn[i].max.index \right)$
		查找数组中的最小元素.	<p>设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, rIndex, 则</p> $\left( rIn[i] = nil \wedge rOut[i] = nil \wedge rIndex[i] = nil \right) \vee \left( rIn[i] = value \wedge rOut[i] = rIn[i].min.data \wedge rIndex[i] = rIn[i].min.index \right)$

		将数组中的元素输出到输出端口的每个 channel 上.	设左侧端口所连接 relation 为 $rIn$ , 右侧端口所连接 relation 为 $rOut$ , 则 $\left( rIn[i] = nil \wedge \bigwedge_{j=1}^{outPort.width} (rOut_j[i] = nil) \right) \vee \left( rIn[i] = value \wedge \left( \bigwedge_{j=1}^{outPort.width} (rOut_j[i] = select(rIn[i], j-1)) \right) \right)$
		改变数组中的一个元素, 并将改变后的数组输出.	设左侧端口所连接 relation 为 $rIn$ , 底端端口所连接 relation 为 $rIndex$ 和 $rValue$ , 右侧端口所连接 relation 为 $rOut$ , $ArrayUpdate$ 中参数 $valueParameter$ , $indexParameter$ , 则 $\left( \begin{aligned} &rIn[i] = vlaue \\ &\wedge ((index[i] = index[i-1] \wedge rIndex[i] = nil) \vee (rIndex[i] = value \wedge index[i] = rIndex[i])) \\ &\wedge ((rValue[i] = nil \wedge value[i] = value[i-1]) \vee (rValue[i] = value \wedge value[i] = rValue[i])) \\ &\wedge \left( \begin{aligned} &(0 \leq index[i] < rIn[i].length \wedge rOut[i] = store(rIn[i], Index[i], value[i])) \\ &\vee ((index[i] < 0 \vee index[i] \geq rIn[i].length) \wedge rOut[i] = rIn[i]) \end{aligned} \right) \end{aligned} \right)$ $\vee (rIn[i] = nil \wedge rOut[i] = nil)$ $value[0] = valueParameter, index[0] = indexParameter$
		根据输入端口各个 channel 上的元素构建一个数组.	设左侧端口所连接 relation 依次为 $rIn1, rIn2, rIn3$ , 右侧端口所连接 relation 为 $rOut$ , 则 $\left( \bigvee_{j=1}^n (rIn_j[i] = nil) \wedge rOut[i] = nil \right) \vee \left( \bigwedge_{j=1}^n (rIn_j[i] = value) \wedge rOut[i] = store \left( \underbrace{store \left( \cdots store \left( store \left( store(rOut[i], 0, rIn_1[i]), 1, rIn_2[i]), 2, rIn_3[i] \right) \cdots \right), (n-1), rIn_n[i] \right)}_{input.Width \text{ 个 store 函数嵌套}} \right) \right)$
Logic Library		输出与输入相反的值.	设左、右两侧端口所连接 relation 分别为 $rIn, rOut$ , 则 $(rIn[i] = value \wedge rOut[i] = \neg rIn_i) \vee (rIn[i] = nil \wedge rOut[i] = nil)$
		输出左侧输入值的合取值 (and)、析取值 (or)、合取值的否定 (nand)、析取值的否定 (nor)、异或运算值 (xor)、同或运算值 (xnor).	①若 logic 为“and”: 设左侧端口所连接 relation 为 $rIn$ , 右侧端口所连接 relation 为 $rOut$ , 则 $\left( \begin{aligned} &\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \left( \begin{aligned} &\left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = true \vee rIn_j[i] = nil) \wedge rOut[i] = true \right) \\ &\vee \left( \bigvee_{j=1}^{inPort.width} (rIn_j[i] = false) \wedge rOut[i] = false \right) \end{aligned} \right) \end{aligned} \right) \vee \left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil \right)$ ②若 logic 为“or”: 设左侧端口所连接 relation 为 $rIn$ , 右侧端口所连接 relation 为 $rOut$ , 则 $\left( \begin{aligned} &\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \left( \begin{aligned} &\left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = false \vee rIn_j[i] = nil) \wedge rOut[i] = false \right) \\ &\vee \left( \bigvee_{j=1}^{inPort.width} (rIn_j[i] = true) \wedge rOut[i] = true \right) \end{aligned} \right) \end{aligned} \right) \vee \left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil \right)$ ③若 logic 为“nand”: 设左侧端口所连接 relation 为 $rIn$ , 右侧端口所连接 relation 为 $rOut$ , 则 $\left( \begin{aligned} &\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \left( \begin{aligned} &\left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = true \vee rIn_j[i] = nil) \wedge rOut[i] = false \right) \\ &\vee \left( \bigvee_{j=1}^{inPort.width} (rIn_j[i] = false) \wedge rOut[i] = true \right) \end{aligned} \right) \end{aligned} \right) \vee \left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil \right)$

		<p>④若 logic 为“nor”: 设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, 则</p> $\sqrt{\left(\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \left(\bigwedge_{j=1}^{inPort.width} (rIn_j[i] = false \vee rIn_j[i] = nil) \wedge rOut[i] = true\right)\right) \vee \left(\bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil\right)}$ <p>⑤若 logic 为“xor”: 设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, 创建数组 trueNumber, falseNumber, numberOfTrue, numberOfFalse 辅助编码, 则:</p> $\sqrt{\left(\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \bigwedge_{j=1}^{inPort.width} \left(\left(rIn_j[i] = true \wedge trueNumber_j[i] = 1 \wedge falseNumber_j[i] = 0\right) \vee \left(rIn_j[i] = false \vee rIn_j[i] = nil\right) \wedge trueNumber_j[i] = 0 \wedge falseNumber_j[i] = 1\right)\right) \wedge \left(numberOfTrue[i] = \sum_{j=1}^{inPort.width} trueNumber_j[i] \wedge \left(numberOfFalse[i] = \sum_{j=1}^{inPort.width} falseNumber_j[i] \wedge \left(\left(numberOfTrue[i] = 0 \wedge numberOfFalse[i] \neq 0 \wedge rOut[i] = false\right) \vee \left(numberOfTrue[i] \neq 0 \wedge \left(\left(numberOfTrue[i] \% 2 = 0 \wedge rOut[i] = false\right) \vee \left(numberOfTrue[i] \% 2 = 1 \wedge rOut[i] = true\right)\right)\right)\right)\right) \wedge \left(\bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil\right)}$ <p>⑥若 logic 为“xnor”: 设左侧端口所连接 relation 为 rIn, 右侧端口所连接 relation 为 rOut, 创建数组 trueNumber, falseNumber, numberOfTrue, numberOfFalse 辅助编码, 则:</p> $\sqrt{\left(\bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \bigwedge_{j=1}^{inPort.width} \left(\left(\left(rIn_j[i] = true \vee rIn_j[i] = nil\right) \wedge trueNumber_j[i] = 1 \wedge falseNumber_j[i] = 0\right) \vee \left(rIn_j[i] = false \wedge trueNumber_j[i] = 0 \wedge falseNumber_j[i] = 1\right)\right)\right) \wedge \left(numberOfTrue[i] = \sum_{j=1}^{inPort.width} trueNumber_j[i] \wedge \left(numberOfFalse[i] = \sum_{j=1}^{inPort.width} falseNumber_j[i] \wedge \left(\left(numberOfTrue[i] = 0 \wedge numberOfFalse[i] \neq 0 \wedge rOut[i] = true\right) \vee \left(numberOfTrue[i] \neq 0 \wedge \left(\left(numberOfTrue[i] \% 2 = 0 \wedge rOut[i] = true\right) \vee \left(numberOfTrue[i] \% 2 = 1 \wedge rOut[i] = false\right)\right)\right)\right)\right) \wedge \left(\bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge rOut[i] = nil\right)}$ <p>注: inPort.width 可通过看 LogicGate actor 的 input 连接几条 relation 获知.</p>
		<p>比较所输入的两个 double 类型的数值的大小, 并输出布尔值, 可能的比较符号有: &gt;, &gt;=, &lt;, &lt;= 及 ==.</p> <p>设左侧端口所连接的 relation 分别为 rIn1, rIn2, 右侧端口所连接 relation 为 rOut, Comparator 内部参数 tolerance, 则</p> <p>①若比较符号是 “==”, 则</p> $\left((rIn1[i] = nil \vee rIn2[i] = nil) \wedge rOut[i] = nil\right) \vee \left(\left(rIn1[i] = value \wedge rIn2[i] = value\right) \wedge \left(\left(\left(abs(rIn1[i] - rIn2[i])\right) \leq tolerance\right) \wedge rOut[i] = true\right) \vee \left(\left(abs(rIn1[i] - rIn2[i])\right) > tolerance\right) \wedge rOut[i] = false\right)\right)$ <p>②若比较符号是 “&gt;=”, 则</p>

			$\left( (rIn[i] = nil \vee rIn2[i] = nil) \wedge rOut[i] = nil \right)$ $\vee \left( \left( rIn[i] = value \wedge rIn2[i] = value \right) \wedge \left( \left( (rIn[i] - rIn2[i] + tolerance) \geq 0 \right) \wedge rOut[i] = true \right) \vee \left( \left( (rIn[i] - rIn2[i] + tolerance) < 0 \right) \wedge rOut[i] = false \right) \right) \right)$ <p>③若比较符号是 “&lt;=”，则</p> $\left( (rIn[i] = nil \vee rIn2[i] = nil) \wedge rOut[i] = nil \right)$ $\vee \left( \left( rIn[i] = value \wedge rIn2[i] = value \right) \wedge \left( \left( (rIn2[i] - rIn[i] + tolerance) \geq 0 \right) \wedge rOut[i] = true \right) \vee \left( \left( (rIn2[i] - rIn[i] + tolerance) < 0 \right) \wedge rOut[i] = false \right) \right) \right)$ <p>④若比较符号是 “&gt;”，则</p> $\left( (rIn[i] = nil \vee rIn2[i] = nil) \wedge rOut[i] = nil \right)$ $\vee \left( \left( rIn[i] = value \wedge rIn2[i] = value \right) \wedge \left( \left( (rIn[i] - rIn2[i] + tolerance) > 0 \right) \wedge rOut[i] = true \right) \vee \left( \left( (rIn[i] - rIn2[i] + tolerance) \leq 0 \right) \wedge rOut[i] = false \right) \right) \right)$ <p>⑤若比较符号是 “&lt;”，则</p> $\left( (rIn[i] = nil \vee rIn2[i] = nil) \wedge rOut[i] = nil \right)$ $\vee \left( \left( rIn[i] = value \wedge rIn2[i] = value \right) \wedge \left( \left( (rIn2[i] - rIn[i] + tolerance) > 0 \right) \wedge rOut[i] = true \right) \vee \left( \left( (rIn2[i] - rIn[i] + tolerance) \leq 0 \right) \wedge rOut[i] = false \right) \right) \right)$
		比较所输入的任意数量的任意类型的 token 是否相等，如果相等，输出 true；否则，输出 false.	<p>设左侧端口所连接的 relation 为 rIn,右侧端口所连接 relation 为 rOut，创建数组 s 辅助编码，则</p> $\left( \bigwedge_{j=1}^{inPort.width} (rIn_j[i] = nil) \wedge \neg (s_1[i] = s_2[i] = \dots = s_{inPort.width}[i]) \wedge rOut[i] = nil \right)$ $\vee \left( \bigvee_{j=1}^{inPort.width} (rIn_j[i] = value) \wedge \bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] = value \wedge s_j[i] = rIn_j[i]) \vee (rIn_j[i] = nil \wedge s_j[i] = s_{j-1}[i]) \right) \wedge \left( (s_1[i] = s_2[i] = \dots = s_{inPort.width}[i]) \wedge rOut[i] = true \right) \vee \left( \neg (s_1[i] = s_2[i] = \dots = s_{inPort.width}[i]) \wedge rOut[i] = false \right) \right)$ <p>注：inPort.width 可通过看 Equals actor 的 input 连接几条 relation 获知.</p>
		若输入 present，则输出 true；否则输出 false.	<p>设左侧多端口所连接的 relation 为 rIn，右侧多端口所连接的 relation 为 rOut，则</p> $\bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] = value \wedge rOut_j[i] = true) \vee (rIn_j[i] = nil \wedge rOut_j[i] = false) \right)$ <p>注：inPort.width 可通过看 IsPresent actor 的 input 连接几条 relation 获知.</p>
		若输入为布尔值 true，则输出 true；否则不输出 token.	<p>设左侧多端口所连接的 relation 依次为 rIn1, rIn2, rIn3，右侧多端口所连接的 relation 依次为 rOut1, rOut2, rOut3,则</p> $\bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] = true \wedge rOut_j[i] = true) \vee \left( (rIn_j[i] = false \vee rIn_j[i] = nil) \wedge rOut_j[i] = nil \right) \right)$ <p>注：inPort.width 可通过看 TrueGate actor 的 input 连接几条 relation 获知.</p>
DomainSpecific -- SR		提供一个 tick 的延迟：每次激励时，输出上个 tick 时输入端口处所	<p>设左、右两侧端口所连接的 relation 分别是 rIn 和 rOut，创建变量 lastInput 辅助编码，则</p>

		读入的任何值；第一个 tick 时，输出值由参数 <code>initialValue</code> 给出；若未给定任何值，则输出 <code>absent</code> .	$\left( (rOut[i] = lastInput[i - 1]) \right) \wedge \left( (rIn[i] = value) \wedge (lastinput[i] = rIn[i]) \right) \vee \left( (rIn[i] = nil) \wedge (lastinput[i] = nil) \right)$
		基于底部信号来过滤当前输入信号：若底部控制端口输入为 <code>present</code> 且值为 <code>true</code> ，则输出为左侧的输入数据；而如果底部控制端口输入为 <code>absent</code> ，或者为 <code>false</code> ，亦或者底部控制端口输入为 <code>present</code> 且值为 <code>true</code> 但左侧输入端口为 <code>absent</code> ，则输出为 <code>absent</code> .	<p>设左侧、底部和右侧端口所对应的 relation 分别是 <code>rIn</code>, <code>rCtrl</code> 和 <code>rOut</code>, 则：</p> $(rCtrl[i] = true \wedge rOut[i] = rIn[i]) \vee ((rCtrl[i] = false \vee rCtrl[i] = nil) \wedge rOut[i] = nil)$
		将两个信号有优先级地进行合并：若优先级较高的左侧端口输入为 <code>present</code> ，则输出为该输入数据；如果左侧输入为 <code>absent</code> ，则输出为底部的输入（无论其是否为 <code>absent</code> ）。	<p>设左侧、底部和右侧端口所对应的 relation 分别是 <code>rIn</code>, <code>rBot</code> 和 <code>rOut</code>, 则：</p> $(rIn[i] = value \wedge rOut[i] = rIn[i]) \vee \left( rIn[i] = nil \wedge \left( (rBot[i] = nil \wedge rOut[i] = nil) \vee (rBot[i] = value \wedge rOut[i] = rBot[i]) \right) \right)$
		输出上次接收到的非 <code>absent</code> 输入。若当前输入是 <code>absent</code> ，则输出也是 <code>absent</code> ；第一次输入是 <code>present</code> 时，输出由参数 <code>initialValue</code> 给出（该参数默认为	<p>设左侧和右侧端口所对应的 relation 分别是 <code>rIn</code> 和 <code>rOut</code>，创建变量 <code>s</code>，<code>index</code> 辅助编码，则</p> $(rIn[i] = nil \wedge rOut[i] = nil \wedge ST[i - 1] = ST[i] \wedge index[i] = index[i - 1]) \vee \left( rIn[i] = value \wedge ST[i] = ST[i - 1] + 1 \wedge s_{index[i]} = rIn[i] \wedge \left( (ST[i - 1] = 0 \wedge rOut[i] = initialValue[i] \wedge index[i] = index[i - 1]) \vee (ST[i] \geq 1 \wedge rOut[i] = s_{index[i - 1]} \wedge index[i] = index[i - 1] + 1) \right) \right)$

		absent).	
		输出最新接收到的非absent输入, 若没有接收到输入, 则输出为absent.	<p>设左侧和右侧端口所对应的 relation 分别是 rIn 和 rOut, 创建变量 s 辅助编码, 则</p> $\left( rIn[i] = value \wedge rOut[i] = rIn[i] \wedge s[i] = rIn[i] \right) \vee \left( rIn[i] = nil \wedge s[i] = s[i-1] \wedge \left( rOut[i] = s[i] \wedge s[i] = value \vee (s[i] = nil \wedge rOut[i] = nil) \right) \right)$ $s[0] = nil$
		输出总是absent.	设右侧端口所对应的 relation 是 rOut, 则 $rOut[i] = nil$
		若输入为present, 则输出true; 否则输出false.	<p>设左、右侧端口所对应 relation 分别是 rIn1, rIn2 和 rOut1, rOut2, 则:</p> $\bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] = value \wedge rOut_j[i] = true) \vee (rIn_j[i] = nil \wedge rOut_j[i] = false) \right)$ <p>注: inPort.width 可通过看 IsPresent actor 的 input 连接几条 relation 获知.</p>
		若输入为present且值为true, 则输出true; 否则输出absent.	<p>设左、右侧端口所对应的 relation 分别是 rIn1, rIn2 和 rOut1, rOut2, 则:</p> $\bigwedge_{j=1}^{inPort.width} \left( (rIn_j[i] = true \wedge rOut_j[i] = true) \vee (rIn_j[i] = nil \vee rIn_j[i] = false) \wedge rOut_j[i] = nil \right)$ <p>注: inPort.width 可通过看 TrueGate actor 的 input 连接几条 relation 获知.</p>
Sink		<p>设定由 variableName 参数所指定的变量的取值, 且有两种情况:</p> <p>情况一 (delayed=false): 在对该 actor 进行 fire 时, 设定变量的取值为输入值, 然后输出与输入端口相同的 token;</p> <p>情况二 (delayed=true): 输出所访问变量的当前值, 且所访问变量在当前迭代运行结束时才进行设定。</p>	<p>情况一: 令输入端口所连接 relation 为 rIn, 令 variableName 为 x, delayed=false, 另令输出端口所连接 relation 为 rOut, 则:</p> $(is\_value(rIn[i]) \wedge x[i] = rIn[i] \wedge rOut[i] = rIn[i]) \vee (rIn[i] = absent \wedge x[i] = x[i-1] \wedge rOut[i] = absent));$ <p>情况二: 令输入端口所连接 relation 为 rIn, 令 variableName 为 x, delayed=true, 另令输出端口所连接 relation 为 rOut, 则:</p> $(is\_value(rIn[i]) \wedge x[i] = rIn[i] \wedge rOut[i] = x[i-1]) \vee (rIn[i] = absent \wedge x[i] = x[i-1] \wedge rOut[i] = x[i-1]))$