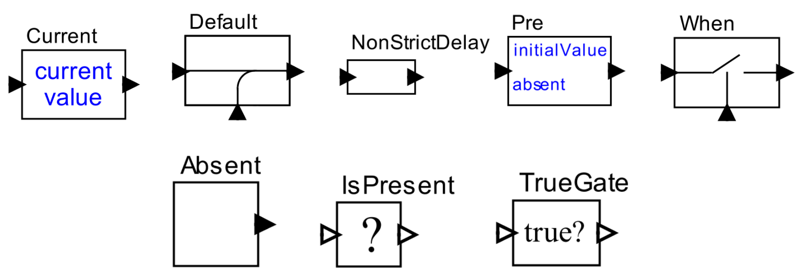
Bounded Model Checking based Encoding of Commonly used NonFSMActors in Ptolemy II Synchronous Reactive Models

No Author Given

No Institute Given

A. SynchronousReactive Library

The Ptolemy II library `DomainSpecific`→`SynchronousReactive` offers the following actors that are mainly be used in SR domain. And we will discuss their usage and encoding in the order shown in the following figure.



- **Current** outputs the most recent non-absent input, that is, if the input at the current tick is non-absent, this actor outputs the input token; however, if the input at the current tick is absent, then this actor outputs the last non-absent input [1]. Suppose this actor has not received any input token up to the current tick, then it outputs `absent`. Let the variable s record the most recent non-absent input (and initially, we set $s[0] = \texttt{absent}$), and let the variables rIn and $rOut$ denote the input and the output of this actor, respectively, then we can encode the behavior of this actor as:

$$T_{Current}[i] \triangleq \Bigg(\Big(\textit{is-value}(rIn[i]) \wedge s[i] = rIn[i]\Big) \vee \Big(rIn[i] = \texttt{absent} \wedge s[i] = s[i-1]\Big)\Bigg) \wedge rOut[i] = s[i].$$

- **Default** merges the two inputs $preferred$ and $alternate$ with priority: if the left $preferred$ input port has a token, this actor produces the value of this token; while if the left $preferred$ input is absent, then this actor outputs the bottom *alternate* input whether it is absent or not [1]. In addition, if the bottom *alternate* input is unknown, this actor can produce an output as long as the left $preferred$ input is non-absent. Let the variables rIn, $rBot$ and $rOut$ denote the $preferred$ input, the *alternate* input and the *output* of this actor respectively, and then the behavior of this actor can be encoded as:

$$T_{Default}[i] \triangleq \Big[\textit{is-value}(rIn[i]) \land rOut[i] = rIn[i] \Big] \lor \Big[rIn[i] = \texttt{absent} \land \Big(\big(rBot[i] = \texttt{absent} \land rOut[i] = \texttt{absent} \big) \lor \big(\textit{is-value}(rBot[i]) \land rOut[i] = rBot[i] \big) \Big) \Big]$$

• **NonStrictDelay** outputs the input at last tick whether it is absent or not. At the first tick, if the $initialValue$ parameter is set, this actor outputs the value of the $initialValue$ parameter; otherwise, this actor outputs `absent` [1]. Let the variable *lastInput* record the last received input, and if the $initialValue$ parameter is given, we set $lastInput[0] = initialValue$, or $lastInput[0] = absent$ otherwise. Suppose the variables rIn and $rOut$ are introduced to represent the input and the output of this actor respectively, we can get that

$$T_{NonStrictDelay}[i] \triangleq \Big(rOut[i] = lastInput[i-1] \Big) \land \Big(\big(\textit{is-value}(rIn[i]) \land lastInput[i] = rIn[i] \big) \lor \big(rIn[i] = \texttt{absent} \land lastInput[i] = \texttt{absent} \big) \Big)$$

• **Pre** produces the last non-absent input, if the current input is non-absent; while if the current input is absent, then this actor outputs `absent` [1]. At the first tick, when the input is non-absent, if the $initialValue$ is given, this actor outputs the value of $initialValue$; otherwise, this actor outputs `absent`. Let the variable *lastNonAbsentInput* record the last non-absent input, and if the $initialValue$ is given, we set $lastNonAbsentInput[0] = initialValue$; otherwise, we set $lastNonAbsentInput[0] = \texttt{absent}$. Suppose the variables rIn and $rOut$ represent the input and the output of this actor respectively, we can get that

$$T_{Pre}[i] \triangleq \Big(\textit{is-value}(rIn[i]) \land rOut[i] = lastNonAbsentInput[i-1] \land lastNonAbsentInput[i] = rIn[i] \Big) \lor \Big(rIn[i] = \texttt{absent} \land rOut[i] =$$

$$\texttt{absent} \wedge lastNonAbsentInput[i] = lastNonAbsentInput[i-1]\Big)$$

- **When** filters the left input based on the bottom control input: if the bottom control input is true, this actor outputs the left input whether it is absent or not; if the bottom control input is false or absent, this actor outputs `absent` [1]. Let the variables rIn, $rCtrl$ and $rOut$ denote the left input, the bottom input, and the output of this actor, respectively, we can encode the behavior of this actor as follows:

$$T_{When}[i] \triangleq \Big(rCtrl[i] = true \wedge rOut[i] = rIn[i]\Big)$$
$$\vee \Big(\big(rCtrl[i] = false \vee rCtrl[i] = \texttt{absent}\big) \wedge rOut[i] = \texttt{absent}\Big)$$

- **Absent** outputs absent and the output type is *int* by default [1], therefore, when connecting this actor to an input port of other type, we need to set the output type of the Absent actor be the same as the connected input port firstly. Let the variable $rOut$ be the output of this actor, then we can encode it as:

$$T_{Absent} \triangleq rOut[i] = \texttt{absent}.$$

- **IsPresent** checks whether each input channel has a token: if one input channel has a token, then its corresponding output channel outputs true; otherwise, this output channel outputs false [1]. Let the variables rIn_1, rIn_2, ..., rIn_m denote the relations connected to every input channel, and the variables $rOut_1$, $rOut_2$, ..., $rOut_m$ denote the relations connected to every output channel, an then this actor can be encoded as:

$$T_{IsPresent} \triangleq \bigwedge_{j=1}^{m} \Bigg(\Big(\textit{is-value}(rIn_j[i]) \wedge rOut_j[i] = true\Big) \vee \Big(rIn_j[i] = \texttt{absent} \wedge$$
$$rOut_j[i] = false\Big)\Bigg)$$

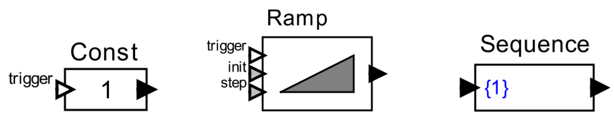
- **TrueGate** checks whether each input channel has a token and its value is true: if one input channel has a token and its value is true, then its corresponding output channel outputs true; otherwise, this output channel outputs `absent` [1]. Let the variables $rIn_1, rIn_2, ..., rIn_m$ denote the relation connected to every input channel, and the variables $rOut_1, rOut_2, ..., rOut_m$ denote the relation connected to every output channel, an then this actor can be encoded as:

$$T_{TrueGate} \triangleq \bigwedge_{j=1}^{m} \Bigg[\Big(rIn_j[i] = true \wedge rOut_j[i] = true\Big) \vee \Big(\big(rIn_j[i] = false \vee$$

$$rIn_j[i] = \texttt{absent}\Big) \wedge rOut_j[i] = \texttt{absent}\Big)\Big]$$

B. Sources Library

The actors in the `Actors→Sources` library are mainly used to provide sources of signals [1]. Here, we will mainly consider the following three actors.



- **Const** produces a constant output whose value is provided by the *value* parameter by default [1]; however, if the value of the *firingCountLimit* parameter is set as a positive integer rather than the default "NONE", when the number of iterations of this actor is greater than that integer, it produces `absent`.

Let the variable $rOut$ denote the relation connected to this output port, and then we will encode this actor by categorizing two situations:

Situation 1: If the value of the *firingCountLimit* parameter is "NONE", we can encode this actor as

$$T_{Const}[i] \ \triangleq \ rOut[i] = value;$$

Situation 2: If the value of the *firingCountLimit* parameter is a positive integer, we need to introduce another variable $iteNum$ to denote the iteration number of this actor and set its initial value as $iteNum[0] = 0$. In this situation, this actor will be encoded as

$$\begin{aligned} T_{Const}[i] \triangleq \Big(&\Big(iteNum[i] \leq firingCountLimit \wedge rOut[i] = value\Big) \\ &\vee \Big(iteNum[i] > firingCountLimit \wedge rOut[i] = \texttt{absent}\Big)\Big) \\ &\wedge \Big(iteNum[i] = iteNum[i-1] + 1\Big) \end{aligned}$$

Note: In order to reduce the amount of variables, we will not introduce additional variables to denote the built-in parameters of an actor (e.g. *value* and $firingCountLimit$ here), because they can be replaced by their concrete values during our automatic encoding process.

- **Ramp** produces a stably increasing or decreasing value sequence along with the number of iteration by default [1]; however, similar to Const actor, if the value of the *firingCountLimit* is set to be a positive integer rather than the default "NONE", when the number of iterations of this actor is greater than that integer, it also produces `absent`. Since this actor is commonly used to provide the source data for an actor-oriented models in Ptolemy II, here, we will mainly consider the following situations where its input ports are not connected to any other actors. In this case, the first output of this actor is given by the *init* parameter, and the increment in the subsequent outputs is given by the *step* parameter.

Let $rOut$ be the relation connected to this output port and ST_{Ramp} be the state of this actor, where the initial value of ST_{Ramp} is 0 (i.e. $ST_{Ramp}[0] = 0$) and increase by 1 along with the iteration of this actor. And then, two situations should be taken into consideration.

Situation 1: If the value of the *firingCountLimit* parameter is "NONE", we can encode this actor as

$$\begin{aligned} T_{Ramp}[i] \triangleq & \Big(ST_{Ramp}[i-1] = 0 \wedge rOut[i] = init \wedge ST_{Ramp}[i] = 1 \Big) \\ & \vee \Big(ST_{Ramp}[i-1] > 0 \wedge rOut[i] = rOut[i-1] + step \\ & \quad \wedge ST_{Ramp}[i] = ST_{Ramp}[i-1] + 1 \Big) \end{aligned}$$

Situation 2: If the value of the *firingCountLimit* parameter is a positive integer, this actor will be encoded as

$$\begin{aligned} & T_{Ramp}[i] \\ \triangleq & \Big(ST_{Ramp}[i-1] = 0 \wedge rOut[i] = init \wedge ST_{Ramp}[i] = 1 \Big) \vee \\ & \Big(ST_{Ramp}[i-1] > 0 \wedge ST_{Ramp}[i-1] < firingCountLimit \wedge rOut[i] = \\ & rOut[i-1] + step \wedge ST_{Ramp}[i] = ST_{Ramp}[i-1] + 1 \Big) \vee \\ & \Big(ST_{Ramp}[i-1] \geq firingCountLimit \wedge rOut[i] = \texttt{absent} \wedge ST_{Ramp}[i] = \\ & ST_{Ramp}[i-1] + 1 \Big) \end{aligned}$$

Note: Some actors in Ptolemy II have states, that is, when these actors are initialized at some tick, they will change their behavior as that when they are in their initial states. For example, as to the *Ramp* actor, normally, the output of this actor at tick $i+1 (i > 0)$ is $rOut[i+1] = rOut[i] + step$; however, if it is initialized at tick i, then the output of this actor at tick $i+1 (i > 0)$ will become $rOut[i+1] = init$.

Let us review the `SimpleTrafficLight` case, the actor named Sec is a Ramp actor, so we can encode it as:

$$T_{Sec}[i] \triangleq \Big(ST_{Sec}[i-1] = 0 \wedge r_1[i] = 0 \wedge ST_{Sec}[i] = 1 \Big) \vee$$

$$\Big(ST_{Sec}[i-1] > 0 \land r_1[i] = r_1[i-1] + 1 \land ST_{Sec}[i] = ST_{Sec}[i-1] + 1\Big),$$

where the variable ST_{Sec} is used to denote the state of the Sec actor (initially, $ST_{Sec}[0] = 0$) and should be added to the Tab. **??**.

- **Sequence** produces a sequence of values specified by the *value* parameter, which will provide one element on the output port at each iteration [1]. Additionally, if the *repeat* parameter is true, this sequence will be repeatedly output; otherwise, if the *holdLastOutput* parameter is true, the last element in this sequence will be repeated; however, if the *holdLastOutput* parameter is false, when all of the elements in this array are output, this actor will output `absent` in its subsequent iterations.

Let the variables $rOut$ and $index$, and the uninterpreted function $s(index)$ denote respectively, the relation connected to the output port, the index of the sequence in *value* parameter, and the element indexed by $index$ in this sequence. Suppose the length of this sequence is $SeqNum$, then the domain of the variable $index$ is $[1, SeqNum]$, and $s(1)$, $s(2)$, ..., $s(SeqNum)$ will correspond to every element in this sequence successively. In addition, the initial value of $index$ is 1, that is, $index[1] = 1$. Similar to the Ramp actor, for the encoding of this actor, we will mainly consider the situations where the input ports are without connection, since this actor is also often used to give the source of data. Here three situations are considered:

Situation 1: If $repeat = true$, then

$$\begin{aligned} T_{Sequence}[i] \triangleq & \Big(index[i] \geq 1 \land index[i] \leq SeqNum \land rOut[i] = s(index[i]) \\ & \land index[i+1] = index[i] + 1\Big) \\ & \lor \Big(index[i] > SeqNum \land rOut[i] = s(1) \land index[i+1] = 2\Big) \end{aligned}$$

Situation 2: If $repeat = false$ and $holdLastOutput = true$, then

$$\begin{aligned} T_{Sequence}[i] \triangleq & \Big(index[i] \geq 1 \land index[i] \leq SeqNum \land rOut[i] = s(index[i]) \\ & \land index[i+1] = index[i] + 1\Big) \\ & \lor \Big(index[i] > SeqNum \land rOut[i] = s(SeqNum) \land index[i+1] \\ & \quad = index[i]\Big) \end{aligned}$$

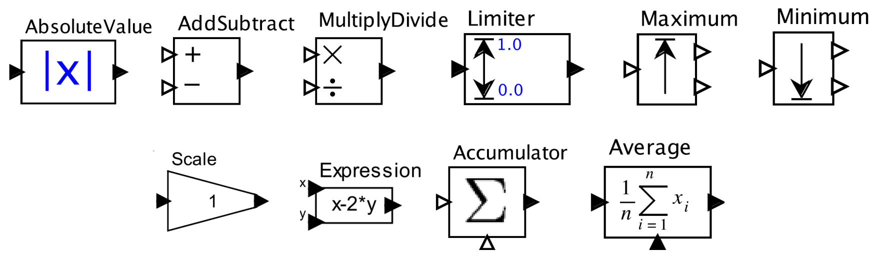
Situation 3: If $repeat = false$ and $holdLastOutput = false$, then

$$\begin{aligned} T_{Sequence}[i] \triangleq & \Big(index[i] \geq 1 \land index[i] \leq SeqNum \land rOut[i] = s(index[i]) \\ & \land index[i+1] = index[i] + 1\Big) \\ & \lor \Big(index[i] > SeqNum \land rOut[i] = \texttt{absent} \land index[i+1] \end{aligned}$$

$$= index[i]\Big)$$

C. Math Library

The actors in `Actors`→`Math` library perform mathematical operations [1]. The actors below are mainly considered here.



• **AbsoluteValue** produces the absolute of its input, however, it will produce `absent` if the input is `absent` [1]. Let the variables rIn and $rOut$ be the relations connected to the input and the output of this actor respectively. Suppose the function *abs* is used to compute an absolute value, and then, this actor can be encoded as:

$$T_{AbsoluteValue}[i] \triangleq \Big(\textit{is-value}(rIn[i]) \wedge rOut[i] = abs(rIn[i]) \Big) \\ \vee \Big(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Big)$$

• **AddSubtract** adds the data arriving at the *plus* input port, and subtracts the data arriving at the *minus* input port [1]. Both of these two input ports are multiports, each of which will connect multiple channels. However, it does not require that these two input ports both have connections, or each input channel connected to the input ports must have a token. Therefore, this actor will add or subtract the available data at the input channels but ignore those input channels or ports without any token.

Let the variables $rAIn_1$, $rAIn_2$, ..., $rAIn_m$ be those relations connected to the *plus* input port if there are, the variables sA_1, sA_2,..., sA_m be the values at those channels, and the variables $nilNumA_1$, $nilNumA_2$, ..., $nilNumA_m$ be the flags whether those channels are without tokens. If some channel $j(1 \leq j \leq m)$ has no token, then we will set as $sA_j = 0$ and $nilNumA_j = 1$; however, if some channel $r(1 \leq r \leq m)$ has an token, then we will set sA_r as the concrete value of that token, and $nilNumA_j$ as 0. Similarly, let the variables $rMIn_1$, $rMIn_2$, ..., $rMIn_n$ be these relations connected the *minus* input port if there are, the variables $sM_1, sM_2, ..., sM_n$ be the values at those channels, and the variables $nilNumM_1, nilNumM_2, ..., nilNumM_n$ be the flags whether these channels are without tokens. The evaluation of these two groups of variables are set in the same way as above. That is, if some channel $j'(1 \leq j' \leq m)$ has no token, then we will set as $sM_{j'} = 0$ and $nilNumM_{j'} = 1$; however, if some channel $r'(1 \leq r' \leq n)$ has an token, then we will set $sM_{r'}$ as the concrete value of that token, and $nilNumM_{r'}$ as 0. Let $rOut$ be the relation connected to the output port, and then, we will give the encoding of this actor in the following three situations:

Situation 1: Suppose there is no connection to the *minus* input port, but there are m relations connected to the *plus* input port, and then, we can encode this actor as follows:

$$
\begin{aligned}
& T_{AddSubtract}[i] \\
\triangleq & \bigwedge_{j=1}^{m} \Bigg(\Big(\textit{is-value}(rAIn_j[i]) \wedge sA_j[i] = rAIn_j[i] \wedge nilNumA_j[i] = 0 \Big) \\
& \qquad \vee \Big(rAIn_j[i] = \texttt{absent} \wedge sA_j[i] = 0 \wedge nilNumA_j[i] = 1 \Big) \Bigg) \\
& \wedge \Bigg(\Bigg(\sum_{j=1}^{m} nilNumA_j[i] \neq m \wedge rOut[i] = \sum_{j=1}^{m} sA_j[i] \Bigg) \\
& \qquad \vee \Bigg(\sum_{j=1}^{m} nilNumA_j[i] = m \wedge rOut[i] = \texttt{absent} \Bigg) \Bigg)
\end{aligned}
$$

Situation 2: Suppose there is no connection to the *plus* input port, but there are n channels connected to the *minus* input port, and then, we can encode this actor as follows:

$$
\begin{aligned}
& T_{AddSubtract}[i] \\
\triangleq & \bigwedge_{j=1}^{n} \Bigg(\Big(\textit{is-value}(rMIn_j[i]) \wedge sM_j[i] = rMIn_j[i] \wedge nilNumM_j[i] = 0 \Big) \\
& \qquad \vee \Big(rMIn_j[i] = \texttt{absent} \wedge sM_j[i] = 0 \wedge nilNumM_j[i] = 1 \Big) \Bigg) \\
& \wedge \Bigg(\Bigg(\sum_{j=1}^{n} nilNumM_j[i] \neq n \wedge rOut[i] = \sum_{j=1}^{n} -sM_j[i] \Bigg) \\
& \qquad \vee \Bigg(\sum_{j=1}^{n} nilNumM_j[i] = n \wedge rOut[i] = \texttt{absent} \Bigg) \Bigg)
\end{aligned}
$$

Situation 3: Suppose there are m channels connecting to the *plus* input port, and n channels connecting to the *minus* input port, and then this actor can be encoded as follows:

$$
\begin{aligned}
&T_{AddSubtract}[i] \\
\triangleq &\bigwedge_{j=1}^{m} \Bigg[\bigg(\textit{is-value}(rAIn_j[i]) \wedge sA_j[i] = rAIn_j[i] \wedge nilNumA_j[i] = 0 \bigg) \\
&\quad \vee \bigg(rAIn_j[i] = \texttt{absent} \wedge sA_j[i] = 0 \wedge nilNumA_j[i] = 1 \bigg) \Bigg] \\
&\wedge \bigwedge_{j=1}^{n} \Bigg[\bigg(\textit{is-value}(rMIn_j[i]) \wedge sM_j[i] = rMIn_j[i] \wedge nilNumM_j[i] = 0 \bigg) \\
&\quad \vee \bigg(rMIn_j[i] = \texttt{absent} \wedge sM_j[i] = 0 \wedge nilNumM_j[i] = 1 \bigg) \Bigg] \\
&\wedge \Bigg\{ \Bigg[\bigg(\sum_{j=1}^{m} nilNumA_j[i] \neq m \vee \sum_{j=1}^{n} nilNumM_j[i] \neq n \bigg) \\
&\quad \wedge \bigg(rOut[i] = \sum_{j=1}^{m} sA_j[i] - \sum_{j=1}^{n} sM_j[i] \bigg) \Bigg] \\
&\quad \vee \Bigg[\bigg(\sum_{j=1}^{m} nilNumA_j[i] = m \wedge \sum_{j=1}^{n} nilNumM_j[i] = n \bigg) \wedge rOut[i] = \texttt{absent} \Bigg] \Bigg\}
\end{aligned}
$$

- **MultiplyDivide** multiplies the data arriving at the *multiply* input port, and subtracts the data arriving at the *divide* input port [1]. Similar to the **AddSubtract** actor, both of these two input ports are also multiports, each of which will connect multiple channels. However, it does not require that these two input ports both have connections, or each input channel connected to the input ports must have a token. Therefore, this actor will multiply or divide the available data at the input channels but ignore those input channels or ports without any token.

Let the variables $rMIn_1$, $rMIn_2$, ..., $rMIn_m$ be those channels connected to the *multiply* input port, the variables sM_1, sM_2,..., sM_m be the values at the channels, and the variables $nilNumM_1$, $nilNumM_2$, ...,$nilNumM_m$ be the flags whether those channels are without tokens. If some channel $j(1 \leq j \leq m)$ has no token, then we will set as $sM_j = 1$ and $nilNumM_j = 1$; however, if some channel $r(1 \leq r \leq m)$ has an token, then we will set sM_j as the concrete value of that token, and $nilNumM_j$ as 0. Similarly, let the variables $rDIn_1, rDIn_2, ..., rDIn_n$ be the channels connected to the *divide* input port, the variables $sD_1, sD_2, ..., sD_n$ be the values at those channels, and the variables $nilNumD_1, nilNumD_2, ..., nilNumD_m$ be the flags whether those channels are without tokens. The evaluation of these two groups of variables are set in the same way as above. If some channel $j'(1 \leq j' \leq n)$ has no token, then we will set as $sD_j = 1$ and $nilNumD_j = 1$; however, if some channel $r'(1 \leq r' \leq n)$ has an token, then we will set sD_j as the concrete value of that token, and $nilNumD_j$ as 0. Let $rOut$ be the relation connected to the output port, and then, we will give the encoding of this actor in the following three situations.

Situation 1: Suppose there is no connection to the *divide* input port, but there are m channels connected to the *multiply* input port, and Based on the above assumptions, we can encode this actor as follows:

$$
\begin{aligned}
& T_{MultiplyDivide}[i] \\
\triangleq & \bigwedge_{j=1}^{m} \Bigg(\bigg(\textit{is-value}(rMIn_j[i]) \wedge sM_j[i] = rMIn_j[i] \wedge nilNumM_j[i] = 0 \bigg) \\
& \quad \vee \bigg(rMIn_j[i] = \texttt{absent} \wedge sM_j[i] = 1 \wedge nilNumM_j[i] = 1 \bigg) \Bigg) \\
& \wedge \Bigg(\bigg(\sum_{j=1}^{m} nilNumM_j[i] \neq m \wedge rOut[i] = \prod_{j=1}^{m} sM_j[i] \bigg) \\
& \quad \vee \bigg(\sum_{j=1}^{m} nilNumM_j[i] = m \wedge rOut[i] = \texttt{absent} \bigg) \Bigg)
\end{aligned}
$$

Situation 2: Suppose there is no connection to the *multiply* input port, but there are n channels connected to the *divide* input port. And then, we can encode this actor as follows:

$$
\begin{aligned}
& T_{MultiplyDivide}[i] \\
\triangleq & \bigwedge_{j=1}^{n} \Bigg(\bigg(\textit{is-value}(rDIn_j[i]) \wedge sM_j[i] = rDIn_j[i] \wedge nilNumD_j[i] = 0 \bigg) \\
& \quad \vee \bigg(rDIn_j[i] = \texttt{absent} \wedge sD_j[i] = 1 \wedge nilNumD_j[i] = 1 \bigg) \Bigg) \\
& \wedge \Bigg(\bigg(\sum_{j=1}^{n} nilNumD_j[i] \neq n \wedge rOut[i] = \prod_{j=1}^{n} (1/sM_j[i]) \bigg) \\
& \quad \vee \bigg(\sum_{j=1}^{n} nilNumD_j[i] = n \wedge rOut[i] = \texttt{absent} \bigg) \Bigg)
\end{aligned}
$$

Situation 3: Suppose there are m channels connecting to the *multiply* input port and n channels connecting to the *divide* input port. And then, this actor can be encoded as follows:

$$
\begin{aligned}
& T_{MultiplyDivide}[i] \\
\triangleq & \bigwedge_{j=1}^{m} \Bigg[\bigg(\textit{is-value}(rMIn_j[i]) \wedge sM_j[i] = rMIn_j[i] \wedge nilNumM_j[i] = 0 \bigg)
\end{aligned}
$$

$$
\begin{aligned}
&\vee \Big(rMIn_j[i] = \texttt{absent} \wedge sM_j[i] = 1 \wedge nilNumM_j[i] = 1\Big)\Big]\\
&\wedge \bigwedge_{j=1}^{n} \Big[\Big(is\text{-}value(rDIn_j[i]) \wedge sD_j[i] = rDIn_j[i] \wedge nilNumD_j[i] = 0\Big)\\
&\vee \Big(rDIn_j[i] = \texttt{absent} \wedge sM_j[i] = 1 \wedge nilNumD_j[i] = 1\Big)\Big]\\
&\wedge \Big\{\Big[\Big(\sum_{j=1}^{m} nilNumM_j[i] \neq m \vee \sum_{j=1}^{n} nilNumD_j[i] \neq n\Big)\\
&\wedge\Big(rOut[i] = \prod_{j=1}^{m} sM_j[i] * \prod_{j=1}^{n}(1/sD_j[i])\Big)\Big]\\
&\vee\Big[\Big(\sum_{j=1}^{m} nilNumM_j[i] = m \wedge \sum_{j=1}^{n} nilNumD_j[i] = n\Big) \wedge rOut[i] = \texttt{absent}\Big]\Big\}
\end{aligned}
$$

- **Limiter** produces the input value if it lies between the built-in *bottom* and *top* parameters of this actor; otherwise, if the input value is greater than *top*, then outputs *top*, while if the input value is less than *bottom*, then outputs *bottom*[1]. Additionally, this actor produces `absent` if the input is `absent`. Let rIn and $rOut$ denote the input and the output of this actor respectively, then we can encode this actor as:

$$
\begin{aligned}
&T_{Limiter}[i]\\
&\triangleq \Big[is\text{-}value(rIn[i]) \wedge \Big(\Big(rIn[i] \geq bottom \wedge rIn[i] \leq top \wedge rOut[i] = rIn[i]\Big)\\
&\vee \Big(rIn[i] > top \wedge rOut[i] = top\Big) \vee \Big(rIn[i] < bottom \wedge rOut[i] = bottom\Big)\Big)\Big]\\
&\vee \Big[rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent}\Big]
\end{aligned}
$$

- **Maximum & Minimum** actors are discussed together for the sake of simplicity, because their function mechanisms are similar.

Maximum (**Minimum**) produces the maximum (minimum) of the current input tokens at the $maximumValue$ ($minimumValue$) output port and the channel number of the maximum (minimum) at the *channelNumber* output port [1]. Either output port can be left unconnected if its results are unnecessary. Additionally, only when all the input channels are `absent`, will these two output ports produce `absent`.

Suppose the variables rIn_1, rIn_2, ..., rIn_m denote the tokens arriving at the channel 0, channel 1, ..., channel m of the input ports of **Maximum** (**Minimum**) actor respectively, and let the variables $rMax$ ($rMin$) and $rMaxCN$ ($rMinCN$) represent, respectively, the outputs of the $maximumValue$ ($minimumValue$) and the $channelNumber$ ports. Additionally, we also introduce two groups of variables $s_1, s_2, ..., s_m$ and $CN_1, CN_2, ..., CN_m$, where $s_j (1 \leq j \leq m)$ denotes the max (min) values of tokens from channel 0 to the currently considering channel j in turn at some certain tick, while CN_j represents the corresponding channel number of the value of s_j. And also, if the input tokens up to channel j are all `absent`, then we set $s_j = \texttt{absent}, CN_j = \texttt{absent}$. Given these variables, we can encode the behavior of the **Maximum** actor as follows:

$$
\begin{aligned}
& T_{Maximum}[i] \\
\triangleq & \Bigg\{ s_1[i] = rIn_1[i] \wedge \Bigg[\Bigg(\textit{is-value}(rIn_1[i]) \wedge CN_1[i] = 0 \Bigg) \vee \Bigg(rIn_1[i] = \texttt{absent} \\
& \quad \wedge CN_1[i] = \texttt{absent} \Bigg) \Bigg] \Bigg\} \\
& \wedge \bigwedge_{j=2}^{m} \Bigg\{ \Bigg[\Bigg(rIn_j[i] = \texttt{absent} \vee \Bigg(\textit{is-value}(rIn_j[i]) \wedge \textit{is-value}(s_{j-1}[i]) \wedge rIn_j[i] \\
& \quad \leq s_{j-1}[i] \Bigg) \Bigg) \wedge s_j[i] = s_{j-1}[i] \wedge CN_j[i] = CN_{j-1}[i] \Bigg] \\
& \quad \vee \Bigg[\Bigg(\textit{is-value}(rIn_j[i]) \wedge \Bigg(s_{j-1}[i] = \texttt{absent} \vee \Bigg(\textit{is-value}(s_{j-1}[i]) \wedge rIn_j[i] \\
& \qquad > s_{j-1}[i] \Bigg) \Bigg) \Bigg) \wedge s_j[i] = rIn_j[i] \wedge CN_j[i] = j - 1 \Bigg] \Bigg\} \\
& \wedge rMax[i] = s_m[i] \\
& \wedge rMaxCN[i] = CN_m[i].
\end{aligned}
$$

For the encoding of the **Minimum** actor, we have

$$
\begin{aligned}
& T_{Minimum}[i] \\
\triangleq & \Bigg\{ s_1[i] = rIn_1[i] \wedge \Bigg[\Bigg(\textit{is-value}(rIn_1[i]) \wedge CN_1[i] = 0 \Bigg) \vee \Bigg(rIn_1[i] = \texttt{absent} \\
& \quad \wedge CN_1[i] = \texttt{absent} \Bigg) \Bigg] \Bigg\}) \\
& \wedge \bigwedge_{j=2}^{m} \Bigg\{ \Bigg[\Bigg(rIn_j[i] = \texttt{absent} \vee \Bigg(\textit{is-value}(rIn_j[i]) \wedge \textit{is-value}(s_{j-1}[i]) \wedge rIn_j[i] \\
& \quad \geq s_{j-1}[i] \Bigg) \Bigg) \wedge s_j[i] = s_{j-1}[i] \wedge CN_j[i] = CN_{j-1}[i] \Bigg]
\end{aligned}
$$

$$\vee \Bigg[\Bigg(\textit{is-value}(rIn_j[i]) \wedge \Big(s_{j-1}[i] = \texttt{absent} \vee \Big(\textit{is-value}(s_{j-1}[i]) \wedge rIn_j[i] < s_{j-1}[i]\Big)\Big)\Bigg) \wedge s_j[i] = rIn_j[i] \wedge CN_j[i] = j-1\Bigg]\Bigg\}$$
$$\wedge\, rMin[i] = s_m[i]$$
$$\wedge\, rMinCN[i] = CN_m[i]$$

Here we mainly consider the situation when the $maximumValue$ ($minimum Value$) and the $channelNumber$ output ports are both connected. However, if the $channelNumber$ output port is left unconnected, we just need to remove the subexpressions $((\textit{is-value}(rIn_1[i]) \wedge CN_1[i] = 0) \vee (rIn_1[i] = \texttt{absent} \wedge CN_1[i] = \texttt{absent}))$, $CN_j[i] = CN_{j-1}[i]$, $CN_j[i] = j-1$ and $rMaxCN[i] = CN_m[i]$ or $rMinCN[i] = CN_m[i]$ from the above expressions. While, if the $maximumValue$ ($minimumValue$) output port is left unconnected, then we only need to remove the subexpression $rMax[i] = s_m[i]$ or $rMin[i] = s_m[i]$.

- **Scale** outputs a value determined by multiplying the input by the $factor$ parameter; while if no input token is available, then this actor outputs `absent` [1]. Let the variables rIn and $rOut$ denote the relations connecting to the input and the output of this actor respectively, then we can encode this actor as:

$$T_{Scale} \triangleq \Big(\textit{is-value}(rIn[i]) \wedge rOut[i] = c * rIn[i]\Big) \vee \Big(rIn[i] = absent \wedge rOut[i] = absent\Big)$$

- **Expression** outputs the value of the given expression, and if one input is absent, then this actor outputs `absent` [1]. By default, this actor has one output and no input, therefore, when using this actor, one has to add input ports firstly and type in a unique name for those added ports. And then, an expression can be specified using the port names as variables. For the Expression actor in the above figure, we have added two input ports named "x" and "y" and have defined the expression as "$x - 2 * y$". Let the variables $rxIn$, $ryIn$ and $rOut$ denote the tokens arriving at x input port, y input port and the output port of this actor respectively, then this actor can be encoded as:

$$T_{Expression}[i]$$
$$\triangleq \Big(\textit{is-value}(rxIn[i]) \wedge \textit{is-value}(ryIn[i]) \wedge rOut[i] = rxIn[i] - 2 * ryIn[i]\Big)$$
$$\vee \Big(\Big(rxIn[i] = \texttt{absent} \vee ryIn[i] = \texttt{absent}\Big) \wedge rOut[i] = \texttt{absent}\Big)$$

- **Accumulator** produces the sum of the built-in $init$ parameter and all tokens that have arrived at the input port, from the last tick when a true token is received at the *reset* port [1]. In particular, if the parameter *lowerBound* is set, then the output is restricted to be greater than or equal to the value of *lowerBound* and the accumulation in the next tick is on the basis of *lowerBound* rather than the real sum at the last tick. Similarly, the parameter *upperBound* restricts the upper bound of the output of this actor.

Suppose two groups of variables rIn_1, rIn_2, ..., rIn_m and s_1, s_2, ..., s_m, are introduces to denote respectively, the input at each channel of the left input port and the values of those input tokens (if some input rIn_j at tick i is `absent`, then we set $s_j[i] = 0$). Similarly, another two groups of variables $rRIn_1, rRIn_2, ..., rRIn_n$ and $b_1, b_2, ..., b_n$ are used to represent the inputs arriving at each channel of the *reset* input port and the values of those input tokens (if some input $rRIn_j$ at tick i is `absent`, then we set $b_j[i] = false$). Additionally, let *rOut* denote the output of this actor, and *sum* denote the sum of all input tokens from the last tick when a true token is received at the *reset* port (initially, $sum[0] = init$). If the *lowerBound* and *upperBound* parameters are set, then the value of *sum* at every tick is restricted to the range between *lowerBound* and *upperBound*. With these variables, this actor can be encoded as follows.

Situation 1: If the *reset* port is unconnected and the parameters *lowerBound* and *upperBound* are not set, then we can get that

$$
\begin{aligned}
&T_{Accumulator}[i] \\
&\triangleq \bigwedge_{j=1}^{m} \Bigg(\bigg(\textit{is-value}(rIn_j[i]) \wedge s_j[i] = rIn_j[i] \bigg) \vee \bigg(rIn_j[i] = \texttt{absent} \wedge s_j[i] = 0 \bigg) \Bigg) \\
&\quad \wedge sum[i] = sum[i-1] + \sum_{j=1}^{n} s_j[i] \wedge rOut[i] = sum[i]
\end{aligned}
$$

Situation 2: If the *reset* port is connected, and the parameters *lowerBound* and *upperBound* are not set, then we can get that

$$
\begin{aligned}
&T_{Accumulator}[i] \\
&\triangleq \bigwedge_{j=1}^{m} \Bigg(\bigg(\textit{is-value}(rIn_j[i]) \wedge s_j[i] = rIn_j[i] \bigg) \vee \bigg(rIn_j[i] = \texttt{absent} \wedge s_j[i] = 0 \bigg) \Bigg) \\
&\quad \wedge \bigwedge_{j=1}^{n} \Bigg(\bigg(\textit{is-value}(rRIn_j[i]) \wedge b_j[i] = rRIn_j[i] \bigg) \vee \bigg(rRIn_j[i] = \texttt{absent} \wedge b_j[i] \\
&\qquad = false \bigg) \Bigg) \\
&\quad \wedge \Bigg(\bigg(\bigwedge_{j=1}^{n} b_j[i] = true \wedge sum[i] = init + \sum_{j=1}^{m} s_j[i] \bigg) \vee \bigg(\bigwedge_{j=1}^{n} b_j[i] = false \wedge sum[i] \\
&\qquad = sum[i-1] + \sum_{j=1}^{m} s_j[i] \bigg) \Bigg)
\end{aligned}
$$

$\wedge \, rOut[i] = sum[i];$

Situation 3: If the *reset* port is connected, and the parameter *lowerBound* and/or *upperBound* are set, then we can get that

$$
\begin{aligned}
& T_{Accumulator}[i] \\
\triangleq & \bigwedge_{j=1}^{m} \Big(\big(is\text{-}value(rIn_j[i]) \wedge s_j[i] = rIn_j[i] \big) \vee \big(rIn_j[i] = \texttt{absent} \wedge s_j[i] = 0 \big) \Big) \\
& \wedge \bigwedge_{j=1}^{n} \Big(\big(is\text{-}value(rRIn_j[i]) \wedge b_j[i] = rRIn_j[i] \big) \vee \big(rRIn_j[i] = \texttt{absent} \wedge b_j[i] \\
& \quad = false \big) \Big) \\
& \wedge \Bigg\{ \Bigg[\bigwedge_{j=1}^{n} b_j[i] = true \wedge \Bigg(\Bigg(\sum_{j=1}^{m} s_j[i] + init \geq upperBound \wedge sum[i] = \\
& \quad upperBound \Bigg) \vee \Bigg(\sum_{j=1}^{m} s_j[i] + init \leq lowerBound \wedge sum[i] = lowerBound \Bigg) \\
& \quad \vee \Bigg(\sum_{j=1}^{m} s_j[i] + init > lowerBound \wedge \sum_{j=1}^{m} s_j[i] + init < upperBound \wedge sum[i] = \\
& \quad \sum_{j=1}^{m} s_j[i] + init \Bigg) \Bigg) \Bigg] \\
& \quad \vee \Bigg[\bigwedge_{j=1}^{n} b_j[i] = false \wedge \Bigg(\Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] \geq upperBound \wedge sum[i] = \\
& \quad upperBound \Bigg) \vee \Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] \leq lowerBound \wedge sum[i] = \\
& \quad lowerBound \Bigg) \vee \Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] > lowerBound \wedge sum[i-1] + \\
& \quad \sum_{j=1}^{m} s_j[i] < upperBound \wedge sum[i] = sum[i-1] + \sum_{j=1}^{m} s_j[i] \Bigg) \Bigg) \Bigg] \Bigg\} \\
& \wedge \, rOut[i] = sum[i],
\end{aligned}
$$

where if the parameter *lowerBound* is not set, we just need to remove *lowerBound−related* subexpressions $sum[i-1]+\sum_{j=1}^{m} s_j[i] \leq lowerBound \wedge sum[i] = lowerBound$, $\sum_{j=1}^{m} s_j[i] + init > lowerBound$, $\sum_{j=1}^{m} s_j[i] + init \leq lowerBound \wedge sum[i] = lowerBound$, and $sum[i-1]+\sum_{j=1}^{m} s_j[i] > lowerBound$. While, if the parameter *upperBound* is not set, similarly, we only need to remove the *upperBounded − related* subexpressions.

Situation 4: If the *reset* port is unconnected, and the parameter *lowerBound* and/or are set, then we can get that

$$
\begin{aligned}
&T_{Accumulator}[i]\\
&\triangleq \bigwedge_{j=1}^{m} \Bigg(\Big(\textit{is-value}(rIn_j[i]) \wedge s_j[i] = rIn_j[i] \Big) \vee \Big(rIn_j[i] = \texttt{absent} \wedge s_j[i] = 0 \Big) \Bigg)\\
&\quad \wedge \Bigg[\Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] \geq upperBound \wedge sum[i] = upperBound \Bigg)\\
&\qquad \vee \Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] \leq lowerBound \wedge sum[i] = lowerBound \Bigg)\\
&\qquad \vee \Bigg(sum[i-1] + \sum_{j=1}^{m} s_j[i] > lowerBound \wedge sum[i-1] + \sum_{j=1}^{m} s_j[i] <\\
&\qquad\qquad upperBound \wedge sum[i] = sum[i-1] + \sum_{j=1}^{m} s_j[i] \Bigg) \Bigg]\\
&\quad \wedge rOut[i] = sum[i].
\end{aligned}
$$

Similarly as Situation 3, if only one of the parameters *lowerBound* and *upperBound* is not set, we just need to remove these subexpressions related to that parameter without setting.

- **Average** produces the average of all tokens that have arrived at the input port, from the last tick when a true token is received at the *reset* port [1]. In particular, the output type is restricted to be the same as the input type, especially when the input type is *int* type.

For the convenience of encoding, we introduce the variables rIn, $rOut$ and $rRIn$ to denote respectively, the tokens of the left input, the output, and the reset input of this actor. Besides, the variables *sum* and *count* are introduced to denote the sum and the number of all input tokens from the last tick when a true token is received at the *reset* port (or from the first tick if the *reset* input port is left unconnected), therefore, they are initialized as $sum[0] = 0, count[0] = 0$. Given these assumptions, we can encode this actor as:

Situation 1: If the *reset* input port is left unconnected, then we can get that when the input type is not `int`, we have

$$
\begin{aligned}
&T_{Average}[i]\\
&\triangleq \Big(\textit{is-value}(rIn[i]) \wedge rOut[i] = sum[i]/count[i] \wedge sum[i] = sum[i-1] + rIn[i]\\
&\quad \wedge count[i] = coun[i-1] + 1 \Big)\\
&\vee \Big(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \wedge sum[i] = sum[i-1] \wedge count[i] =\\
&\quad count[i-1] \Big);
\end{aligned}
$$

when the input type is `int`, we have

$$
\begin{aligned}
&T_{Average}[i] \\
&\triangleq\Bigg(\textit{is-value}(rIn[i]) \wedge \bigg(\Big(\textit{sum}[i]/\textit{count}[i] \geq 0 \wedge rOut[i] = \textit{sum}[i]/\textit{count}[i]\Big) \\
&\quad \vee \Big(\textit{sum}[i]/\textit{count}[i] < 0 \wedge rOut[i] = -\textit{to\_int}(-\textit{sum}[i]/\textit{count}[i])\Big)\bigg) \\
&\quad \wedge \textit{sum}[i] = \textit{sum}[i-1] + rIn[i] \wedge \textit{count}[i] = \textit{count}[i-1] + 1\Bigg) \\
&\vee\Bigg(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \wedge \textit{sum}[i] = \textit{sum}[i-1] \wedge \textit{count}[i] = \\
&\quad \textit{count}[i-1]\Bigg),
\end{aligned}
$$

where $\textit{to\_int}$ is a function that maps each real number r to the largest integer less than or equal to r.

Situation 2: If the *reset* input port is connected, then we can get that when the input type is not `int`, we have

$$
\begin{aligned}
&T_{Average}[i] \\
&\triangleq\Bigg(\textit{is-value}(rIn[i]) \wedge \bigg(\Big(rRIn[i] = \textit{true} \wedge rOut[i] = rIn[i] \wedge \textit{sum}[i] = rIn[i] \\
&\quad \wedge \textit{count}[i] = 1\Big) \vee \Big(\big(rRIn[i] = \textit{false} \vee rRIn[i] = \texttt{absent}\big) \wedge rOut[i] = \\
&\quad \textit{sum}[i]/\textit{count}[i] \wedge \textit{sum}[i] = \textit{sum}[i-1] + rIn[i] \wedge \textit{count}[i] = \textit{count}[i-1] + 1\Big)\bigg)\Bigg) \\
&\vee\Bigg(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \wedge \textit{sum}[i] = \textit{sum}[i-1] \wedge \textit{count}[i] = \\
&\quad \textit{count}[i-1]\Bigg);
\end{aligned}
$$

when the input type is `int`, we have

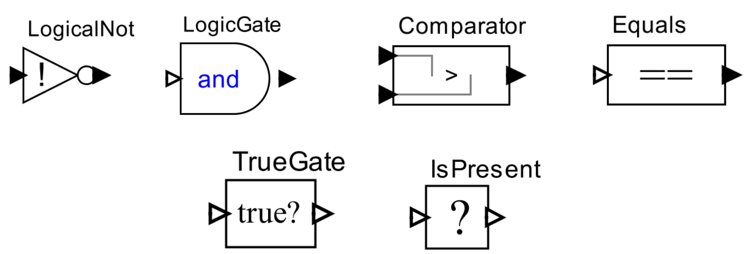
$$
\begin{aligned}
&T_{Average}[i] \\
&\triangleq\Bigg\{ \textit{is-value}(rIn[i]) \wedge \bigg[\Big(rRIn[i] = \textit{true} \wedge rOut[i] = rIn[i] \wedge \textit{sum}[i] = rIn[i] \wedge \\
&\quad \textit{count}[i] = 1\Big) \vee \bigg(\Big(rRIn[i] = \textit{false} \vee rRIn[i] = \texttt{absent}\Big) \wedge \bigg(\Big(\textit{sum}[i]/\textit{count}[i]
\end{aligned}
$$

$$\geq 0 \wedge rOut[i] = sum[i]/count[i]\Big) \vee \Big(sum[i]/count[i] < 0 \wedge rOut[i] = -to\_int$$
$$(-sum[i]/count[i])\Big)\Bigg) \wedge sum[i] = sum[i-1] + rIn[i] \wedge count[i] = count[i-1]$$
$$+1\Bigg)\Bigg]\Bigg\}$$
$$\vee\Bigg\{rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \wedge sum[i] = sum[i-1] \wedge count[i] =$$
$$count[i-1]\Bigg\},$$

where the function to_int is the same as that in Situation 1.

D. Logic Library

Ptolemy II provides the following six logic actors in the library `Actors`→`Logic` for building control logic. Among them, the function of the **TrueGate** and **IsPresent** actors are the same as that in the `DomainSpecific`→`SynchronousReactive` library, and therefore, we will not repeat the encoding of these two actors here.



- **LogicalNot** implements the logical "Not" or "¬" operator and outputs the negation of the input token; however, if no token arrives at the input port, then this actor outputs `absent` [1]. Let the variables rIn and $rOut$ denote the input and the output of this actor respectively, then we can encode the behavior this actor as:

$$T_{LogicalNot} \triangleq \Bigg(is\text{-}value(rIn[i]) \wedge rOut[i] = \neg rIn[i]\Bigg) \vee \Bigg(rIn[i] = \texttt{absent} \wedge$$
$$rOut[i] = \texttt{absent}\Bigg)$$

- **LogicGate** produces a value which is equal to the result of the specified logic operations on the left inputs, and these operations are: and, or, xor, nand, nor, xnor [1]; however, if there are no available input tokens, the actor produces `absent`. Let the variables $rIn_1, rIn_2, ..., rIn_n$ denote the relations connected to each channel of the left inputs, the variables $trueNumber_1, trueNumber_2, ...,$ $trueNumber_n$ denote the number of the true input token at each channel, and the variables $rOut$ be the relation connected to the output of this actor, then we can encode the behaviors of this actor as follows.

1) If the operation is "and", then we can get

$$
\begin{aligned}
&T_{LogicGate}[i] \\
\triangleq &\left[\bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i]) \wedge \left(\left(\bigvee_{j=1}^{n}\Big(rIn_j[i] = false\Big) \wedge rOut[i] = false\right)\vee\right.\right. \\
&\left.\left.\left(\bigwedge_{j=1}^{n}\Big(rIn_j[i] = true \vee rIn_j[i] = \texttt{absent}\Big) \wedge rOut[i] = true\right)\right)\right] \\
&\vee \left[\bigwedge_{j=1}^{n}\left(rIn_j[i] = \texttt{absent}\right) \wedge rOut[i] = \texttt{absent}\right];
\end{aligned}
$$

2) If the operation is "or", then we can get

$$
\begin{aligned}
&T_{LogicGate}[i] \\
\triangleq &\left[\bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i]) \wedge \left(\left(\bigvee_{j=1}^{n}\Big(rIn_j[i] = true\Big) \wedge rOut[i] = true\right)\vee\right.\right. \\
&\left.\left.\left(\bigwedge_{j=1}^{n}\Big(rIn_j[i] = false \vee rIn_j[i] = \texttt{absent}\Big) \wedge rOut[i] = false\right)\right)\right] \\
&\vee \left[\bigwedge_{j=1}^{n}\left(rIn_j[i] = \texttt{absent}\right) \wedge rOut[i] = \texttt{absent}\right];
\end{aligned}
$$

3) If the operation is "nand", then we can get

$$
\begin{aligned}
&T_{LogicGate}[i] \\
\triangleq &\left[\bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i]) \wedge \left(\left(\bigvee_{j=1}^{n}\Big(rIn_j[i] = false\Big) \wedge rOut[i] = true\right)\vee\right.\right. \\
&\left.\left.\left(\bigwedge_{j=1}^{n}\Big(rIn_j[i] = true \vee rIn_j[i] = \texttt{absent}\Big) \wedge rOut[i] = false\right)\right)\right] \\
&\vee \left[\bigwedge_{j=1}^{n}\left(rIn_j[i] = \texttt{absent}\right) \wedge rOut[i] = \texttt{absent}\right];
\end{aligned}
$$

4) If the operation is "nor", then we can get

$$T_{LogicGate}[i]$$

$$\triangleq \Bigg[\bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i]) \wedge \Bigg(\Bigg(\bigvee_{j=1}^{n} \Big(rIn_j[i] = true \Big) \wedge rOut[i] = false \Bigg) \vee$$

$$\Bigg(\bigwedge_{j=1}^{n} \Big(rIn_j[i] = false \vee rIn_j[i] = \texttt{absent} \Big) \wedge rOut[i] = true \Bigg) \Bigg) \Bigg]$$

$$\vee \left[\bigwedge_{j=1}^{n} \left(rIn_j[i] = \texttt{absent} \right) \wedge rOut[i] = \texttt{absent} \right];$$

5) If the operation is "xor", then we can get

$$T_{LogicGate}[i]$$

$$\triangleq \Bigg\{ \bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i])$$

$$\wedge \Bigg[\bigvee_{j=1}^{n} \Bigg(\Big(rIn_j[i] = true \wedge trueNumber_j[i] = 1 \Big) \vee \Big(\Big(rIn_j[i] = false \vee$$

$$rIn_j[i] = \texttt{absent} \Big) \wedge trueNumber_j[i] = 0 \Big) \Bigg) \wedge \Bigg(\Big(\sum_{j=1}^{n} trueNumber_j[i]$$

$$= 0 \wedge rOut[i] = false \Big) \vee \Bigg(\Big(\sum_{j=1}^{n} trueNumber_j[i] \% 2 = 0 \wedge rOut[i] =$$

$$false \Big) \vee \Big(\sum_{j=1}^{n} trueNumber_j[i] \% 2 = 1 \wedge rOut[i] = true \Big) \Bigg) \Bigg) \Bigg] \Bigg\}$$

$$\vee \left\{ \bigwedge_{j=1}^{n} \left(rIn_j[i] = \texttt{absent} \right) \wedge rOut[i] = \texttt{absent} \right\};$$

5) If the operation is "xnor", then we can get

$$T_{LogicGate}[i]$$

$$\triangleq \Bigg\{ \bigvee_{j=1}^{n} \textit{is-value}(rIn_j[i])$$

$$\wedge \Bigg[\bigvee_{j=1}^{n} \Bigg(\Big(rIn_j[i] = false \wedge trueNumber_j[i] = 0 \Big) \vee \Big(\Big(rIn_j[i] = true \vee$$

$$rIn_j[i] = \texttt{absent} \Big) \wedge trueNumber_j[i] = 1 \Big) \Bigg) \wedge \Bigg(\Big(\sum_{j=1}^{n} trueNumber_j[i]$$

$$= 0 \wedge rOut[i] = false \Big) \vee \Bigg(\Big(\sum_{j=1}^{n} trueNumber_j[i] \% 2 = 0 \wedge rOut[i] =$$

$$true\Big) \vee \Big(\sum_{j=1}^{n} trueNumber_j[i]\%2 = 1 \wedge rOut[i] = false \Big)\Big)\Big)\Big]\Big]\Big\}$$

$$\vee \left\{ \bigwedge_{j=1}^{n} \left(rIn_j[i] = \texttt{absent} \right) \wedge rOut[i] = \texttt{absent} \right\};$$

- **Comparator** compares two inputs of double type (or any time being losslessly converted to double such as integer), and outputs true if the comparison test is satisfied, or false otherwise [1]. The *comparison* parameter gives the available comparison: $>$, $\geq$, $<$, $\leq$, $==$. While the *tolerance* parameter (by default, 0.0) gives an error tolerance: when it is equal to zero, the comparison result is true only the exact comparison is satisfied; when it is greater than zero, the comparison restriction is broaden, and then the comparison result can be true even if the comparison test is not exactly satisfied but rather satisfied within the *tolerance* parameter; however, when the *tolerance* parameter is less than zero, the comparison restriction is strengthen, and then the comparison result can be false even if the comparison test is exactly satisfied but rather satisfied within the *tolerance* parameter.

Let the variables $rLIn$, $rRIn$, and $rOut$ denote the $left$ input, the $right$ input and the *output* of this actor respectively, then we can encode this actor as follows.

1) If the *comparison* parameter takes the "==" value, then we can obtain

$$\begin{aligned} T_{Comparator} \triangleq \Bigg[& \textit{is-value}(rLIn[i]) \wedge \textit{is-value}(rRIn[i]) \wedge \Bigg(\Bigg(abs(rLIn[i] - rRIn[i]) \\ & \leq tolerance \wedge rOut[i] = true \Bigg) \vee \Bigg(abs(rLIn[i] - rRIn[i]) > \\ & tolerance \wedge rOut[i] = false \Bigg) \Bigg) \Bigg] \\ & \vee \Bigg[\Bigg(rLIn[i] = \texttt{absent} \vee rRIn[i] = \texttt{absent} \Bigg) \wedge rOut[i] = \texttt{absent} \Bigg] \end{aligned}$$

2) If the *comparison* parameter takes the ">" value, then we can obtain

$$\begin{aligned} T_{Comparator} \triangleq \Bigg[& \textit{is-value}(rLIn[i]) \wedge \textit{is-value}(rRIn[i]) \wedge \Bigg(\Bigg(rLIn[i] - rRIn[i] + \\ & tolerance > 0 \wedge rOut[i] = true \Bigg) \vee \Bigg(rLIn[i] - rRIn[i] + \\ & tolerance \leq 0 \wedge rOut[i] = false \Bigg) \Bigg) \Bigg] \\ & \vee \Bigg[\Bigg(rLIn[i] = \texttt{absent} \vee rRIn[i] = \texttt{absent} \Bigg) \wedge rOut[i] = \texttt{absent} \Bigg] \end{aligned}$$

3) If the *comparison* parameter takes the "$\geq$" value, then we can obtain

$$T_{Comparator} \triangleq \Bigg[\textit{is-value}(rLIn[i]) \land \textit{is-value}(rRIn[i]) \land \Bigg(\Bigg(rLIn[i] - rRIn[i] + tolerance \geq 0 \land rOut[i] = true \Bigg) \lor \Bigg(rLIn[i] - rRIn[i] + tolerance < 0 \land rOut[i] = false \Bigg) \Bigg) \Bigg] \lor \Bigg[\Bigg(rLIn[i] = \texttt{absent} \lor rRIn[i] = \texttt{absent} \Bigg) \land rOut[i] = \texttt{absent} \Bigg]$$

4) If the *comparison* parameter takes the "$<$" value, then we can obtain

$$T_{Comparator} \triangleq \Bigg[\textit{is-value}(rLIn[i]) \land \textit{is-value}(rRIn[i]) \land \Bigg(\Bigg(rRIn[i] - rLIn[i] + tolerance > 0 \land rOut[i] = true \Bigg) \lor \Bigg(rRIn[i] - rLIn[i] + tolerance \leq 0 \land rOut[i] = false \Bigg) \Bigg) \Bigg] \lor \Bigg[\Bigg(rLIn[i] = \texttt{absent} \lor rRIn[i] = \texttt{absent} \Bigg) \land rOut[i] = \texttt{absent} \Bigg]$$

5) If the *comparison* parameter takes the "$\leq$" value, then we can obtain

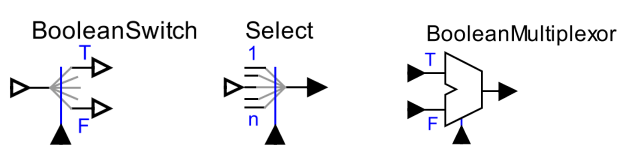
$$T_{Comparator} \triangleq \Bigg[\textit{is-value}(rLIn[i]) \land \textit{is-value}(rRIn[i]) \land \Bigg(\Bigg(rRIn[i] - rLIn[i] + tolerance \geq 0 \land rOut[i] = true \Bigg) \lor \Bigg(rRIn[i] - rLIn[i] + tolerance < 0 \land rOut[i] = false \Bigg) \Bigg) \Bigg] \lor \Bigg[\Bigg(rLIn[i] = \texttt{absent} \lor rRIn[i] = \texttt{absent} \Bigg) \land rOut[i] = \texttt{absent} \Bigg]$$

• **Equals** compares whether all available input tokens of any type at each tick are equal, and outputs true if equal, and false otherwise [1]; and only when there is no available input tokens for all input channels, will this actor output `absent`. Suppose the variables rIn_1, rIn_2, ..., rIn_m are used to denote the relation connected to each input channel, and let the variables s_1, s_2, ..., s_m record the value of the token arriving at each input channel. If some certain channel $j(1 \leq j \leq m)$ has no token, we let s_j take the value of another channel. For the sake of convenience, in this situation, we set $s_j[i] = s_{j-1}[i](2 \leq j \leq m)$, and if the first channel has no token, we set $s_1[i] = s_m[i]$. Additionally, let $rOut$ be the output of this actor, we can get the encoding as:

$$
\begin{aligned}
&T_{Equals} \\
&\triangleq \Bigg\{ \Bigg[\Bigg(\mathit{is\text{-}value}(rIn_1[i]) \wedge s_1[i] = rIn_1[i] \Bigg) \vee \Bigg(rIn_1[i] = \texttt{absent} \wedge s_1[i] = s_m[i] \Bigg) \Bigg] \\
&\quad \vee \bigwedge_{j=2}^{m} \Bigg[\Bigg(\mathit{is\text{-}value}(rIn_j[i]) \wedge s_j[i] = rIn_j[i] \Bigg) \vee \Bigg(rIn_j[i] = \texttt{absent} \wedge s_j[i] = s_{j-1}[i] \Bigg) \Bigg] \\
&\quad \vee \Bigg[\Bigg(\underbrace{s_1[i] = s_2[i] = s_3[i] = \cdots = s_{m-1}[i] = s_m[i]}_{mitems} \wedge rOut[i] = true \Bigg) \\
&\quad \vee \Bigg(\neg \Bigg(\underbrace{s_1[i] = s_2[i] = s_3[i] = \cdots = s_{m-1}[i] = s_m[i]}_{mitems} \Bigg) \wedge rOut[i] = false \Bigg) \Bigg] \Bigg\} \\
&\quad \vee \Bigg\{ \bigwedge_{j=1}^{m} \Bigg(rIn_j[i] = \texttt{absent} \Bigg) \wedge rOut[i] = \texttt{absent} \Bigg\}
\end{aligned}
$$

E. Flow Control Library

In this library, we mainly consider the following three representative and commonly used actors used to route tokens in a model, and these actors can be found in `Actors`$\rightarrow$`FlowControl` library.



- **BooleanSwitch** outputs the given inputs at the output port specified by the bottom *control* input port [1]. In an iteration, if the *control* input port has no available token, then the left inputs will be output at the same output port as that in last tick; while if the *control* input has never received any available token, then those inputs will be output at the $falseOutput$ port.

Let the variables $rIn_1, rIn_2, ..., rIn_m$ denote the input of every channel at the left *input* port respectively, and let another two groups of variables $rtOut_1, rtOut_2$, ..., $rtOut_m$ and $rfOut_1, rfOut_2, ..., rfOut_m$ denote respectively, the output at every channel of the *trueOutput* port, and the output at every channel of the

$falseOutput$ port. Additionally, suppose the variable $rCtrl$ represents the input at the bottom *control* port, and let the variables $lastCtrl$ and s record respectively, last non-absent input token (initially, we set $lastCtrl[0] = false$), and the most recently received non-absent input token at the *control* input port. And then, we can encode this actor as

$$
\begin{aligned}
&T_{BooleanSwitch}[i] \\
&\triangleq \Bigg\{ \Bigg(\textit{is-value}(rCtrl[i]) \wedge s[i] = rCtrl[i] \Bigg) \vee \Bigg(rCtrl[i] = \texttt{absent} \wedge s[i] = \\
&lastCtrl[i-1] \Bigg) \Bigg\} \wedge \Bigg\{ lastCtrl[i] = s[i] \Bigg\} \\
&\wedge \Bigg\{ \Bigg[s[i] = true \wedge \bigwedge_{j=1}^{m} \Bigg(\Big(\textit{is-value}(rIn_j[i]) \wedge rtOut_j[i] = rIn_j[i] \Big) \vee \Big(rIn_j[i] = \\
&\qquad \texttt{absent} \wedge rtOut_j[i] = \texttt{absent} \Big) \Bigg) \wedge \bigwedge_{j=1}^{m} \Bigg(rfOut_j[i] = \texttt{absent} \Bigg) \Bigg] \\
&\quad \vee \Bigg[s[i] = false \wedge \bigwedge_{j=1}^{m} \Bigg(\Big(\textit{is-value}(rIn_j[i]) \wedge rfOut_j[i] = rIn_j[i] \Big) \vee \Big(rIn_j[i] = \\
&\qquad \texttt{absent} \wedge rfOut[i] = \texttt{absent} \Big) \Bigg) \wedge \bigwedge_{j=1}^{m} \Big(rtOut[i] = \texttt{absent} \Big) \Bigg] \Bigg\}
\end{aligned}
$$

- **Select** outputs the token at the input channel specified by the *control* input port [1]. In an iteration, if the *control* input port has no available token, then the token at the input channel specified in the last tick will be send to the output port; while if the *control* input port has never received any available token up to the current tick, then the input token at channel0 will be output. And if the value of the most recently received token on the *control* input port is less than zero or greater than or equal to the width of the input, then this actor will not fire() and output absent.

Let the variables $rIn_1, rIn_2, ..., rIn_m$ denote the inputs of every channel at the left *input* port, and let the variables $rCtrl$ and $rOut$ denote the *control* input and the output respectively. Suppose the variables $lastCtrl$ and s are introduced to record respectively, the last non-absent input (initially, $lastCtrl[0] = 0$), and the most recently received non-absent input at the *control* input port, and then we can encode the behavior of this actor as:

$$
\begin{aligned}
&T_{Select}[i] \\
&\triangleq \Bigg\{ \Bigg[\textit{is-value}(rCtrl[i]) \wedge s[i] = rCtrl[i] \Bigg] \vee \Bigg[rCtrl[i] = \texttt{absent} \wedge s[i] = lastCtrl \\
&\quad [i-1] \Bigg] \Bigg\} \wedge \Bigg\{ lastCtrl[i] = s[i] \Bigg\}
\end{aligned}
$$

$$\wedge\left\{\bigvee_{j=1}^{m}\left[s[j]=j-1\wedge\left(\left(\textit{is-value}(rIn_j[i])\wedge rOut[i]=rIn_j[i]\right)\vee\left(rIn_j[i]=\right.\right.\right.\right.$$

$$\left.\left.\left.\left.\texttt{absent}\wedge rOut[i]=\texttt{absent}\right)\right)\right]\vee\left[\left(s[i]<0\vee s[i]\geq m\right)\wedge rOut[i]=\texttt{absent}\right]\right\}$$

- **BooleanMultiplexor** outputs one input token specified by the most recently received *select* input and discards all other input tokens [1]. Similar to **Select** actor, in an iteration, if the *select* input port has no available token, then the token at the same input channel as that in the previous tick will be output. But if the *select* input port has never received any available token up to the current considering tick, then this actor outputs `absent`.

Let the variables $rtIn$, $rfIn$, $rSec$ and $rOut$ denote respectively, the relations connected to the ports *trueInput*, $falseInput$, *select*, and *output* of this actor. Suppose the variable *lastSec* is introduced to record the last non-absent input at the *select* input port if there is, otherwise, it records *absent* (initially, we set $lastSelect[0] = absent$). Similarly, the variable s is used to record the most recently received non-absent input at the *select* input port, otherwise, it records `absent`. Based on these variables, we can encode this actor as follows:

$$T_{BooleanMultiplexor}[i]$$

$$\triangleq\left[\left(\textit{is-value}(rSec[i])\wedge s[i]=rSec[i]\right)\vee\left(rSec[i]=\texttt{absent}\wedge s[i]=lastSec\right.\right.$$

$$\left.\left.[i-1]\right)\right]\wedge\left[lastSec[i]=s[i]\right]$$

$$\wedge\left[\left(s[i]=true\wedge\left(\left(\textit{is-value}(rtIn[i])\wedge rOut[i]=rtIn[i]\right)\vee\left(rtIn[i]=\texttt{absent}\right.\right.\right.\right.$$

$$\left.\left.\left.\wedge\, rOut[i]=\texttt{absent}\right)\right)\right)$$

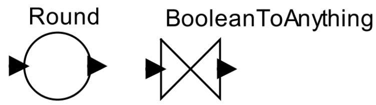
$$\vee\left(s[i]=false\wedge\left(\left(\textit{is-value}(rfIn[i])\wedge rOut[i]=rfIn[i]\right)\vee\left(rfIn[i]=\texttt{absent}\right.\right.\right.$$

$$\left.\left.\left.\wedge\, rOut[i]=\texttt{absent}\right)\right)\right)$$

$$\left.\vee\left(s[i]=\texttt{absent}\wedge rOut[i]=\texttt{absent}\right)\right]$$

F. Conversions Library

The following two actors **Round** and **BooleanToAnything** are the main actors we considered in the library `Actors`$\rightarrow$`Conversions`.



- **Round** converts an input token of double type into an int token, and the implemented functions are cell (towards positive infinity), floor (towards negative infinity), round (towards nearest integer) and truncate (towards zero) [1]. Let the variables rIn and $rOut$ be the relations connected to the input and output ports respectively, and let *to_int* be the function that maps each real number r to the largest integer x less than or equal to r, then we can encode this actor as follows.

1) If the specified function is "ceil", then we obtain that

$$T_{Round}[i] \triangleq \Bigg[\textit{is-value}(rIn[i]) \wedge \bigg(\Big(rIn[i] = \textit{to\_int}(rIn[i]) \wedge rOut[i] = rIn[i] \Big) \vee \Big(rIn[i] \neq \textit{to\_int}(rIn[i]) \wedge rOut[i] = \textit{to\_int}(rIn[i]) + 1 \Big) \bigg) \Bigg] \vee \Bigg[rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Bigg]$$

2) If the specified function is "floor", then we obtain that

$$T_{Round}[i] \triangleq \Bigg(\textit{is-value}(rIn[i]) \wedge rOut[i] = \textit{to\_int}(rIn[i]) \Bigg) \vee \Bigg(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Bigg)$$

3) If the specified function is "round", then we obtain that

$$T_{Round}[i] \triangleq \Bigg[\textit{is-value}(rIn[i]) \wedge \bigg(\Big(rIn[i] = \textit{to\_int}(rIn[i]) \wedge rOut[i] = rIn[i] \Big) \vee \Big(rIn[i] \neq \textit{to\_int}(rIn[i]) \wedge rIn[i] - \textit{to\_int}(rIn[i]) < 0.5 \wedge rOut[i] = \textit{to\_int}(rIn[i]) \Big) \vee \Big(rIn[i] \neq \textit{to\_int}(rIn[i]) \wedge rIn[i] - \textit{to\_int}(rIn[i]) \geq 0.5 \wedge rOut[i] = \textit{to\_int}(rIn[i]) + 1 \Big) \bigg) \Bigg] \vee \Bigg[rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Bigg]$$

4) If the specified function is "truncate", then we obtain that

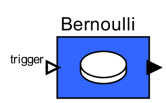
$$T_{Round}[i] \triangleq \Bigg[\textit{is-value}(rIn[i]) \wedge \Bigg(\Big(rIn[i] = to\_int(rIn[i]) \wedge rOut[i] = rIn[i] \Big) \vee$$
$$\Big(rIn[i] \neq to\_int(rIn[i]) \wedge rIn[i] < 0 \wedge rOut[i] = to\_int(rIn[i]) + 1 \Big)$$
$$\vee \Big(rIn[i] \neq to\_int(rIn[i]) \wedge rIn[i] \geq 0 \wedge rOut[i] = to\_int(rIn[i]) \Big) \Bigg) \Bigg]$$
$$\vee \Big[rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Big]$$

- **BooleanToAnything** converts a boolean input into a value of any type [1]. Specifically, when a true token is arriving at the input port, this actor outputs the value given by the $trueValue$ parameter; when a false token is arriving at the input port, this actor outputs the value given by the $falseValue$ parameter; however, if no input token is available, this actor outputs `absent`. Suppose the variables rIn and $rOut$ are introduced to denote the input and the output of this actor respectively, then this actor can be encoded as:

$$T_{BooleanToAnything}[i]$$
$$\triangleq \Bigg(\textit{is-value}(rIn[i]) \wedge \Big(\big(rIn[i] = true \wedge rOut[i] = trueValue \big) \vee \big(rIn[i] = false$$
$$\wedge\, rOut[i] = falseValue \big) \Big) \Bigg) \vee \Big(rIn[i] = \texttt{absent} \wedge rOut[i] = \texttt{absent} \Big)$$

G. Random Library

In the library `Actors`→`Conversions`, the most commonly applied actor in an SR model is **Bernoulli**. This actor produces true or false value randomly with some fixed probability set by the $trueProbability$ parameter.



Let the variable $rOut$ denote the output of this actor, and then we can encode this actor as

$$T_{Bernoulli}[i] \triangleq rOut[i] = true \vee rOut[i] = false.$$

Here, we don't consider the value of the $trueProbability$ parameter, however, this way of encoding is still correct because it contains all possibility that this parameter can be evaluated to. For example, suppose the false output of this actor can result in the unsafety of a considering system and the value of the $trueProbability$ parameter is 0.999, then we can indicate that the system in consideration is unsafe only in a 0.001 percent chance. Even so, our encoding for this system should contain this possibility, and therefore, we encode the Bernoulli actor as above.

References

1. Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II.* Ptolemy.org, 2014.