section *Stateflow2Circus* **parents** *StateflowAbstractSyntax*, *CircusFormalSyntax*

$$
\begin{array}{l}
chartname, chartid : WF\_CHART \rightarrowtail NAME \\
statename : WF\_STATE \rightarrowtail NAME \\
stateid : WF\_STATE \rightarrowtail NAME \\
junctionname, junctionid : WF\_JUNCTION \rightarrowtail NAME \\
transitionname, transitionid : WF\_TRANSITION \rightarrowtail NAME \\
dataname : WF\_DATA \rightarrowtail NAME \\
eventname : WF\_EVENT \rightarrowtail NAME \\
eventcountername : WF\_EVENT \rightarrowtail NAME \\
datachannelname : WF\_DATA \rightarrowtail NAME \\
outputeventchannelname : WF\_EVENT \rightarrowtail NAME \\
processname, processstatename : WF\_CHART \rightarrowtail NAME \\
DuringActionName : WF\_STATE \rightarrowtail NAME \\
EntryActionName : WF\_STATE \rightarrowtail NAME \\
ExitActionName : WF\_STATE \rightarrowtail NAME \\
ConditionActionName : WF\_TRANSITION \rightarrowtail NAME \\
TransitionActionName : WF\_TRANSITION \rightarrowtail NAME \\
ConditionName : WF\_TRANSITION \rightarrowtail NAME \\
TriggerName : WF\_TRANSITION \rightarrowtail NAME \\
namesets : \mathrm{seq}(\mathbb{P}\, NAME)
\end{array}
$$

$$
\begin{array}{l}
namesets = \langle \mathrm{ran}\, chartname, \mathrm{ran}\, chartid, \mathrm{ran}\, statename, \mathrm{ran}\, stateid, \\
\quad \mathrm{ran}\, junctionname, \mathrm{ran}\, junctionid, \mathrm{ran}\, transitionname, \mathrm{ran}\, transitionid, \\
\quad \mathrm{ran}\, dataname, \mathrm{ran}\, eventname, \mathrm{ran}\, datachannelname, \mathrm{ran}\, processname, \\
\quad \mathrm{ran}\, processstatename, \mathrm{ran}\, DuringActionName, \mathrm{ran}\, EntryActionName, \\
\quad \mathrm{ran}\, ExitActionName, \mathrm{ran}\, ConditionActionName, \mathrm{ran}\, TransitionActionName, \\
\quad \mathrm{ran}\, ConditionName, \mathrm{ran}\, TriggerName, \mathrm{ran}\, eventcountername \rangle \\
\mathrm{disjoint}\, namesets
\end{array}
$$

$$
\begin{array}{l}
getdatabyname : WF\_CHART \times DNAME \pfun Data \\
geteventbyname : WF\_CHART \times ENAME \pfun Event \\
gettransitionbyid : WF\_CHART \times TID \pfun Transition \\
getstatebyid : WF\_CHART \times SID \pfun State \\
getchartbyid : WF\_CHART \times CID \pfun WF\_CHART \\
getjunctionbyid : WF\_CHART \times JID \pfun Junction
\end{array}
$$

$STATESTATUS : NAME$

$EXECUTEDURINGACTION, DURINGACTIONS : N$

$EXECUTEENTRYACTION, X, ENTRYACTIONS : N$

$EXECUTEEXITACTION, EXITACTIONS : N$

$EXECUTECONDITIONACTION, CONDITIONACTIONS, O : N$

$EXECUTETRANSITIONACTION, TRANSITIONACTIONS : N$

$EVALUATECONDITION, CONDITIONSACTION : N$

$CHECKTRIGGERCHANNEL, TRIGGERSACTION, RESULTCHANNEL : N$

$GETSTATE, GETTRANSITION, GETJUNCTION, GETCHART, GETEVENTS : NAME$
$STATUSACTION, HISTORYACTION, STATEHISTORY : NAME$
$EVENTSCHANNEL, STATECHANNEL, JUNCTIONCHANNEL : NAME$
$CHARTCHANNEL, TRANSITIONCHANNEL : NAME$
$STATUSCHANNEL, HISTORYCHANNEL : NAME$

$ACTIVATIONACTION, ACTIVATECHANNEL, ACTIVATESCHEMA : NAME$
$DEACTIVATIONACTION, DEACTIVATECHANNEL, DEACTIVATESCHEMA : NAME$
$CHARTACTIONS, INTERFACEACTIONS, INPUTSACTION, OUTPUTSACTION : NAME$
$READINPUTSCHANNEL, WRITEOUTPUTSCHANNEL, ACTIONVARIABLE : NAME$

$STATE, STATEID, TRANSITION, TRANSITIONID, JUNCTION, JUNCTIONID : NAME$
$EVENT, UNIVERSE, REAL, SFBOOL : NAME$
$IDENTIFIER, DEFAULT, INNER, OUTER, PARENT, TRANS : NAME$
$LEFT, RIGHT, SUBSTATES, DECOMPOSITION, TYPE, HISTORY : NAME$
$CLUSTER, SET, CHART, AND, OR : NAME$
$SOURCE, DESTINATION, NEXT, SNODE, JNODE : NAME$
$NULLSTATE, NULLTRANSITION, NULLJUNCTION, NULLEVENT : NAME$
$NULLTRANSITIONID, NULLSTATEID : NAME$

$SimulationData, InitSimulationData : NAME$
$SimulationInstance, InitSimulationInstance, InitState : NAME$

$STATES, TRANSITIONS, JUNCTIONS, STATEFLOWCHART : N$

$BROADCAST, CHECK, ORIGIN, DEST, ALLACTIONS : NAME$
$ENDLOCALEXECUTION, ENDACTION : NAME$
$I, J, E, S, B : NAME$

$[X]$

$$set2seq : \mathbb{P}\, X \to \operatorname{seq} X$$

$$\forall\, s : \mathbb{P}\, X \bullet \operatorname{ran}(set2seq(s)) = s$$

$binarysfop : \mathbb{P}(EXPR \times EXPR \rightarrowtail EXPR)$

$binarysfop = \{or, and, bor, band, bxor, neq, eq, leq, geq, lt, gt, lshift, rshift, sum, sub,$
$modulus, mult, division\}$

$$unarysfop : \mathbb{P}(EXPR \rightarrowtail EXPR)$$

$$unarysfop = \{not, bnot, neg\}$$

$$binaryop : binarysfop \nrightarrow (Expression \times Expression \rightarrow Expression)$$
$$unaryop : unarysfop \nrightarrow (Expression \rightarrow Expression)$$

$$translate_{VectorDomain} : (\mathbb{N} \times \operatorname{seq} \mathbb{N}) \rightarrow Expression$$

$$\forall\, n : \mathbb{N};\ s : \operatorname{seq} \mathbb{N} \bullet translate_{VectorDomain}(n, s) =$$
$$\left(\begin{array}{l} \textbf{if}\ \ s = \langle\rangle \\ \textbf{then}\ functionapplication(app(UPTO, \langle numberliteral(0), numberliteral(n-1)\rangle)) \\ \textbf{else}\ product(\\ \qquad functionapplication(app(UPTO, \langle numberliteral(0), numberliteral(n-1)\rangle)), \\ \qquad translate_{VectorDomain}(head\ s, tail\ s) \\ \quad) \end{array}\right)$$

$$translate_{Type} : DATATYPE \rightarrow Expression$$

$$translate_{Type}(scalar(INT32)) = reference(ref(INTEGER))$$
$$translate_{Type}(scalar(INT16)) = reference(ref(INTEGER))$$
$$translate_{Type}(scalar(INT8)) = reference(ref(INTEGER))$$
$$translate_{Type}(scalar(UINT32)) = reference(ref(NATURAL))$$
$$translate_{Type}(scalar(UINT16)) = reference(ref(NATURAL))$$
$$translate_{Type}(scalar(UINT8)) = reference(ref(NATURAL))$$
$$translate_{Type}(scalar(DOUBLE)) = reference(ref(REAL))$$
$$translate_{Type}(scalar(SINGLE)) = reference(ref(REAL))$$
$$translate_{Type}(scalar(BOOLEAN)) = reference(ref(SFBOOL))$$
$$\forall\, s : (\operatorname{seq}_1 \mathbb{N});\ t : SCALAR \bullet$$
$$translate_{Type}(vector(t, s)) = functionapplication(app(FUN, \langle$$
$$translate_{VectorDomain}(head\ s, tail\ s),$$
$$translate_{Type}(scalar(t))$$
$$\rangle))$$

$translate_{Expr} : WF\_EXPR \to Expression$
$translate_{Exprs} : WF\_CHART \times \operatorname{seq} EXPR \nrightarrow \operatorname{seq} Expression$

$\forall\, op : \operatorname{dom} unaryop;\ e : EXPR;\ c : WF\_CHART \mid (c, op(e)) \in WF\_EXPR \bullet$
$\quad translate_{Expr}(c, op(e)) = unaryop(op)(translate_{Expr}(c, e))$
$\forall\, op : \operatorname{dom} binaryop;\ e_1, e_2 : EXPR;\ c : WF\_CHART \mid (c, op(e_1, e_2)) \in WF\_EXPR \bullet$
$\quad translate_{Expr}(c, op(e_1, e_2)) = binaryop(op)(translate_{Expr}(c, e_1), translate_{Expr}(c, e_2))$
$\forall\, n : DNAME;\ es : \operatorname{seq} EXPR;\ c : WF\_CHART \mid (c, name(n, es)) \in WF\_EXPR \bullet$
$translate_{Expr}(c, name(n, es)) =$
$\left(\begin{array}{l} \textbf{if } \#\, es = 0 \\ \textbf{then } reference(ref(dataname(c, getdatabyname(c, n)))) \\ \textbf{else } functionapplication(app(dataname(c, getdatabyname(c, n)), translate_{Exprs}(c, es))) \end{array}\right)$
$\forall\, n : NAME;\ es : \operatorname{seq} EXPR;\ c : WF\_CHART \mid (c, fun(n, es)) \in WF\_EXPR \bullet$
$translate_{Expr}(c, fun(n, es)) = functionapplication(app(n, translate_{Exprs}(c, es)))$
$\forall\, v : \mathbb{A};\ c : WF\_CHART \mid (c, value(v)) \in WF\_EXPR \bullet$
$\quad translate_{Expr}(c, value(v)) = numberliteral(v)$
$\forall\, s : \operatorname{seq}_1 \mathbb{A};\ c : WF\_CHART \mid (c, array(s)) \in WF\_EXPR \bullet$
$\quad translate_{Expr}(c, array(s)) =$
$\quad\quad \textbf{let } pairs == \{i : 0 \,..\, \#\, s - 1 \bullet$
$\quad\quad\quad functionapplication(app(MAPSTO, \langle numberliteral(i), numberliteral(s(i+1))\rangle))\}$
$\quad\quad \bullet\ setextension(set2seq(pairs))$
$\forall\, s : \operatorname{seq}_1 \operatorname{seq}_1 \mathbb{A};\ c : WF\_CHART \mid (c, matrix(s)) \in WF\_EXPR \bullet$
$\quad translate_{Expr}(c, matrix(s)) =$
$\quad\quad \textbf{let } pairs == \{i : 0 \,..\, \#\, s - 1;\ j : 0 \,..\, \#\, s(1) - 1 \bullet$
$\quad\quad\quad functionapplication(app(MAPSTO, \langle$
$\quad\quad\quad\quad tupleextension(\langle numberliteral(i), numberliteral(j)\rangle),$
$\quad\quad\quad\quad numberliteral(s(i+1)(j+1))\rangle))\}$
$\quad\quad \bullet\ setextension(set2seq(pairs))$
$\forall\, n : SNAME;\ s : State;\ c : WF\_CHART \mid s.name = n \wedge (c, s) \in WF\_STATE \bullet$
$\quad translate_{Expr}(c, in(n)) =$
$\quad\quad functionapplication(app(STATESTATUS, \langle reference(ref(stateid(c, s)))\rangle))$
$\forall\, c : WF\_CHART \bullet translate_{Exprs}(c, \langle\rangle) = \langle\rangle$
$\forall\, es : \operatorname{seq} EXPR;\ c : WF\_CHART \mid \#\, es > 0 \wedge (\forall\, e : \operatorname{ran} es \bullet (c, e) \in WF\_EXPR) \bullet$
$\quad translate_{Exprs}(c, es) = \langle translate_{Expr}(c, head\ es)\rangle \frown translate_{Exprs}(c, tail\ es)$

$$translate_{InitExpr} : WF\_DATA \to Expression$$

$$\forall\, d : Data;\ c : WF\_CHART \mid (c, d) \in WF\_DATA \bullet translate_{InitExpr}(c, d) =$$
$$\left(\begin{array}{l} \textbf{if } d.type \in \operatorname{ran} scalar \\ \textbf{then } translate_{Expr}(c, d.initial) \\ \textbf{else } \left(\begin{array}{l} \textbf{if } d.initial \in \operatorname{ran} value \\ \textbf{then let } decl == declpart(\langle variable(\langle decl(X)\rangle, \\ \qquad translate_{VectorDomain}(head\ ((vector\sim)(d.type)).2, tail\ ((vector\sim)(d.type)).2))\rangle) \\ \qquad \bullet\ functionconstruction(schematext(\langle decl\rangle, \langle\rangle), translate_{Expr}(c, d.initial)) \\ \textbf{else } translate_{Expr}(c, d.initial) \end{array}\right) \end{array}\right)$$

$$translate_{Var} : Action \to Action$$

$$\forall\, a : Action \bullet translate_{Var}(a) = \textbf{let}$$
$$decl == declpart(\langle variable(\langle decl(B)\rangle, reference(ref(BOOL)))\rangle) \bullet$$
$$commandaction(variableblock(decl, a))$$

$$translate_{Interrupt} : Action \to Action$$

$$\forall\, a : Action \bullet translate_{Interrupt}(a) =$$
$$\textbf{let } pred == relationapplication(rel(EQUAL, \langle reference(ref(B)), reference(ref(T))\rangle)) \bullet$$
$$\textbf{let } conditions == gactions(\langle (pred, cspaction(skip)), (negation(pred), a)\rangle) \bullet$$
$$commandaction(ifcommand(conditions))$$

$$translate_{Check} : Action$$

$$translate_{Check} =$$
$$\textbf{let } pars == \langle reference(ref(B))\rangle$$
$$\bullet\ cspaction(paractioninstantiation(paraction(namedaction(CHECK)), pars))$$

$$translate_{Broadcast} : ENAME \times (SID \cup CID) \times WF\_CHART \to Action$$

$$\forall\, n : ENAME;\ d : (SID \cup CID);\ c : WF\_CHART \bullet translate_{Broadcast}(n, d, c) =$$
$$\left(\begin{array}{l} \textbf{let}\ dest == \left(\begin{array}{l} \textbf{if}\ d \in CID \\ \textbf{then}\ chartid(c) \\ \textbf{else}\ stateid(c, getstatebyid(c, d)) \end{array} \right) \bullet \\ \textbf{let}\ pars == \langle reference(ref(eventname(c, geteventbyname(c, n)))), reference(ref(dest)) \rangle \\ \bullet\ cspaction(paractioninstantiation(paraction(namedaction(BROADCAST)), pars)) \end{array} \right)$$

$translate_{Act} : WF\_ACTION \to Action$

$\forall\, n : DNAME;\ es : \mathrm{seq}\, EXPR;\ e : EXPR;\ c : WF\_CHART \mid (c, assign(n, es, e)) \in WF\_ACTION \bullet$
$translate_{Act}(c, assign(n, es, e)) =$

$$
\left(\begin{array}{l}
\textbf{if } \#\, es = 0 \\
\textbf{then } commandaction(assignment(\langle(dataname(c, getdatabyname(c, n)), translate_{Expr}(c, e))\rangle)) \\
\textbf{else} \\
\left(\begin{array}{l}
\textbf{if } \#\, es = 1 \\
\textbf{then } commandaction(assignment(\langle \\
\quad (dataname(c, getdatabyname(c, n)), functionapplication(app(OVERRIDE, \langle \\
\quad\quad reference(ref(dataname(c, getdatabyname(c, n)))), setextension(\langle \\
\quad\quad\quad tupleextension(\langle translate_{Expr}(c, head\ es), translate_{Expr}(c, e)\rangle) \\
\quad\quad \rangle) \\
\quad \rangle))) \\
\rangle)) \\
\textbf{else } commandaction(assignment(\langle \\
\quad (dataname(c, getdatabyname(c, n)), functionapplication(app(OVERRIDE, \langle \\
\quad\quad reference(ref(dataname(c, getdatabyname(c, n)))), setextension(\langle \\
\quad\quad\quad tupleextension(\langle tupleextension(translate_{Exprs}(c, es)), translate_{Expr}(c, e)\rangle) \\
\quad\quad \rangle) \\
\quad \rangle))) \\
\rangle))
\end{array}\right)
\end{array}\right)
$$

$\forall\, n : ENAME;\ d : (SID \cup CID);\ c : WF\_CHART \mid (c, bcast(n, d)) \in WF\_ACTION \bullet$
$translate_{Act}(c, bcast(n, d)) =$

$$
\left(\begin{array}{l}
\textbf{if } (geteventbyname(c, n)).scope = OUTPUTEVENT \\
\textbf{then } commandaction(\\
\quad assignment(\langle \\
\quad\quad (eventcountername(c, geteventbyname(c, n)), \\
\quad\quad functionapplication(app(PLUS, \\
\quad\quad\quad \langle reference(ref(eventcountername(c, geteventbyname(c, n)))), numberliteral(1)\rangle))) \\
\quad \rangle) \\
) \\
\textbf{else} cspaction(actionsequentialcomposition(translate_{Broadcast}(n, d, c), translate_{Check}))
\end{array}\right)
$$

$\forall\, e : EXPR;\ c : WF\_CHART \mid (c, expr(e)) \in WF\_ACTION \bullet$
$translate_{Act}(c, expr(e)) = cspaction(skip)$

$$actions\_have\_broadcast : \mathbb{P}(WF\_CHART \times \operatorname{seq} ACTION)$$

$$\forall c : WF\_CHART;\ as : \operatorname{seq} ACTION \bullet (c, as) \in actions\_have\_broadcast \Leftrightarrow \exists a : \operatorname{ran} as \bullet (a \in \operatorname{ran} bcast \land (geteventbyname(c, ((bcast^{\sim})(a)).1)).scope = LOCALEVENT)$$

$$daction\_has\_broadcast : \mathbb{P}(WF\_CHART \times DACTION)$$

$$\forall c : WF\_CHART;\ a : DACTION \bullet (c, a) \in daction\_has\_broadcast \Leftrightarrow (a \in \operatorname{ran} action \land (c, ((action^{\sim})a)) \in actions\_have\_broadcast) \lor (a \in \operatorname{ran} on \land (c, (((on^{\sim})a).2)) \in actions\_have\_broadcast)$$

$$dactions\_have\_broadcast : \mathbb{P}(WF\_CHART \times \operatorname{seq} DACTION)$$

$$\forall c : WF\_CHART;\ as : \operatorname{seq} DACTION \bullet (c, as) \in dactions\_have\_broadcast \Leftrightarrow \exists a : \operatorname{ran} as \bullet ((c, a) \in daction\_has\_broadcast)$$

$$translate_{Acts} : WF\_CHART \times \operatorname{seq} ACTION \nrightarrow Action$$

$$\forall as : \operatorname{seq} ACTION;\ c : WF\_CHART \mid \# as = 0 \bullet translate_{Acts}(c, as) = cspaction(skip)$$

$$\forall as : \operatorname{seq} ACTION;\ c : WF\_CHART \mid \# as \geq 1 \land (\forall a : \operatorname{ran} as \bullet (c, a) \in WF\_ACTION) \bullet translate_{Acts}(c, as) = \left(\begin{array}{l} \textbf{if}((head\ as) \in \operatorname{ran} bcast) \land ((geteventbyname(c, ((bcast^{\sim})(head\ as)).1)).scope = LOCALEVENT) \\ \textbf{then}\ cspaction(actionsequentialcomposition(translate_{Act}(c, head\ as), translate_{Interrupt}(translate_{Acts}(c, tail\ as)))) \\ \textbf{else}\ cspaction(actionsequentialcomposition(translate_{Act}(c, head\ as), translate_{Acts}(c, tail\ as))) \end{array} \right)$$

$$translate_{ONAct} : WF\_ONACTION \rightarrow Action$$

$$\forall n : ENAME;\ as : \operatorname{seq} ACTION;\ c : WF\_CHART \mid (c, (n, as)) \in WF\_ONACTION \bullet translate_{ONAct}(c, (n, as)) = commandaction(ifcommand(gactions(\langle (relationapplication(rel(EQUAL, \langle reference(ref(E)), reference(ref(eventname(c, geteventbyname(c, n))))\rangle)), translate_{Acts}(c, as)), (negation(relationapplication(rel(EQUAL, \langle reference(ref(E)), reference(ref(eventname(c, geteventbyname(c, n))))\rangle))), cspaction(skip)) \rangle)))$$

$$translate_{DAct} : WF\_DACTION \rightarrow Action$$

$$\forall as : \operatorname{seq} ACTION;\ c : WF\_CHART \mid \forall a : \operatorname{ran} as \bullet (c, a) \in WF\_ACTION \bullet translate_{DAct}(c, action(as)) = translate_{Acts}(c, as)$$

$$\forall a : ONACTION;\ c : WF\_CHART \mid (c, a) \in WF\_ONACTION \bullet translate_{DAct}(c, on(a)) = translate_{ONAct}(c, a)$$

$translate_{DActs} : WF\_CHART \times \operatorname{seq} DACTION \pfun Action$

$\forall as : \operatorname{seq} DACTION;\ c : WF\_CHART \mid \# as = 0 \bullet translate_{DActs}(c, as) = cspaction(skip)$

$\forall as : \operatorname{seq} DACTION;\ c : WF\_CHART \mid \# as \geq 1 \land (\forall a : \operatorname{ran} as \bullet (c, a) \in WF\_DACTION) \bullet translate_{DActs}(c, as) =$

$$\left(\begin{array}{l} \textbf{if}(c, head\ as) \in daction\_has\_broadcast \\ \textbf{then} cspaction(actionsequentialcomposition(translate_{DAct}(c, head\ as), translate_{Interrupt}(translate_{DActs}(c, tail\ as)))) \\ \textbf{else} cspaction(actionsequentialcomposition(translate_{DAct}(c, head\ as), translate_{DActs}(c, tail\ as))) \end{array} \right)$$

$tv : Expression \to Predicate$

$\forall e : Expression \bullet tv(e) = negation(relationapplication(rel(EQUAL, \langle e, numberliteral(0)\rangle)))$

$translate_{Pred} : WF\_EXPR \to Predicate$

$\forall e : EXPR;\ c : WF\_CHART \mid (c, e) \in WF\_EXPR \bullet translate_{Pred}(c, e) = tv(translate_{Expr}(c, e))$

$translate_{Trigger} : WF\_TRIGGER \to Predicate$

$\forall n : ENAME;\ c : WF\_CHART \mid (c, e(n)) \in WF\_TRIGGER \bullet translate_{Trigger}(c, e(n)) =$
$relationapplication(rel(EQUAL, \langle reference(ref(E)), reference(ref(eventname(c, geteventbyname(c, n))))\rangle))$

$\forall texp : TEXPR;\ c : WF\_CHART \mid (c, t(texp)) \in WF\_TRIGGER \bullet translate_{Trigger}(c, t(texp)) =$
$translate_{Pred}(c, texp)$

$translate_{Triggers} : WF\_CHART \times \operatorname{seq}_1 TRIGGER \pfun Predicate$

$\forall ts : \operatorname{seq}_1 TRIGGER;\ c : WF\_CHART \mid \# ts = 1 \land (c, head\ ts) \in WF\_TRIGGER \bullet$
$translate_{Triggers}(c, ts) = translate_{Trigger}(c, head\ ts)$

$\forall ts : \operatorname{seq}_1 TRIGGER;\ c : WF\_CHART \mid \# ts > 1 \land \forall t : \operatorname{ran} ts \bullet (c, t) \in WF\_TRIGGER \bullet$
$translate_{Triggers}(c, ts) = disjunction(translate_{Trigger}(c, head\ ts), translate_{Triggers}(c, tail\ ts))$

$$StateIdentifierDeclaration : WF\_CHART \to Paragraph$$

$$\begin{array}{l}
\forall\, c : WF_CHART \bullet StateIdentifierDeclaration(c) = \textbf{let} \\
\quad snames == set2seq(\{s : \operatorname{ran} c.states \bullet decl(stateid(c, s))\}); \\
\quad cname == \langle decl(chartid(c))\rangle; \\
\quad exprs == set2seq(\{s : \operatorname{ran} c.states \bullet reference(ref(stateid(c, s)))\}) \frown \langle reference(ref(chartid(c)))\rangle; \\
\quad type == reference(ref(STATEID)) \bullet \textbf{let} \\
\qquad decl == declpart(\langle variable(snames \frown cname, type)\rangle); \\
\qquad pred == relationapplication(rel(EQUAL, \langle reference(ref(STATEID)), setextension(exprs)\rangle)) \bullet \\
\qquad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))
\end{array}$$

$$state\_identifier : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, s : State;\ c : WF\_CHART \mid (c, s) \in WF\_STATE \bullet state\_identifier(c, s) = (decl(IDENTIFIER), reference(ref(stateid(c, s))))$$

$$state\_default : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF_CHART;\ s : State \mid (c, s) \in WF_STATE \bullet state_default(c, s) = \left(\begin{array}{l}
\textbf{if}\ \#s.default = 0 \\
\textbf{then}\,(decl(DEFAULT), \\
\qquad reference(ref(NULLTRANSITIONID))) \\
\textbf{else}\,(decl(DEFAULT), \\
\qquad reference(ref(transitionid(c, gettransitionbyid(c, head\,(s.default))))))
\end{array}\right)$$

$$state\_inner : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF_CHART;\ s : State \mid (c, s) \in WF_STATE \bullet state_inner(c, s) = \left(\begin{array}{l}
\textbf{if}\ \#s.inner = 0 \\
\textbf{then}\,(decl(INNER), \\
\qquad reference(ref(NULLTRANSITIONID))) \\
\textbf{else}\,(decl(INNER), \\
\qquad reference(ref(transitionid(c, gettransitionbyid(c, head\,(s.inner))))))
\end{array}\right)$$

$$state\_outer : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_outer(c, s) =$$
$$\left(\begin{array}{l} \textbf{if } \#\, s.outer = 0 \\ \textbf{then } (decl(OUTER), \\ \qquad reference(ref\,(NULLTRANSITIONID))) \\ \textbf{else } (decl(OUTER), \\ \qquad reference(ref\,(transitionid(c, gettransitionbyid(c, head\,(s.outer)))))) \end{array}\right)$$

$$state\_parent : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_parent(c, s) =$$
$$\textbf{if } s.parent \in \operatorname{dom} c.states$$
$$\textbf{then } (decl(PARENT), reference(ref\,(stateid(c, getstatebyid(c, s.parent)))))$$
$$\textbf{else } (decl(PARENT), reference(ref\,(chartid(c))))$$

$$state\_left : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_left(c, s) =$$
$$\left(\begin{array}{l} \textbf{if } \#\, s.left = 0 \\ \textbf{then } (decl(LEFT), reference(ref\,(NULLSTATEID))) \\ \textbf{else } (decl(LEFT), reference(ref\,(stateid(c, getstatebyid(c, head(s.left)))))) \end{array}\right)$$

$$state\_right : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_right(c, s) =$$
$$\left(\begin{array}{l} \textbf{if } \#\, s.right = 0 \\ \textbf{then } (decl(RIGHT), reference(ref\,(NULLSTATEID))) \\ \textbf{else } (decl(RIGHT), reference(ref\,(stateid(c, getstatebyid(c, head(s.right)))))) \end{array}\right)$$

$$state\_substates : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_substates(c, s) =$$
$$(decl(SUBSTATES), sequence(\{i : \operatorname{dom} s.substates \bullet i \mapsto reference(ref\,(stateid(c, getstatebyid(c, (s.substates)(i)))))\}))$$

$$state\_decomposition : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_decomposition(c, s) =$$
$$\quad \textbf{if}\ s.decomp = SEQ$$
$$\quad \textbf{then}\ (decl(DECOMPOSITION), reference(ref(CLUSTER)))$$
$$\quad \textbf{else}\ (decl(DECOMPOSITION), reference(ref(SET)))$$

$$state\_type : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_type(c, s) =$$
$$\quad \textbf{if}\ s.type = PAR$$
$$\quad \textbf{then}\ (decl(TYPE), reference(ref(AND)))$$
$$\quad \textbf{else}\ (decl(TYPE), reference(ref(OR)))$$

$$state\_history : WF\_STATE \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet state\_history(c, s) =$$
$$\quad \textbf{if}\ s.history = \textbf{True}$$
$$\quad \textbf{then}\ (decl(HISTORY), reference(ref(T)))$$
$$\quad \textbf{else}\ (decl(HISTORY), reference(ref(F)))$$

$$StateDeclaration : WF\_STATE \rightarrow Paragraph$$

$$\forall c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet StateDeclaration(c, s) = \textbf{let}$$
$$name == \langle decl(statename(c, s))\rangle;$$
$$type == reference(ref(STATE));$$
$$binding == bindingextension(\langle$$
$$state\_identifier(c, s),$$
$$state\_default(c, s),$$
$$state\_inner(c, s),$$
$$state\_outer(c, s),$$
$$state\_parent(c, s),$$
$$state\_left(c, s),$$
$$state\_right(c, s),$$
$$state\_substates(c, s),$$
$$state\_decomposition(c, s),$$
$$state\_type(c, s),$$
$$state\_history(c, s)$$
$$\rangle) \bullet \textbf{let}$$
$$decl == declpart(\langle variable(name, type)\rangle);$$
$$pred == relationapplication(rel(EQUAL, \langle reference(ref(statename(c, s))), binding\rangle)) \bullet$$
$$axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$$

$$StateSetDeclaration : WF\_CHART \rightarrow Paragraph$$

$$\forall c : WF\_CHART \bullet StateSetDeclaration(c) = \textbf{let}$$
$$names == \left(\begin{array}{l} set2seq(\{s : \operatorname{ran} c.states \bullet reference(ref(statename(c, s)))\} \cup \\ \{reference(ref(NULLSTATE)), reference(ref(chartname(c)))\}) \end{array} \right) \bullet \textbf{let}$$
$$pred == relationapplication(rel(EQUAL, \langle reference(ref(STATE)), setextension(names)\rangle)) \bullet$$
$$axiomaticdescription(schematext(\langle\rangle, \langle pred\rangle))$$

$$chart\_identifier : WF\_CHART \rightarrow (DeclName \times Expression)$$

$$\forall c : WF\_CHART \bullet chart\_identifier(c) =$$
$$(decl(IDENTIFIER), reference(ref(chartid(c))))$$

$$chart\_default : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_default(c) =$$
$$\left(\begin{array}{l} \textbf{if } \#\, c.default = 0 \\ \textbf{then } (decl(DEFAULT), \\ \qquad reference(ref(NULLTRANSITIONID))) \\ \textbf{else } (decl(DEFAULT), \\ \qquad reference(ref(transitionid(c, gettransitionbyid(c, head\,(c.default))))))) \end{array}\right)$$

$$chart\_inner : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_inner(c) =$$
$$(decl(INNER),$$
$$\quad reference(ref(NULLTRANSITIONID)))$$

$$chart\_outer : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_outer(c) =$$
$$(decl(OUTER),$$
$$\quad reference(ref(NULLTRANSITIONID)))$$

$$chart\_parent : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_parent(c) =$$
$$(decl(PARENT), reference(ref(NULLSTATEID)))$$

$$chart\_left : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_left(c) =$$
$$(decl(LEFT), reference(ref(NULLSTATEID)))$$

$$chart\_right : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_right(c) =$$
$$(decl(RIGHT), reference(ref(NULLSTATEID)))$$

$$chart\_substates : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_substates(c) = (decl(SUBSTATES), sequence(\{i : \operatorname{dom} c.substates \bullet i \mapsto reference(ref(stateid(c, getstatebyid(c, (c.substates)(i)))))\}))$$

$$chart\_decomposition : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_decomposition(c) = \textbf{if } c.decomp = SEQ \textbf{ then } (decl(DECOMPOSITION), reference(ref(CLUSTER))) \textbf{ else } (decl(DECOMPOSITION), reference(ref(SET)))$$

$$chart\_type : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_type(c) = (decl(TYPE), reference(ref(CHART)))$$

$$chart\_history : WF\_CHART \to (DeclName \times Expression)$$

$$\forall\, c : WF\_CHART \bullet chart\_history(c) = (decl(HISTORY), reference(ref(F)))$$

$ChartAsStateDeclaration : WF\_CHART \to Paragraph$

$\forall\, c : WF\_CHART \bullet ChartAsStateDeclaration(c) = \textbf{let}$
$\quad name == \langle decl(chartname(c))\rangle;$
$\quad type == reference(ref(STATE));$
$\quad binding == bindingextension(\langle$
$\quad\quad chart\_identifier(c),$
$\quad\quad chart\_default(c),$
$\quad\quad chart\_inner(c),$
$\quad\quad chart\_outer(c),$
$\quad\quad chart\_parent(c),$
$\quad\quad chart\_left(c),$
$\quad\quad chart\_right(c),$
$\quad\quad chart\_substates(c),$
$\quad\quad chart\_decomposition(c),$
$\quad\quad chart\_type(c),$
$\quad\quad chart\_history(c)$
$\quad\rangle) \bullet \textbf{let}$
$\quad\quad decl == declpart(\langle variable(name, type)\rangle);$
$\quad\quad pred == relationapplication(rel(EQUAL, \langle reference(ref(chartname(c))), binding\rangle)) \bullet$
$\quad\quad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$

$TransitionIdentifierDeclaration : WF\_CHART \to Paragraph$

$\forall\, c : WF\_CHART \bullet TransitionIdentifierDeclaration(c) = \textbf{let}$
$\quad tnames == set2seq(\{t : \operatorname{ran} c.transitions \bullet decl(transitionid(c, t))\});$
$\quad exprs == set2seq(\{t : \operatorname{ran} c.transitions \bullet reference(ref(transitionid(c, t)))\});$
$\quad type == reference(ref(TRANSITIONID)) \bullet \textbf{let}$
$\quad\quad decl == declpart(\langle variable(tnames, type)\rangle);$
$\quad\quad pred == relationapplication(rel(EQUAL, \langle reference(ref(TRANSITIONID)), setextension(exprs)\rangle)) \bullet$
$\quad\quad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$

$transition\_identifier : WF\_TRANSITION \to (DeclName \times Expression)$

$\forall\, c : WF\_CHART;\ t : Transition \mid (c, t) \in WF\_TRANSITION \bullet transition\_identifier(c, t) =$
$(decl(IDENTIFIER), (reference(ref(transitionid(c, t)))))$

$$transition\_source : WF\_TRANSITION \rightarrow (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ t : Transition \mid (c, t) \in WF_TRANSITION \bullet transition_source(c, t) = \\
\quad \textbf{if } \#\, t.source = 0 \\
\quad \textbf{then } (decl(SOURCE), (reference(ref(NULLTRANSITIONID)))) \\
\quad \textbf{else if } head\ t.source \in \mathrm{dom}\ c.states \\
\qquad \textbf{then } (decl(SOURCE), \\
\qquad\quad application(reference(ref(SNODE)), \\
\qquad\qquad reference(ref(stateid(c, getstatebyid(c, head\ t.source)))))) \\
\qquad \textbf{else } (decl(SOURCE), \\
\qquad\quad application(reference(ref(JNODE)), \\
\qquad\qquad reference(ref(junctionid(c, getjunctionbyid(c, head\ t.source))))))
\end{array}$$

$$transition\_destination : WF\_TRANSITION \rightarrow (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ t : Transition \mid (c, t) \in WF_TRANSITION \bullet transition_destination(c, t) = \\
\textbf{if } t.destination \in \mathrm{dom}\ c.states \\
\qquad \textbf{then } (decl(DESTINATION), \\
\qquad\quad application(reference(ref(SNODE)), \\
\qquad\qquad reference(ref(stateid(c, getstatebyid(c, t.destination)))))) \\
\qquad \textbf{else } (decl(DESTINATION), \\
\qquad\quad application(reference(ref(JNODE)), \\
\qquad\qquad reference(ref(junctionid(c, getjunctionbyid(c, t.destination))))))
\end{array}$$

$$transition\_next : WF\_TRANSITION \rightarrow (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ t : Transition \mid (c, t) \in WF_TRANSITION \bullet transition_next(c, t) = \\
\left(\begin{array}{l}
\textbf{if } \#\, t.next = 0 \\
\textbf{then } (decl(NEXT), reference(ref(NULLTRANSITIONID))) \\
\textbf{else } (decl(NEXT), reference(ref(transitionid(c, gettransitionbyid(c, head\ t.next)))))
\end{array} \right)
\end{array}$$

$$transition\_parent : WF\_TRANSITION \rightarrow (DeclName \times Expression)$$

$$\forall c : WF\_CHART;\ t : Transition \mid (c, t) \in WF\_TRANSITION \bullet transition\_parent(c, t) =$$
$$\quad \textbf{if}\ t.parent \in \operatorname{dom} c.states$$
$$\quad \textbf{then}\ (decl(PARENT), reference(ref(stateid(c, getstatebyid(c, t.parent)))))$$
$$\quad \textbf{else}\ (decl(PARENT), reference(ref(chartid(c))))$$

$$TransitionDeclaration : WF\_TRANSITION \rightarrow Paragraph$$

$$\forall c : WF\_CHART;\ t : Transition \mid (c, t) \in WF\_TRANSITION \bullet TransitionDeclaration(c, t) = \textbf{let}$$
$$\quad name == \langle decl(transitionname(c, t))\rangle;$$
$$\quad type == reference(ref(TRANSITION));$$
$$\quad binding == bindingextension(\langle$$
$$\qquad transition\_identifier(c, t),$$
$$\qquad transition\_source(c, t),$$
$$\qquad transition\_destination(c, t),$$
$$\qquad transition\_next(c, t),$$
$$\qquad transition\_parent(c, t) \quad \rangle) \bullet \textbf{let}$$
$$\qquad decl == declpart(\langle variable(name, type)\rangle);$$
$$\qquad pred == relationapplication(rel(EQUAL, \langle reference(ref(transitionname(c, t))), binding\rangle)) \bullet$$
$$\qquad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$$

$$TransitionSetDeclaration : WF\_CHART \rightarrow Paragraph$$

$$\forall c : WF\_CHART \bullet TransitionSetDeclaration(c) = \textbf{let}$$
$$\quad names == set2seq(\{t : \operatorname{ran} c.transitions \bullet reference(ref(transitionname(c, t)))\} \cup \{reference(ref(NULLTRANSITION))\}) \bullet \textbf{let}$$
$$\qquad pred == relationapplication(rel(EQUAL, \langle reference(ref(TRANSITION)), setextension(names)\rangle)) \bullet$$
$$\qquad\quad axiomaticdescription(schematext(\langle\rangle, \langle pred\rangle))$$

$$JunctionIdentifierDeclaration : WF\_CHART \to Paragraph$$

$$\begin{array}{l}
\forall\, c : WF_CHART \bullet JunctionIdentifierDeclaration(c) = \textbf{let} \\
\quad jnames == set2seq(\{j : \operatorname{ran} c.junctions \bullet decl(junctionid(c, j))\}); \\
\quad exprs == set2seq(\{j : \operatorname{ran} c.junctions \bullet reference(ref(junctionid(c, j)))\}); \\
\quad type == reference(ref(JUNCTIONID)) \bullet \textbf{let} \\
\qquad decl == declpart(\langle variable(jnames, type)\rangle); \\
\qquad pred == relationapplication(rel(EQUAL, \langle reference(ref(JUNCTIONID)), setextension(exprs)\rangle)) \bullet \\
\qquad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle\rangle))
\end{array}$$

$$junction\_identifier : WF\_JUNCTION \to (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ j : Junction \mid (c, j) \in WF_JUNCTION \bullet junction_identifier(c, j) = \\
(decl(IDENTIFIER), (reference(ref(junctionid(c, j)))))
\end{array}$$

$$junction\_transition : WF\_JUNCTION \to (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ j : Junction \mid (c, j) \in WF_JUNCTION \bullet junction_transition(c, j) = \\
\left(\begin{array}{l}
\textbf{if}\ \#j.transition = 0 \\
\textbf{then}\ (decl(TRANS), reference(ref(NULLTRANSITIONID))) \\
\textbf{else}\ (decl(TRANS), reference(ref(transitionid(c, gettransitionbyid(c, head\, j.transition)))))
\end{array}\right)
\end{array}$$

$$junction\_parent : WF\_JUNCTION \to (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ j : Junction \mid (c, j) \in WF_JUNCTION \bullet junction_parent(c, j) = \\
\quad \textbf{if}\ j.parent \in \operatorname{dom} c.states \\
\quad \textbf{then}\ (decl(PARENT), reference(ref(stateid(c, getstatebyid(c, j.parent)))))\\
\quad \textbf{else}\ (decl(PARENT), reference(ref(chartid(c))))
\end{array}$$

$$junction\_history : WF\_JUNCTION \to (DeclName \times Expression)$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ j : Junction \mid (c, j) \in WF_JUNCTION \bullet junction_history(c, j) = \\
\quad \textbf{if}\ j.history = \textbf{True} \\
\quad \textbf{then}\ (decl(HISTORY), reference(ref(T))) \\
\quad \textbf{else}\ (decl(HISTORY), reference(ref(F)))
\end{array}$$

$JunctionDeclaration : WF\_JUNCTION \rightarrow Paragraph$

$\forall\, c : WF\_CHART;\ j : Junction \mid (c, j) \in WF\_JUNCTION \bullet JunctionDeclaration(c, j) = \textbf{let}$
$\quad name == \langle decl(junctionname(c, j))\rangle;$
$\quad type == reference(ref(JUNCTION));$
$\quad binding == bindingextension(\langle$
$\quad\quad junction\_identifier(c, j),$
$\quad\quad junction\_transition(c, j),$
$\quad\quad junction\_parent(c, j),$
$\quad\quad junction\_history(c, j) \quad \rangle) \bullet \textbf{let}$
$\quad\quad decl == declpart(\langle variable(name, type)\rangle);$
$\quad\quad pred == relationapplication(rel(EQUAL, \langle reference(ref(junctionname(c, j))), binding\rangle)) \bullet$
$\quad\quad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$

$JunctionSetDeclaration : WF\_CHART \rightarrow Paragraph$

$\forall\, c : WF\_CHART \bullet JunctionSetDeclaration(c) = \textbf{let}$
$\quad names == set2seq(\{j : \text{ran } c.junctions \bullet reference(ref(junctionname(c, j)))\} \cup \{reference(ref(NULLJUNCTION))\}) \bullet \textbf{let}$
$\quad\quad pred == relationapplication(rel(EQUAL, \langle reference(ref(JUNCTION)), setextension(names)\rangle)) \bullet$
$\quad\quad\quad axiomaticdescription(schematext(\langle\rangle, \langle pred\rangle))$

$EventsDeclaration : WF\_CHART \rightarrow Paragraph$

$\forall\, c : WF\_CHART \bullet EventsDeclaration(c) = \textbf{let}$
$\quad names == set2seq(\{e : \text{ran } c.events \mid e.scope \neq OUTPUTEVENT \bullet decl(eventname(c, e))\} \cup \{decl(NULLEVENT)\});$
$\quad exprs == set2seq(\{e : \text{ran } c.events \mid e.scope \neq OUTPUTEVENT \bullet reference(ref(eventname(c, e)))\} \cup \{reference(ref(NULLEVENT))\});$
$\quad type == reference(ref(EVENT)) \bullet \textbf{let}$
$\quad\quad decl == declpart(\langle variable(names, type)\rangle);$
$\quad\quad pred == relationapplication(rel(EQUAL, \langle reference(ref(EVENT)), setextension(exprs)\rangle)) \bullet$
$\quad\quad\quad axiomaticdescription(schematext(\langle decl\rangle, \langle pred\rangle))$

$$ChartDeclaration : WF\_CHART \to Paragraph$$

$$\forall c : WF\_CHART \bullet ChartDeclaration(c) = \textbf{let}$$
$$states == set2seq(\{s : \operatorname{ran} c.states \bullet$$
$$tupleextension(\langle reference(ref(stateid(c, s))), reference(ref(statename(c, s)))\rangle)\});$$
$$chart == \langle tupleextension(\langle reference(ref(chartid(c))), reference(ref(chartname(c)))\rangle)\rangle;$$
$$transitions == set2seq(\{t : \operatorname{ran} c.transitions \bullet$$
$$tupleextension(\langle reference(ref(transitionid(c, t))), reference(ref(transitionname(c, t)))\rangle)\});$$
$$junctions == set2seq(\{j : \operatorname{ran} c.junctions \bullet$$
$$tupleextension(\langle reference(ref(junctionid(c, j))), reference(ref(junctionname(c, j)))\rangle)\}) \bullet \textbf{let}$$
$$decl == declpart(\langle expression(reference(ref(STATEFLOWCHART)))\rangle);$$
$$pred == newline(relationapplication(rel(EQUAL, \langle reference(ref(IDENTIFIER)), reference(ref(chartid(c)))\rangle)),$$
$$newline(relationapplication(rel(EQUAL, \langle reference(ref(STATES)), setextension(states \frown chart)\rangle)),$$
$$newline(relationapplication(rel(EQUAL, \langle reference(ref(TRANSITIONS)), setextension(transitions)\rangle)),$$
$$relationapplication(rel(EQUAL, \langle reference(ref(JUNCTIONS)), setextension(junctions)\rangle)))) \bullet$$
$$schemadefinition(processstatename(c), schematext(\langle decl\rangle, \langle pred\rangle))$$

$$DataDeclaration : WF\_DATA \to Declaration$$

$$\forall c : WF\_CHART;\ d : Data \mid (c, d) \in WF\_DATA \bullet DataDeclaration(c, d) =$$
$$variable(\langle decl(dataname(c, d))\rangle, translate_{Type}(d.type))$$

$$OutputEventDeclaration : WF\_EVENT \to Declaration$$

$$\forall c : WF\_CHART;\ e : Event \mid (c, e) \in WF\_EVENT \bullet OutputEventDeclaration(c, e) =$$
$$variable(\langle decl(eventcountername(c, e))\rangle, reference(ref(NATURAL)))$$

$$SimulationInstanceDeclaration : WF\_CHART \to Paragraph$$

$$\forall c : WF\_CHART \bullet SimulationInstanceDeclaration(c) = \textbf{let}$$
$$variables == squash\{i : \operatorname{dom} c.data \bullet i \mapsto DataDeclaration(c, c.data(i))\};$$
$$outputevents == squash\{i : \operatorname{dom} c.events \mid (c.events(i)).scope = OUTPUTEVENT \bullet$$
$$i \mapsto OutputEventDeclaration(c, c.events(i))\} \bullet \textbf{let}$$
$$decl == declpart(variables \frown outputevents) \bullet$$
$$schemadefinition(SimulationInstance, schematext(\langle decl\rangle, \langle\rangle))$$

$newline\_conjunction : \text{seq}_1\ Predicate \to Predicate$

$\forall\, ps : \text{seq}\ Predicate \mid \#\, ps = 1 \bullet newline\_conjunction(ps) = ps(1)$
$\forall\, ps : \text{seq}\ Predicate \mid \#\, ps > 1 \bullet newline\_conjunction(ps) = newline(head\ ps, newline\_conjunction(tail\ ps))$

$InitSimulationInstanceDeclaration : WF\_CHART \to Paragraph$

$\forall\, c : WF\_CHART \bullet InitSimulationInstanceDeclaration(c) = \textbf{let}$
$datainit == \{i : \text{dom}\ c.data \bullet$
$i \mapsto relationapplication(rel(EQUAL, \langle reference(ref(dataname(c, c.data(i)))), translate_{InitExpr}(c, c.data(i))\rangle))\};$
$eventinit == \{i : \text{dom}\ c.events \mid (c.events(i)).scope = OUTPUTEVENT \bullet$
$i \mapsto relationapplication(rel(EQUAL, \langle reference(ref(eventcountername(c, c.events(i)))), numberliteral(0)\rangle))\}$
$\bullet\ \textbf{let}\ decl == declpart(\langle expression(reference(ref(SimulationInstance)))\rangle);$
$pred == newline\_conjunction(datainit \frown eventinit) \bullet$
$schemadefinition(InitSimulationInstance, schematext(\langle decl\rangle, \langle pred\rangle))$

$InitStateDeclaration : Paragraph$

$InitStateDeclaration = horizontaldefinition(decl(InitState),$
$schemaconjunction(reference(ref(InitSimulationInstance)), reference(ref(InitSimulationData)))$
$)$

$EntryActionDeclaration : WF\_STATE \to PPar$

$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet EntryActionDeclaration(c, s) = \textbf{let}$
$name == EntryActionName(c, s);$
$comm == communication(EXECUTEENTRYACTION, \langle simpleparameter(reference(ref(stateid(c, s))))\rangle) \bullet$
$\left(\begin{array}{l} \textbf{if}\ \#\, s.entry > 0 \\ \textbf{then} \left(\begin{array}{l} \textbf{if}\ (c, s.entry) \in actions\_have\_broadcast \\ \textbf{then}\ actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Var}(translate_{Acts}(c, s.entry)))))) \\ \textbf{else}\ actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Acts}(c, s.entry))))) \end{array}\right) \\ \textbf{else}\ actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(skip))))) \end{array}\right)$

$$externalchoiceextension : \operatorname{seq} N \to Action$$

$$\forall as : \operatorname{seq} N \mid \# as = 0 \bullet externalchoiceextension(as) = cspaction(skip)$$
$$\forall as : \operatorname{seq} N \mid \# as = 1 \bullet externalchoiceextension(as) = namedaction(head\ as)$$
$$\forall as : \operatorname{seq} N \mid \# as > 1 \bullet externalchoiceextension(as) =$$
$$cspaction(actionexternalchoice(namedaction(head\ as), externalchoiceextension(tail\ as)))$$

$$EntryActionsDeclaration : WF\_CHART \to PPar$$

$$\forall c : WF\_CHART \bullet EntryActionsDeclaration(c) = \textbf{let}$$
$$actions == \{s : \operatorname{ran} c.states \bullet EntryActionName(c, s)\} \bullet$$
$$actionparagraph(ENTRYACTIONS, paraction(externalchoiceextension(set2seq(actions))))$$

$$DuringActionDeclaration : WF\_STATE \to PPar$$

$$\forall c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet DuringActionDeclaration(c, s) = \textbf{let}$$
$$name == DuringActionName(c, s);$$
$$comm == communication(EXECUTEDURINGACTION, \langle simpleparameter(reference(ref(stateid(c, s)))), inputparameter(E)\rangle) \bullet$$
$$\left(\begin{array}{l} \textbf{if } \#s.during > 0 \\ \textbf{then} \left(\begin{array}{l} \textbf{if } (c, s.during) \in dactions\_have\_broadcast \\ \textbf{then } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Var}(translate_{DActs}(c, s.during)))))) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{DActs}(c, s.during)))))) \end{array}\right) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))) \end{array}\right)$$

$$DuringActionsDeclaration : WF\_CHART \to PPar$$

$$\forall c : WF\_CHART \bullet DuringActionsDeclaration(c) = \textbf{let}$$
$$actions == \{s : \operatorname{ran} c.states \bullet DuringActionName(c, s)\} \bullet$$
$$actionparagraph(DURINGACTIONS, paraction(externalchoiceextension(set2seq(actions))))$$

$$ExitActionDeclaration : WF\_STATE \to PPar$$

$$\forall\, c : WF\_CHART;\ s : State \mid (c, s) \in WF\_STATE \bullet ExitActionDeclaration(c, s) = \textbf{let}$$
$$name == ExitActionName(c, s);$$
$$comm == communication(EXECUTEEXITACTION, \langle simpleparameter(reference(ref(stateid(c, s))))\rangle) \bullet$$
$$\left(\begin{array}{l} \textbf{if } \#\, s.exit > 0 \\ \textbf{then} \left(\begin{array}{l} \textbf{if } (c, s.exit) \in actions\_have\_broadcast \\ \textbf{then } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Var}(translate_{Acts}(c, s.exit)))))) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Acts}(c, s.exit)))))) \end{array}\right) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))) \end{array}\right)$$

$$ExitActionsDeclaration : WF\_CHART \to PPar$$

$$\forall\, c : WF\_CHART \bullet ExitActionsDeclaration(c) = \textbf{let}$$
$$actions == \{s : \operatorname{ran} c.states \bullet ExitActionName(c, s)\} \bullet$$
$$actionparagraph(EXITACTIONS, paraction(externalchoiceextension(set2seq(actions))))$$

$$ConditionActionDeclaration : WF\_TRANSITION \to PPar$$

$$\forall\, c : WF\_CHART;\ t : Transition \mid (c, t) \in WF\_TRANSITION \bullet ConditionActionDeclaration(c, t) = \textbf{let}$$
$$name == ConditionActionName(c, t);$$
$$comm == communication(EXECUTECONDITIONACTION, \langle simpleparameter(reference(ref(transitionid(c, t))))\rangle) \bullet$$
$$\left(\begin{array}{l} \textbf{if } \#\, t.condact > 0 \\ \textbf{then} \left(\begin{array}{l} \textbf{if } (c, t.condact) \in actions\_have\_broadcast \\ \textbf{then } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Var}(translate_{Acts}(c, t.condact)))))) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Acts}(c, t.condact)))))) \end{array}\right) \\ \textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))) \end{array}\right)$$

$$ConditionActionsDeclaration : WF\_CHART \to PPar$$

$$\forall\, c : WF\_CHART \bullet ConditionActionsDeclaration(c) = \textbf{let}$$
$$actions == \{t : \operatorname{ran} c.transitions \bullet ConditionActionName(c, t)\} \bullet$$
$$actionparagraph(CONDITIONACTIONS, paraction(externalchoiceextension(set2seq(actions))))$$

$$TransitionActionDeclaration : WF\_TRANSITION \to PPar$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ t : Transition \mid (c,t) \in WF_TRANSITION \bullet TransitionActionDeclaration(c,t) = \textbf{let} \\
\quad name == TransitionActionName(c,t); \\
\quad comm == communication(EXECUTETRANSITIONACTION, \langle simpleparameter(reference(ref(transitionid(c,t))))\rangle) \bullet \\
\left(\begin{array}{l}
\textbf{if } \#\, t.transact > 0 \\
\textbf{then} \left(\begin{array}{l}
\textbf{if } (c, t.transact) \in actions_have_broadcast \\
\textbf{then } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Var}(translate_{Acts}(c, t.transact)))))) \\
\textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, translate_{Acts}(c, t.transact)))))
\end{array}\right) \\
\textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))
\end{array}\right)
\end{array}$$

$$TransitionActionsDeclaration : WF\_CHART \to PPar$$

$$\begin{array}{l}
\forall\, c : WF_CHART \bullet TransitionActionsDeclaration(c) = \textbf{let} \\
\quad actions == \{t : \operatorname{ran} c.transitions \bullet TransitionActionName(c,t)\} \bullet \\
\quad\quad actionparagraph(TRANSITIONACTIONS, paraction(externalchoiceextension(set2seq(actions))))
\end{array}$$

$$ConditionDeclaration : WF\_TRANSITION \to PPar$$

$$\begin{array}{l}
\forall\, c : WF_CHART;\ t : Transition \mid (c,t) \in WF_TRANSITION \bullet ConditionDeclaration(c,t) = \textbf{let} \\
\quad name == ConditionName(c,t); \\
\quad truecomm == communication(EVALUATECONDITION, \\
\quad \langle simpleparameter(reference(ref(transitionid(c,t)))), outputparameter(reference(ref(T))))\rangle; \\
\quad falsecomm == communication(EVALUATECONDITION, \\
\quad \langle simpleparameter(reference(ref(transitionid(c,t)))), outputparameter(reference(ref(F))))\rangle) \bullet \\
\quad\quad \left(\begin{array}{l}
\textbf{if } \#\, t.condition > 0 \\
\textbf{then } actionparagraph(name, paraction(commandaction(ifcommand(gactions(\langle \\
\qquad\qquad (translate_{Pred}(c, head\ t.condition), cspaction(prefixedaction(truecomm, cspaction(skip)))), \\
\qquad\qquad (negation(translate_{Pred}(c, head\ t.condition)), cspaction(prefixedaction(falsecomm, cspaction(skip)))) \\
\qquad \rangle)))))) \\
\textbf{else } actionparagraph(name, paraction(cspaction(prefixedaction(truecomm, cspaction(skip)))))
\end{array}\right)
\end{array}$$

$$ConditionsDeclaration : WF\_CHART \to PPar$$

$$\begin{array}{l}
\forall\, c : WF_CHART \bullet ConditionsDeclaration(c) = \textbf{let} \\
\quad conditions == \{t : \operatorname{ran} c.transitions \bullet ConditionName(c,t)\} \bullet \\
\quad\quad actionparagraph(CONDITIONSACTION, paraction(externalchoiceextension(set2seq(conditions))))
\end{array}$$

$TriggerDeclaration : WF\_TRANSITION \to PPar$

$$
\begin{array}{l}
\forall\, c : WF\_CHART;\ t : Transition \mid (c,t) \in WF\_TRANSITION \bullet TriggerDeclaration(c,t) = \textbf{let} \\
\quad name == TriggerName(c,t); \\
\quad comm == communication(CHECKTRIGGERCHANNEL, \langle \\
\qquad simpleparameter(reference(ref(transitionid(c,t)))), \\
\qquad inputparameter(E) \\
\rangle);\quad trueguard == translate_{Triggers}(c, t.trigger);\quad truecomm == communication(RESULTCHANNEL, \langle \\
\qquad simpleparameter(reference(ref(transitionid(c,t)))), \\
\qquad simpleparameter(reference(ref(E))), \\
\qquad outputparameter(reference(ref(T))) \\
\quad \rangle);\quad falsecomm == communication(RESULTCHANNEL, \langle \\
\qquad simpleparameter(reference(ref(transitionid(c,t)))), \\
\qquad simpleparameter(reference(ref(E))), \\
\qquad outputparameter(reference(ref(F))) \\
\quad \rangle) \bullet \\
\qquad \left(\begin{array}{l}
\textbf{if}\ \#t.trigger > 0 \\
\textbf{then}\ actionparagraph(name, paraction(cspaction(prefixedaction(comm, \\
\qquad\qquad commandaction(ifcommand(gactions(\langle \\
\qquad\qquad\quad (trueguard, cspaction(prefixedaction(truecomm, cspaction(skip)))), \\
\qquad\qquad\quad (negation(trueguard), cspaction(prefixedaction(falsecomm, cspaction(skip)))) \\
\qquad\qquad \rangle)))))))) \\
\textbf{else}\ actionparagraph(name, paraction(cspaction(prefixedaction(comm, cspaction(prefixedaction(truecomm, cspaction(skip))))))))
\end{array} \right)
\end{array}
$$

$TriggersDeclaration : WF\_CHART \to PPar$

$$
\begin{array}{l}
\forall\, c : WF\_CHART \bullet TriggersDeclaration(c) = \textbf{let} \\
\quad triggers == \{t : \mathrm{ran}\, c.transitions \bullet TriggerName(c,t)\} \bullet \\
\qquad actionparagraph(TRIGGERSACTION, paraction(externalchoiceextension(set2seq(triggers))))
\end{array}
$$

Next, we define the paragraphs of the specification that are independent from the diagram; these are defined as constants of type $PPar$.

$GetStateDeclaration : PPar$

$GetStateDeclaration = \textbf{let}$
$comm == communication(STATECHANNEL, \langle$
$guardedinputparameter(X, relationapplication(rel(IN, \langle$
$reference(ref(X)), functionapplication(app(DOM, \langle reference(ref(STATES))\rangle)))$
$\rangle))),$
$outputparameter(functionapplication(app(STATES, \langle reference(ref(X))\rangle)))$
$\rangle) \bullet actionparagraph(GETSTATE, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$GetJunctionDeclaration : PPar$

$GetJunctionDeclaration = \textbf{let}$
$comm == communication(JUNCTIONCHANNEL, \langle$
$guardedinputparameter(X, relationapplication(rel(IN, \langle$
$reference(ref(X)), functionapplication(app(DOM, \langle reference(ref(JUNCTIONS))\rangle)))$
$\rangle))),$
$outputparameter(functionapplication(app(JUNCTIONS, \langle reference(ref(X))\rangle)))$
$\rangle) \bullet actionparagraph(GETJUNCTION, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$GetTransitionDeclaration : PPar$

$GetTransitionDeclaration = \textbf{let}$
$comm == communication(TRANSITIONCHANNEL, \langle$
$guardedinputparameter(X, relationapplication(rel(IN, \langle$
$reference(ref(X)), functionapplication(app(DOM, \langle reference(ref(TRANSITIONS))\rangle)))$
$\rangle))),$
$outputparameter(functionapplication(app(TRANSITIONS, \langle reference(ref(X))\rangle)))$
$\rangle) \bullet actionparagraph(GETTRANSITION, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$GetEventsDeclaration : WF\_CHART \to PPar$

$\forall\, c : WF\_CHART \bullet GetEventsDeclaration(c) = \textbf{let}$
$\quad seqevents == squash(\{e : \mathrm{ran}(c.events) \mid e.scope = INPUTEVENT \bullet e.identifier \mapsto reference(ref(eventname(c, e)))\}) \bullet \textbf{let}$
$comm == communication(EVENTSCHANNEL, \langle outputparameter(sequence($
$\quad \textbf{if } seqevents = \langle\rangle \textbf{ then } \langle reference(ref(NULLEVENT))\rangle \textbf{ else } seqevents))\rangle)$
$\bullet\ actionparagraph(GETEVENTS, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$GetChartDeclaration : PPar$

$GetChartDeclaration = \textbf{let}$
$\quad comm == communication(CHARTCHANNEL, \langle$
$\quad\quad outputparameter(functionapplication(app(STATES, \langle reference(ref(IDENTIFIER))\rangle)))$
$\quad \rangle) \bullet actionparagraph(GETCHART, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$StatusDeclaration : PPar$

$StatusDeclaration = \textbf{let}$
$\quad comm == communication(STATUSCHANNEL, \langle$
$\quad\quad guardedinputparameter(X, relationapplication(rel(IN, \langle$
$\quad\quad\quad reference(ref(X)), functionapplication(app(DOM, \langle reference(ref(STATESTATUS))\rangle))$
$\quad\quad \rangle))),$
$\quad\quad outputparameter(functionapplication(app(STATESTATUS, \langle reference(ref(X))\rangle)))$
$\quad \rangle) \bullet actionparagraph(STATUSACTION, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$HistoryDeclaration : PPar$

$HistoryDeclaration = \textbf{let}$
$\quad comm == communication(HISTORYCHANNEL, \langle$
$\quad\quad guardedinputparameter(X, relationapplication(rel(IN, \langle$
$\quad\quad\quad reference(ref(X)), functionapplication(app(DOM, \langle reference(ref(STATEHISTORY))\rangle))$
$\quad\quad \rangle))),$
$\quad\quad outputparameter(functionapplication(app(STATEHISTORY, \langle reference(ref(X))\rangle)))$
$\quad \rangle) \bullet actionparagraph(HISTORYACTION, paraction(cspaction(prefixedaction(comm, cspaction(skip)))))$

$ActivationDeclaration : PPar$

$ActivationDeclaration =$ **let**
$comm == communication(ACTIVATECHANNEL, \langle$
$inputparameter(X)\rangle);$
$activate == schemaaction(reference(ref(ACTIVATESCHEMA))) \bullet$
$actionparagraph(ACTIVATIONACTION, paraction(cspaction(prefixedaction(comm, activate))))$

$DeactivationDeclaration : PPar$

$DeactivationDeclaration =$ **let**
$comm == communication(DEACTIVATECHANNEL, \langle$
$inputparameter(X)\rangle);$
$deactivate == schemaaction(reference(ref(DEACTIVATESCHEMA))) \bullet$
$actionparagraph(DEACTIVATIONACTION, paraction(cspaction(prefixedaction(comm, deactivate))))$

$ChartActionsDeclaration : PPar$

$ChartActionsDeclaration =$ **let**
$actions == \langle ENTRYACTIONS, DURINGACTIONS, EXITACTIONS, CONDITIONACTIONS, TRANSITIONACTIONS\rangle \bullet$ **let**
$a1 == externalchoiceextension(actions);$
$a2 == cspaction(prefixedaction(communication(ENDACTION, \langle\rangle), cspaction(skip))) \bullet$
$actionparagraph(CHARTACTIONS, paraction(cspaction(actionsequentialcomposition(a1, a2))))$

$InterfaceActionsDeclaration : PPar$

$InterfaceActionsDeclaration =$ **let**
$actions == \langle GETCHART, GETSTATE, GETJUNCTION, GETTRANSITION, STATUSACTION, HISTORYACTION,$
$ACTIVATIONACTION, DEACTIVATIONACTION\rangle \bullet$
$actionparagraph(INTERFACEACTIONS, paraction(externalchoiceextension(actions)))$

$$interleaveextension : \text{seq}(Action \times \text{seq}\, N) \to Action$$

$$\forall\, as : \text{seq}(Action \times \text{seq}\, N) \mid \#\, as = 0 \bullet interleaveextension(as) = cspaction(skip)$$
$$\forall\, as : \text{seq}(Action \times \text{seq}\, N) \mid \#\, as = 1 \bullet interleaveextension(as) = (head\ as).1$$
$$\forall\, as : \text{seq}(Action \times \text{seq}\, N) \mid \#\, as > 1 \bullet interleaveextension(as) =$$
$$cspaction(actioninterleave((head\ as).1, ns((head\ as).2), ns(\frown/\{i : \text{dom}\,(tail\ as) \bullet i \mapsto ((tail\ as)i).2\}), interleaveextension(tail\ as)))$$

$$InputsDeclaration : WF\_CHART \to PPar$$

$$\forall\, c : WF\_CHART \bullet InputsDeclaration(c) = \textbf{let}$$
$$channels == \{d : \text{ran}\, c.data \mid d.scope = INPUTDATA \bullet (datachannelname(c, d), dataname(c, d))\} \bullet \textbf{let}$$
$$commassign == \{cv : channels \bullet (communication(cv.1, \langle inputparameter(X)\rangle), assignment(\langle(cv.2, reference(ref(X)))\rangle), \langle cv.2\rangle)\} \bullet \textbf{let}$$
$$actions == \{ca : commassign \bullet (cspaction(prefixedaction(ca.1, commandaction(ca.2))), ca.3)\} \bullet$$
$$actionparagraph(INPUTSACTION, paraction(cspaction(prefixedaction(communication(READINPUTSCHANNEL, \langle\rangle),$$
$$interleaveextension(set2seq(actions))))))$$

$$WriteOutputEvent : WF\_CHART \times Event \to Action$$

$$\forall\, c : WF\_CHART;\ e : Event \mid e.scope = OUTPUTEVENT \bullet$$
$$WriteOutputEvent(c, e) = \textbf{let}\ pred == relationapplication(rel(GT, \langle reference(ref(eventcountername(c, e))), numberliteral(0)\rangle));$$
$$pair == (eventcountername(c, e), functionapplication(app(MINUS, \langle reference(ref(eventcountername(c, e))), numberliteral(1)\rangle)));$$
$$commtrue == communication(outputeventchannelname(c, e), \langle outputparameter(reference(ref(T)))\rangle);$$
$$commfalse == communication(outputeventchannelname(c, e), \langle outputparameter(reference(ref(F)))\rangle) \bullet \textbf{let}$$
$$a1 == cspaction(prefixedaction(commtrue, commandaction(assignment(\langle pair\rangle))));$$
$$a2 == cspaction(prefixedaction(commfalse, cspaction(skip))) \bullet$$
$$commandaction(ifcommand(gactions(\langle(pred, a1), (negation(pred), a2)\rangle)))$$

$$OutputsDeclaration : WF\_CHART \to PPar$$

$$\forall\, c : WF\_CHART \bullet OutputsDeclaration(c) = \textbf{let}$$
$$channels == \{d : \text{ran}\, c.data \mid d.scope = OUTPUTDATA \bullet (datachannelname(c, d), dataname(c, d))\} \bullet \textbf{let}$$
$$comm == \{cv : channels \bullet (communication(cv.1, \langle outputparameter(reference(ref(cv.2)))\rangle), \langle cv.2\rangle)\} \bullet \textbf{let}$$
$$actions == \{c : comm \bullet (cspaction(prefixedaction(c.1, cspaction(skip))), c.2)\} \cup$$
$$\{e : \text{ran}\, c.events \mid e.scope = OUTPUTEVENT \bullet (WriteOutputEvent(c, e), \langle eventcountername(c, e)\rangle)\} \bullet$$
$$actionparagraph(OUTPUTSACTION, paraction(cspaction(prefixedaction(communication(WRITEOUTPUTSCHANNEL, \langle\rangle),$$
$$interleaveextension(set2seq(actions))))))$$

$AllActionsDeclaration : PPar$

$AllActionsDeclaration = \textbf{let}$
$\quad actions == \langle CONDITIONSACTION, TRIGGERSACTION, INPUTSACTION, OUTPUTSACTION, CHARTACTIONS,$
$\quad\quad INTERFACEACTIONS\rangle \bullet$
$\quad actionparagraph(ALLACTIONS, paraction(externalchoiceextension(actions)))$

$MainActionDeclaration : Action$

$MainActionDeclaration = \textbf{let}$
$\quad first == schemaaction(reference(ref(InitState))) \bullet$
$\quad cspaction(actionsequentialcomposition(first, cspaction(recursiveaction(X,$
$\quad\quad cspaction(actionsequentialcomposition(namedaction(ALLACTIONS), namedaction(X)))))))$

$ProcessStateDeclaration : WF\_CHART \rightarrow ProcessState$

$\forall c : WF\_CHART \bullet ProcessStateDeclaration(c) =$
$\quad stateDecl(decl(processstatename(c)), schemaconjunction(reference(ref(SimulationInstance)), reference(ref(SimulationData))))$

$BroadcastDeclaration : PPar$
$CheckDeclaration : PPar$

$ProcessDeclaration : WF\_CHART \to ProcDecl$

$\forall\, c : WF\_CHART \bullet ProcessDeclaration(c) = \textbf{let}$
$stateparagraphs == \langle zparagraph(ChartDeclaration(c)), zparagraph(SimulationInstanceDeclaration(c)),$
$\quad zparagraph(InitSimulationInstanceDeclaration(c)), zparagraph(InitStateDeclaration)\rangle;$
$state == ProcessStateDeclaration(c);$
$entryactions == set2seq(\{s : \text{ran } c.states \bullet EntryActionDeclaration(c, s)\} \cup \{EntryActionsDeclaration(c)\});$
$duringactions == set2seq(\{s : \text{ran } c.states \bullet DuringActionDeclaration(c, s)\} \cup \{DuringActionsDeclaration(c)\});$
$exitactions == set2seq(\{s : \text{ran } c.states \bullet ExitActionDeclaration(c, s)\} \cup \{ExitActionsDeclaration(c)\});$
$conditionactions == set2seq(\{t : \text{ran } c.transitions \bullet ConditionActionDeclaration(c, t)\} \cup \{ConditionActionsDeclaration(c)\});$
$transitionactions == set2seq(\{t : \text{ran } c.transitions \bullet TransitionActionDeclaration(c, t)\} \cup \{TransitionActionsDeclaration(c)\});$
$conditions == set2seq(\{t : \text{ran } c.transitions \bullet ConditionDeclaration(c, t)\} \cup \{ConditionsDeclaration(c)\});$
$triggers == set2seq(\{t : \text{ran } c.transitions \bullet TriggerDeclaration(c, t)\} \cup \{TriggersDeclaration(c)\});$
$getters == \langle GetStateDeclaration, GetJunctionDeclaration, GetTransitionDeclaration, GetChartDeclaration, GetEventsDeclaration(c)\rangle;$
$broadcast == \langle BroadcastDeclaration, CheckDeclaration\rangle;$
$others == \langle StatusDeclaration, HistoryDeclaration, ActivationDeclaration, DeactivationDeclaration\rangle;$
$collections == \langle ChartActionsDeclaration, InterfaceActionsDeclaration, InputsDeclaration(c), OutputsDeclaration(c), AllActionsDeclaration\rangle \bullet$
$\textbf{let } actions == entryactions \frown duringactions \frown exitactions \frown conditionactions \frown transitionactions \frown conditions \frown triggers \frown$
$\quad getters \frown broadcast \frown others \frown collections \bullet$
$\quad\quad simpleprocessdeclaration(processname(c), processdefinition(proc(stateparagraphs, state, actions, MainActionDeclaration)))$

$ChannelsDeclaration : WF\_CHART \to CircusPar$

$\forall\, c : WF\_CHART \bullet ChannelsDeclaration(c) = \textbf{let}$
$datachannelsdecl == set2seq(\{d : \text{ran } c.data \mid d.scope = INPUTDATA \lor d.scope = OUTPUTDATA \bullet$
$\quad typedsimplecdecl(\langle datachannelname(c, d)\rangle, translate_{Type}(d.type))\});$
$eventchannels == \{e : \text{ran } c.events \mid e.scope = OUTPUTEVENT \bullet outputeventchannelname(c, e)\} \bullet$
$\quad channeldeclaration(cdecl(datachannelsdecl \frown \langle typedsimplecdecl(set2seq(eventchannels), reference(ref(BOOL)))\rangle))$

$translate : WF\_CHART \to Program$

$\forall\, c : WF\_CHART \bullet translate(c) = \textbf{let}$
$identifiers == \langle zparagraphdeclaration(StateIdentifierDeclaration(c)),$
$\quad zparagraphdeclaration(JunctionIdentifierDeclaration(c)),$
$\quad zparagraphdeclaration(TransitionIdentifierDeclaration(c))\rangle;$
$chart == \langle zparagraphdeclaration(ChartAsStateDeclaration(c))\rangle;$
$states == set2seq(\{s : \operatorname{ran} c.states \bullet zparagraphdeclaration(StateDeclaration(c, s))\})^\frown$
$\quad \langle zparagraphdeclaration(StateSetDeclaration(c))\rangle;$
$junctions == set2seq(\{j : \operatorname{ran} c.junctions \bullet zparagraphdeclaration(JunctionDeclaration(c, j))\})^\frown$
$\quad \langle zparagraphdeclaration(JunctionSetDeclaration(c))\rangle;$
$transitions == set2seq(\{t : \operatorname{ran} c.transitions \bullet zparagraphdeclaration(TransitionDeclaration(c, t))\})^\frown$
$\quad \langle zparagraphdeclaration(TransitionSetDeclaration(c))\rangle;$
$events == \langle zparagraphdeclaration(EventsDeclaration(c))\rangle;$
$channels == \langle ChannelsDeclaration(c)\rangle;$
$proc == \langle processdeclaration(ProcessDeclaration(c))\rangle \bullet$
$program(identifiers^\frown chart^\frown states^\frown junctions^\frown transitions^\frown events^\frown channels^\frown proc)$