Translation Rules

High Level

section *CircusBNFEncoding* **parents** *standard_toolkit*

$[Predicate, N, Expression, Paragraph, Schema_Exp, Declaration]$

$Command ::= type\langle\langle \mathrm{seq}\, N \times Predicate \times Predicate \rangle\rangle \mid equals\langle\langle N \times \mathrm{seq}\, Expression \rangle\rangle \mid$
$\quad dot\langle\langle Expression \rangle\rangle$

$CParameter ::= shriek\langle\langle N \rangle\rangle \mid shreikRestrict\langle\langle N \times Predicate \rangle\rangle \mid bang\langle\langle Expression \rangle\rangle \mid$
$\quad dotParam\langle\langle Expression \rangle\rangle$

$Communication == N \times \mathrm{seq}\, CParameter$

$CSExpression ::= emptyCS \mid cs\langle\langle \mathrm{seq}\, N \rangle\rangle \mid csName\langle\langle N \rangle\rangle \mid$
$\quad union\langle\langle CSExpression \times CSExpression \rangle\rangle \mid$
$\quad intersect\langle\langle CSExpression \times CSExpression \rangle\rangle \mid$
$\quad subtract\langle\langle CSExpression \times CSExpression \rangle\rangle$

$Action ::= actSe\langle\langle Schema_Exp \rangle\rangle \mid com\langle\langle Command \rangle\rangle \mid skip \mid stop \mid chaos \mid$
$\quad thenExp\langle\langle Communication \times Action \rangle\rangle \mid$
$\quad guard\langle\langle Predicate \times Action \rangle\rangle \mid seqExp\langle\langle Action \times Action \rangle\rangle \mid$
$\quad extChoice\langle\langle Action \times Action \rangle\rangle \mid intChoice\langle\langle Action \times Action \rangle\rangle \mid$
$\quad actParam\langle\langle Action \times CSExpression \times Action \rangle\rangle \mid actIntr\langle\langle Action \times Action \rangle\rangle \mid$
$\quad actHide\langle\langle Action \times CSExpression \rangle\rangle \mid itr\langle\langle N \times Action \rangle\rangle \mid cspSpot\langle\langle Declaration \times Action \rangle\rangle \mid$
$\quad tmp6\langle\langle Action \times \mathrm{seq}\, Expression \rangle\rangle$

$GuardedAction ::= thenAct\langle\langle Predicate \times Action \rangle\rangle \mid$
$\quad thenActComp\langle\langle Predicate \times Action \times GuardedAction \rangle\rangle$

$PParagraph ::= pPar\langle\langle Paragraph \rangle\rangle \mid def\langle\langle N \times Action \rangle\rangle$

$Process ::= proc\langle\langle \mathrm{seq}\, PParagraph \times Action \rangle\rangle \mid procName\langle\langle N \rangle\rangle \mid$
$\quad procSeq\langle\langle Process \times Process \rangle\rangle \mid procExtChoice\langle\langle Process \times Process \rangle\rangle \mid$
$\quad procIntChoice\langle\langle Process \times Process \rangle\rangle \mid procPar\langle\langle Process \times CSExpression \times Process \rangle\rangle \mid$
$\quad procItr\langle\langle Process \times Process \rangle\rangle \mid procHide\langle\langle Process \times CSExpression \rangle\rangle \mid$
$\quad oSpot\langle\langle Declaration \times Process \rangle\rangle \mid tmp1\langle\langle Process \times \mathrm{seq}\, Expression \rangle\rangle \mid$
$\quad tmp2\langle\langle Process \times \mathrm{seq}\, N \times \mathrm{seq}\, N \rangle\rangle \mid procSpot\langle\langle Declaration \times Process \rangle\rangle \mid$
$\quad tmp3\langle\langle Process \times \mathrm{seq}\, Expression \rangle\rangle \mid tmp4\langle\langle \mathrm{seq}\, N \times Process \rangle\rangle \mid$
$\quad tmp5\langle\langle Process \times \mathrm{seq}\, Expression \rangle\rangle$

$ProcDefinition ::= pd\langle\langle Process \times N \times Process \rangle\rangle$

$ChanSetDefinition ::= csdName\langle\langle N \times CSExpression \rangle\rangle$

$SCDeclaration ::= chanName\langle\langle \mathrm{seq}\, N \rangle\rangle \mid chanNameWithType\langle\langle \mathrm{seq}\, N \times Expression \rangle\rangle \mid$
$\quad scSe\langle\langle Schema_Exp \rangle\rangle$

$CDeclaration ::= scDecl\langle\langle SCDeclaration \rangle\rangle \mid multiDecl\langle\langle SCDeclaration \times CDeclaration \rangle\rangle$

$ChannelDefinition == CDeclaration$

$CircusParagraph ::= para\langle\langle Paragraph \rangle\rangle \mid chanDef\langle\langle ChannelDefinition \rangle\rangle \mid$
$\quad chanSetDef\langle\langle ChanSetDefinition \rangle\rangle \mid procDef\langle\langle ProcDefinition \rangle\rangle$

$Program == \mathrm{seq}\, CircusParagraph$

section *SCJBNFEncoding* **parents** *standard_toolkit*

$[MethodBody, ClassBodyDeclaration, Identifier, MethodDeclaration]$

$$Run == MethodBody$$
$$ManagedThreadClassBody == Run \times \text{seq}\ ClassBodyDeclaration$$
$$ManagedThread == Identifier \times ManagedThreadClassBody$$

$$HandleAsyncEvent == MethodBody$$
$$HandleAsyncLongEvent == MethodBody$$
$$EventHandlerClassBody == HandleAsyncEvent \times \text{seq}\ ClassBodyDeclaration$$
$$OneShotEventHandler == Identifier \times EventHandlerClassBody$$
$$LongEventHandlerClassBody == HandleAsyncLongEvent \times \text{seq}\ ClassBodyDeclaration$$
$$AperiodicEventHandler ::= apehType\langle\langle Identifier \times EventHandlerClassBody\rangle\rangle \mid aplehType\langle\langle Identifier \times LongEventHandlerClassBody\rangle\rangle$$
$$PeriodicEventHandler == Identifier \times EventHandlerClassBody$$
$$EventHandler ::= pehDecl\langle\langle PeriodicEventHandler\rangle\rangle \mid apehDecl\langle\langle AperiodicEventHandler\rangle\rangle \mid osehDecl\langle\langle OneShotEventHandler\rangle\rangle$$

$$GetNextMission == MethodBody$$
$$MissionSequencerClassBody == GetNextMission \times \text{seq}\ ClassBodyDeclaration$$
$$MissionSequencer == Identifier \times MissionSequencerClassBody$$

$$NestedMissionSequencer == MissionSequencer$$

$$SchedulableObject ::= handler\langle\langle EventHandler\rangle\rangle \mid apeh\langle\langle AperiodicEventHandler\rangle\rangle \mid nms\langle\langle NestedMissionSequencer\rangle\rangle$$

$$Cleanup == MethodBody$$
$$Initialize == MethodBody$$
$$MissionClassBody == Initialize \times Cleanup \times \text{seq}\ ClassBodyDeclaration$$
$$Mission == Identifier \times MissionClassBody$$

$$Tier == Mission \times \text{seq}\ SchedulableObject$$

$$TopLevelMissionSequencer == MissionSequencer$$

$$ImmortalMemorySize == MethodDeclaration$$
$$InitializeApplication == MethodBody$$
$$GetSequencer == MethodBody$$
$$SafeletClassBody == GetSequencer \times InitializeApplication \times ImmortalMemorySize \times \mathrm{seq}\ ClassBodyDeclaration$$
$$Safelet == Identifier \times SafeletClassBody$$

$$SCJProgram == Safelet \times TopLevelMissionSequencer \times \mathrm{seq}\ Tier$$

section *HighLevelRules* **parents** *SCJBNFEncoding*, *CircusBNFEncoding*, *standard_toolkit*

$$TransMeth : SCJMethod \pfun Action$$

$$\forall m : SCJMethod \bullet \exists ms : SCJMethSig;\ b : SCJBlock;\ p : Params \bullet TransMeth(m) = (TransMethSig(ms), TransBlock(b), TransParams(p))$$

$$TransMeths : \seq SCJMethod \pfun \seq Action$$

$$\forall meths : \seq SCJMethod \bullet \forall meth : SCJMethod \bullet TransMeths(\langle meth \rangle) = TransMeth(meth) \land TransMeths(\langle meth \rangle \cat meths) = \langle TransMeth(meth) \cat TransMeths(meths) \rangle$$

$$TransParams : Params \pfun CircParams$$

$$\forall params : Params \bullet \forall param : Param \bullet TransParams(\langle param \rangle) = TransParam(param) \land TransParams(\langle param \rangle \cat params) = \langle TransParam(param) \cat TransParams(params) \rangle$$

$$TransParam : Param \pfun CircParam$$

$$\forall p : Param \bullet \exists cParam : CircParam \bullet TransParam(p) = cParam$$

$$TransBlock : SCJBlock \pfun CircBlock$$

$$\forall block : SCJBlock \bullet \forall e : Expr \bullet TransBlock(\langle e \rangle) = TransExpr(e) \land TransBlock(\langle e \rangle \cat block) = \langle TransExpr(e) \cat TransBlock(block) \rangle$$

$$TransClass : SCJClass \pfun \seq CircParagraph$$

$$\forall class : SCJClass \mid \exists meths : \seq Methods \bullet meths = MethodsOf(class) \bullet TransClass(class) = \langle TransProc(class), TransOhClass(class), TransChans(meths)$$

$$TransClasses : \seq SCJClass \pfun \seq CircParagraph$$

$$\forall classes : \seq SCJClass \bullet \forall class : SCJClass \bullet TransClasses(\langle class \rangle) = TransClass(class) \land TransClasses(\langle class \rangle \cat classes) = TransClass(class) \cat TransClasses(classes)$$

$$TransSCJProg : SCJProg \pfun CircusProg \times CircusProg$$

$$\forall scjProg : SCJProg \bullet \exists f : Framework \bullet TransSCJProg(scjProg) = (TransClasses(scjProg), f)$$

Low Level

- $Method : MethodDeclaration \pfun (Name, Params, ReturnType, Body)$: translates an active application method into a *Circus* action
- $DataMethod : MethodDeclaration \pfun$: translates data methods into an *OhCircus* method
- $MethodBody : Block \pfun \seq CircExpression$: translates a Java block, for example a method body
- $Registers : Block \pfun \seq Name$: extracts the Names of the schedulables registered in a Java block
- $Returns : Block \pfun \seq Name$: extracts the Names of the variables retuned in a Java block
- $Variable : (Name, Type, InitExpression) \pfun (CircName, CircType, CircExpression)$: translates a variable
- $Parameters : (Name, Params, ReturnType, Body) \pfun \seq CircParam$: translates a list of method parameters
- $[\![Name]\!]_{name}$: translates the *name* to a Z identifier
- $[\![varType]\!]_{type}$: translates types
- $[\![expr]\!]_{expression}$: translates expressions

Auxiliary Functions

- *IdOf*(*name*): yields the identifier of a component called *name*
- *ObjectIdOf*(*name*): yields the identifier of the *Object* process of a component called *name*
- *ThreadIdOf*(*name*): yields the identifier of the *Thread* process of a component called *name*
- *MethodName*(*method*): yields the method name of *method*
- *MethodsOf*(*name*) : yeilds a sequence of methods from the class *name*

Pattern Matching Rules

Safelet

```
1  public class Identifier implements Safelet
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     initializeApplication
10
11    getSequencer
12
13    AppMeth_1
14    ...
15    AppMeth_n
16 }
```

process $[\![Identifier]\!]_{Name}$ $App \mathrel{\widehat=} [\![[\![ConstructorDeclaration]\!]_{Method}]\!]_{Parameters}$ **begin**

State

$this : \text{ref } [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new } [\![Identifier]\!]_{name}\ Class()$

$$InitializeApplication \mathrel{\widehat=} \left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ [\![[\![InitializeApplication]\!]_{Method}]\!]_{MethBody} \\ initializeApplicationRet \longrightarrow \\ \textbf{Skip} \end{array} \right)$$

$$GetSequencer \mathrel{\widehat=} \left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet\,!\,[\![GetSequencer]\!]_{Returns} \longrightarrow \\ \textbf{Skip} \end{array} \right)$$

$[\![AppMeth\_1]\!]_{Method}$

...

$[\![AppMeth\_n]\!]_{Method}$

$$
\begin{array}{l}
Methods \mathrel{\widehat{=}} \\
\left(\begin{array}{l}
GetSequencer \\
\Box \\
InitializeApplication \\
\Box \\
MethName(AppMeth\_1) \\
\Box \\
\ldots \\
MethName(AppMeth\_n) \\
\ldots
\end{array}\right) \; ; \; Methods
\end{array}
$$

- $(Init \; ; \; Methods) \;\; \triangle(end\_safelet\_app \longrightarrow \mathbf{Skip})$

end

Mission Sequencer

```
1  public class Identifier extends MissionSequencer
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     getNextMission
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

process $[\![Identifier]\!]_{Name}\ App \mathrel{\widehat=} [\![\ [\![ConstructorDeclaration]\!]_{Method}]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$GetNextMission \mathrel{\widehat=} \textbf{var}\ ret : MissionID \bullet$

$$\begin{pmatrix} getNextMissionCall\,.\,IdOf(Identifier) \longrightarrow \\ ret := this\,.\,getNextMission(); \\ getNextMissionRet\,.\,IdOf(Identifier)\,!\,ret \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$[\![AppMeth\_1]\!]_{Method}$

$\ldots$

$[\![AppMeth\_n]\!]_{Method}$

$$Methods \mathrel{\widehat=} \begin{pmatrix} GetNextMission \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ;\ Methods$$

$\bullet\ (Init\ ;\ Methods)\ \ \triangle(end\_sequencer\_app\,.\,IdOf(Identifier) \longrightarrow \textbf{Skip})$

end

Mission

```
1  public class Identifier extends Mission
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     initialize
10
11    cleanUp
12
13    AppMeth_1
14    ...
15    AppMeth_n
16 }
```

process $[\![Indentifier]\!]\ App \mathrel{\widehat=} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$$InitializePhase \mathrel{\widehat=} \begin{pmatrix} initializeCall\,.\,IdOf(Indentifier) \longrightarrow \\ [\![initialize]\!]_{Registers}\ initializeRet\,.\,IdOf(Indentifier) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$CleanupPhase \mathrel{\widehat=} \begin{pmatrix} cleanupMissionCall\,.\,IdOf(Indentifier) \longrightarrow \\ cleanupMissionRet\,.\,IdOf(Indentifier)\,!\,\textbf{True} \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$[\![AppMeth\_1]\!]_{Method}$

...

$[\![AppMeth\_n]\!]_{Method}$

$$Methods \mathrel{\widehat=} \begin{pmatrix} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ;\ Methods$$

• $(Init\ ;\ Methods)\ \ \triangle(end\_mission\_app\ .\ IdOf(Identifier) \longrightarrow \textbf{Skip}$

end

Handlers

```
1  class Identifier extends HandlerType
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     handleAsyncEvent
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

process $[\![PName]\!]\ App \mathrel{\widehat=} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}$ **begin**

State

$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$

state *State*

Init

$State'$

$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$

$$handleAsyncEvent \mathrel{\widehat=} \begin{pmatrix} handleAsyncEventCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![\ HandleAsyncBody\ ]\!]_{Method}\ ]\!]_{MethBody}; \\ handleAsyncEventRet\,.\,IdOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$[\![AppMeth\_1]\!]_{Method}$

...

$[\![AppMeth\_n]\!]_{Method}$

$$Methods \mathrel{\widehat=} \begin{pmatrix} handleAsyncEvent \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \dots \end{pmatrix};\ Methods$$

$\bullet\ (Init\ ;\ Methods)\ \ \triangle(end\_[\![\ HandlerTypeIdOf(PName)\ ]\!] \longrightarrow \textbf{Skip})$

end

Managed Thread

```
1  public class Identifier extends ManagedThread
2  {
3     FieldDeclaration_1
4     ...
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     run
10
11    AppMeth_1
12    ...
13    AppMeth_n
14 }
```

$$\textbf{process}\ [\![PName]\!]\ App \mathrel{\widehat=} [\![\ [\![ConstructorDeclaration]\!]_{Method}]\!]_{Parameters}\ \textbf{begin}$$

State

$$this : \text{ref}\ [\![Identifier]\!]_{name}\ Class$$

$$\textbf{state}\ State$$

Init

$$State'$$
$$this := \textbf{new}\ [\![Identifier]\!]_{name}\ Class()$$

$$Run \mathrel{\widehat=} \begin{pmatrix} runCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![run]\!]_{Method}]\!]_{MethBody}; \\ runRet\,.\,IfOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$[\![AppMeth\_1]\!]_{Method}$$

$$\ldots$$

$$[\![AppMeth\_n]\!]_{Method}$$

$$Methods \mathrel{\widehat=} \begin{pmatrix} Run \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix};\ Methods$$

$$\bullet\ (Init\ ;\ Methods)\ \ \triangle(end\_managedThread\_app\,.\,IdOf(PName) \longrightarrow \textbf{Skip})$$

$$\textbf{end}$$

Data Class

class $[\![PName]\!]_{name}$ *Class* $\widehat{=}$ **begin**

state *State*

$[\![VarName]\!]_{name} : [\![VarType]\!]_{type}$

state *State*

initial *Init*

$State'$

$[\![VarName]\!]'_{name} = [\![VarInit]\!]_{expression}$

$[\![DataMeth1]\!]_{dataMeth}$
$[\![DataMeth2]\!]_{dataMeth}$
...

- **Skip**

end