

Translation Rules

High Level

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

$[Predicate, N, Expression, Paragraph, Schema_{Exp}, Declaration]$

$Command ::= type \langle \langle seq\ N \times Predicate \times Predicate \rangle \rangle \mid equals \langle \langle N \times seq\ Expression \rangle \rangle \mid dot \langle \langle Expression \rangle \rangle$

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

$Communication == N \times seq\ CParameter$

$CSExpression ::= emptyCS \mid cs \langle \langle seq\ N \rangle \rangle \mid csName \langle \langle N \rangle \rangle \mid union \langle \langle CSExpression \times CSExpression \rangle \rangle \mid intersect \langle \langle CSExpression \times CSExpression \rangle \rangle \mid subtract \langle \langle CSExpression \times CSExpression \rangle \rangle$

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

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

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

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

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

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

$SCDeclaration ::= chanName \langle \langle seq\ N \rangle \rangle \mid chanNameWithType \langle \langle seq\ N \times Expression \rangle \rangle \mid scSe \langle \langle Schema_{Exp} \rangle \rangle$

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

$ChannelDefinition == CDeclaration$

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

$Program == seq\ CircusParagraph$

section *SCJBNFEncoding parents standard_toolkit*

[*MethodBody*, *ClassBodyDeclaration*, *Identifier*, *MethodDeclaration*]

Run == *MethodBody*
ManagedThreadClassBody == *Run* × seq *ClassBodyDeclaration*
ManagedThread == *Identifier* × *ManagedThreadClassBody*

HandleAsyncEvent == *MethodBody*
HandleAsyncLongEvent == *MethodBody*
EventHandlerClassBody == *HandleAsyncEvent* × seq *ClassBodyDeclaration*
OneShotEventHandler == *Identifier* × *EventHandlerClassBody*
LongEventHandlerClassBody == *HandleAsyncLongEvent* × seq *ClassBodyDeclaration*
AperiodicEventHandler ::= *apehType*⟨⟨*Identifier* × *EventHandlerClassBody*⟩⟩ |
 apehType⟨⟨*Identifier* × *LongEventHandlerClassBody*⟩⟩
PeriodicEventHandler == *Identifier* × *EventHandlerClassBody*
EventHandler ::= *pehDecl*⟨⟨*PeriodicEventHandler*⟩⟩ |
 apehDecl⟨⟨*AperiodicEventHandler*⟩⟩ |
 osehDecl⟨⟨*OneShotEventHandler*⟩⟩

GetNextMission == *MethodBody*
MissionSequencerClassBody == *GetNextMission* × seq *ClassBodyDeclaration*
MissionSequencer == *Identifier* × *MissionSequencerClassBody*

NestedMissionSequencer == *MissionSequencer*

SchedulableObject ::= *handler*⟨⟨*EventHandler*⟩⟩ |
 apeh⟨⟨*AperiodicEventHandler*⟩⟩ |
 nms⟨⟨*NestedMissionSequencer*⟩⟩

Cleanup == *MethodBody*
Initialize == *MethodBody*
MissionClassBody == *Initialize* × *Cleanup* × seq *ClassBodyDeclaration*
Mission == *Identifier* × *MissionClassBody*

Tier == *Mission* × seq *SchedulableObject*

$TopLevelMissionSequencer == MissionSequencer$

$ImmortalMemorySize == MethodDeclaration$

$InitializeApplication == MethodBody$

$GetSequencer == MethodBody$

$SafeletClassBody ==$

$GetSequencer \times InitializeApplication \times ImmortalMemorySize \times \text{seq } ClassBodyDeclaration$

$Safelet == Identifier \times SafeletClassBody$

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

section *TransSCJProg* **parents** *standard_toolkit, SCJBNFEncoding, CircusBNFEncoding*

$\text{safeletFW} : \text{Process}$
 $\text{topLevelMissionSequencerFW} : \text{Process}$
 $\text{controlTierSync} : \text{CSExpression}$
 $\text{Tier0} : N$

$\text{GenerateTierProc} : \text{seq Tier} \rightarrow \text{Process}$

$\text{GenerateFWProc} : \text{SCJProgram} \rightarrow \text{Process}$

$\forall \text{scj} : \text{SCJProgram} \bullet$
 $\exists s : \text{Safelet}; \text{tlms} : \text{TopLevelMissionSequencer}; \text{tiers} : \text{seq Tier} \bullet$
 $\text{scj} = (s, \text{tlms}, \text{tiers}) \wedge$
 $\exists \text{fwProc} : \text{Process}; \text{controlTierProc} : \text{Process}; \text{tierProc} : \text{Process} \mid$
 $\text{fwProc} = \text{procPar}(\text{controlTierProc}, \text{controlTierSync}, \text{tierProc}) \wedge$
 $\text{controlTierProc} =$
 $\quad \text{procPar}(\text{safeletFW}, \text{controlTierSync}, \text{topLevelMissionSequencerFW}) \wedge$
 $\text{tierProc} = \text{GenerateTierProc}(\text{tiers}) \bullet$
 $\text{GenerateFWProc}(\text{scj}) = \text{fwProc}$

$\text{GenerateAppProc} : \text{SCJProgram} \rightarrow \text{Process}$

$\text{GenerateMCBProc} : \text{SCJProgram} \rightarrow \text{Process}$

$\text{GenerateLockProc} : \text{SCJProgram} \rightarrow \text{Process}$

$\text{TransClasses} : \text{SCJProgram} \rightarrow \text{seq CircusParagraph}$

$TransSCJProg : Identifier \times SCJProgram \rightarrow CircusProgram$

$\forall scjProg : SCJProgram; name : Identifier \bullet$
 $\exists framework : CircusProgram; app : CircusProgram; mcb : CircusProgram;$
 $locking : CircusProgram; program : CircusProgram; n : N; p : Process;$
 $appComms : CSEExpression; mcbComms : CSEExpression; lockComms : CSEExpression$
 $fwProc : Process; appProc : Process; lockProc : Process; mcbProc : Process \mid$
 $app = TransClasses(scjProg) \wedge$
 $fwProc = GenerateFWProc(scjProg) \wedge$
 $appProc = GenerateAppProc(scjProg) \wedge$
 $mcbProc = GenerateMCBProc(scjProg) \wedge$
 $lockProc = GenerateLockProc(scjProg) \wedge$
 $program = \langle procDef(pd(n, procHide(procPar(procHide(procPar(procHide($
 $procPar(fwProc, appComms, appProc), appComms), mcbComms, mcbProc)$
 $, mcbComms), lockComms, lockProc), lockComms))) \rangle \bullet$
 $TransSCJProg(name, scjProg) =$
 $framework \wedge \langle procDef(pd(n, fwProc)) \rangle \wedge$
 $app \wedge \langle procDef(pd(n, appProc)) \rangle \wedge$
 $mcb \wedge \langle procDef(pd(n, mcbProc)) \rangle \wedge$
 $locking \wedge \langle procDef(pd(n, lockProc)) \rangle \wedge$
 $program$

Low Level

- $Method : MethodDeclaration \mapsto (Name, Params, ReturnType, Body) :$ translates an active application method into a *Circus* action
- $DataMethod : MethodDeclaration \mapsto :$ translates data methods into an *OhCircus* method
- $MethodBody : Block \mapsto \text{seq } CircExpression :$ translates a Java block, for example a method body
- $Registers : Block \mapsto \text{seq } Name :$ extracts the Names of the schedulables registered in a Java block
- $Returns : Block \mapsto \text{seq } Name :$ extracts the Names of the variables returned in a Java block
- $Variable : (Name, Type, InitExpression) \mapsto (CircName, CircType, CircExpression) :$ translates a variable
- $Parameters : (Name, Params, ReturnType, Body) \mapsto \text{seq } CircParam :$ translates a list of method parameters
- $\llbracket Name \rrbracket_{name} :$ translates the *name* to a Z identifier
- $\llbracket varType \rrbracket_{type} :$ translates types
- $\llbracket expr \rrbracket_{expression} :$ 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 $\llbracket Identifier \rrbracket_{Name} App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters} \mathbf{begin}$

State
 $this : \mathbf{ref} \llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
 $State'$
 $this := \mathbf{new} \llbracket Identifier \rrbracket_{name} Class()$

InitializeApplication $\hat{=}$

$$\left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ \llbracket \llbracket InitializeApplication \rrbracket_{Method} \rrbracket_{MethBody} \\ initializeApplicationRet \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

GetSequencer $\hat{=}$

$$\left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet ! \llbracket GetSequencer \rrbracket_{Returns} \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$$Methods \hat{=} \left(\begin{array}{l} GetSequencer \\ \square \\ InitializeApplication \\ \square \\ MethName(AppMeth_1) \\ \square \\ \dots \\ MethName(AppMeth_n) \\ \dots \end{array} \right) ; Methods$$

- $(Init ; Methods) \triangle (end_safelet_app \longrightarrow \mathbf{Skip})$

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 $\llbracket Identifier \rrbracket_{Name} App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
 $this : \text{ref } \llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
 $State'$
 $this := \text{new } \llbracket Identifier \rrbracket_{name} Class()$

$GetNextMission \hat{=} \text{var } ret : MissionID \bullet$

$$\left(\begin{array}{l} getNextMissionCall . IdOf(Identifier) \longrightarrow \\ ret := this . getNextMission(); \\ getNextMissionRet . IdOf(Identifier) ! ret \longrightarrow \\ \text{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$Methods \hat{=}$

$$\left(\begin{array}{l} GetNextMission \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

$\bullet (Init ; Methods) \triangle (end_sequencer_app . IdOf(Identifier) \longrightarrow \text{Skip})$

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 $\llbracket Identifier \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
 $this : \text{ref } \llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
 $State'$
 $this := \text{new } \llbracket Identifier \rrbracket_{name} Class()$

InitializePhase $\hat{=}$

$$\left(\begin{array}{l} initializeCall . IdOf(Identifier) \longrightarrow \\ \llbracket initialize \rrbracket_{Registers} initializeRet . IdOf(Identifier) \longrightarrow \\ \text{Skip} \end{array} \right)$$

CleanupPhase $\hat{=}$

$$\left(\begin{array}{l} cleanupMissionCall . IdOf(Identifier) \longrightarrow \\ cleanupMissionRet . IdOf(Identifier) ! \text{True} \longrightarrow \\ \text{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$Methods \hat{=}$
$$\left(\begin{array}{l} InitializePhase \\ \square \\ CleanupPhase \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right) ; Methods$$

• $(Init ; Methods) \triangle (end_mission_app . IdOf(Identifier) \longrightarrow \mathbf{Skip}$

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 $\llbracket PName \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
this : ref $\llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
State '
this := **new** $\llbracket Identifier \rrbracket_{name} Class()$

handleAsyncEvent $\hat{=}$

$$\left(\begin{array}{l} handleAsyncEventCall . IdOf(PName) \longrightarrow \\ \llbracket \llbracket HandleAsyncBody \rrbracket_{Method} \rrbracket_{MethBody}; \\ handleAsyncEventRet . IdOf(PName) \longrightarrow \\ \mathbf{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

Methods $\hat{=}$

$$\left(\begin{array}{l} handleAsyncEvent \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

• (*Init* ; *Methods*) $\triangle (end_ \llbracket HandlerTypeIdOf(PName) \rrbracket \longrightarrow \mathbf{Skip})$

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 }

```

process $\llbracket PName \rrbracket App \hat{=} \llbracket \llbracket ConstructorDeclaration \rrbracket_{Method} \rrbracket_{Parameters}$ **begin**

State
 $this : \text{ref } \llbracket Identifier \rrbracket_{name} Class$

state *State*

Init
 $State'$
 $this := \text{new } \llbracket Identifier \rrbracket_{name} Class()$

$Run \hat{=}$

$$\left(\begin{array}{l} runCall . IdOf(PName) \longrightarrow \\ \llbracket \llbracket run \rrbracket_{Method} \rrbracket_{MethBody}; \\ runRet . IfOf(PName) \longrightarrow \\ \text{Skip} \end{array} \right)$$

$\llbracket AppMeth_1 \rrbracket_{Method}$

...

$\llbracket AppMeth_n \rrbracket_{Method}$

$Methods \hat{=}$

$$\left(\begin{array}{l} Run \\ \square \\ MethName(AppMeth_1) \\ \square \\ MethName(AppMeth_n) \\ \dots \end{array} \right); Methods$$

• $(Init ; Methods) \triangle (end_managedThread_app . IdOf(PName) \longrightarrow \text{Skip})$

end

Data Class

class $\llbracket PName \rrbracket_{name}$ *Class* $\hat{=}$ **begin**

state *State*

$\llbracket VarName \rrbracket_{name} : \llbracket VarType \rrbracket_{type}$

state *State*

initial *Init*

State '

$\llbracket VarName \rrbracket'_{name} = \llbracket VarInit \rrbracket_{expression}$

$\llbracket DataMeth1 \rrbracket_{dataMeth}$

$\llbracket DataMeth2 \rrbracket_{dataMeth}$

...

• **Skip**

end