RoboScene to CSP Translation Rules

Holly Hendry

September 30, 2025

Rule 1. RoboScene model

$[\![\textbf{rsm} : \textbf{RoboSceneModel}]\!]_M \mathrel{\widehat{=}}$

$[\![\texttt{rsm.constant}]\!]_{CO}\ [\![\texttt{rsm.capabilities}]\!]_{ABS}$	(1)
$[\![\texttt{rsm.sequencediagramgroup}]\!]_{SDGS}\ [\![\texttt{rsm.types}]\!]_{TY}$	(2)

Rule 2. Constants

$[\![\textbf{varList} : \textbf{VariableList},\ \textbf{const} : \textbf{Constants},\ \textbf{timeunit} : \textbf{TimeUnit}]\!]_{CO} \mathrel{\widehat{=}}$

$[\![\texttt{varList}]\!]_{CON}\ \underline{(\text{if}\ \texttt{const} \neq \text{null then}\ [\![\texttt{const,timeunit}]\!]_{CO})}$	(1)

Rule 3. Constants List

$[\![\textbf{const} : \textbf{Variable},\ \textbf{constList} : \textbf{VariableList}]\!]_{CON} \mathrel{\widehat{=}}$

$[\![\texttt{const}]\!]_C\ \underline{(\text{if}\ \texttt{constList} \neq \text{null then}\ [\![\textit{const.constants}\texttt{constList}]\!]_{CON})}$	(1)

Rule 4. Constant Specification

$[\![\underline{(\text{const constID} = \text{val})} : \textbf{Variable}]\!]_C \mathrel{\widehat{=}}$

$\text{constID} = \text{val}$	(1)

Rule 5. Capabilities

$[\![\textbf{ab} : \textbf{ActorBlock},\ \textbf{abs} : \underline{\text{seq}}\ \textbf{ActorBlock}]\!]_{ABS} \mathrel{\widehat{=}}$

$[\![\texttt{ab}]\!]_{AB}\ \underline{(\text{if}\ \texttt{abs} \neq \text{null then}\ [\![\texttt{abs}]\!]_{ABS})}$	(1)

Rule 6. ActorBlock

$[\![\textbf{ab} : \textbf{ActorBlock}]\!]_{AB} \mathrel{\widehat{=}}$

$[\![\texttt{ab.outs}]\!]_{LES}\ [\![\texttt{ab.ins}]\!]_{LES}\ [\![\texttt{ab.variables}]\!]_{VARS}$	(1)

Rule 7. Lifeline Events

$[\![\textbf{event} : \textbf{LifelineEvents},\ \textbf{les} : \underline{\text{seq}}\ \textbf{LifelineEvents}]\!]_{LES} \mathrel{\widehat{=}}$

$[\![\texttt{varList}]\!]_{VAR}\ \underline{(\text{if}\ \texttt{varLists} \neq \text{null then}\ [\![\texttt{varLists}]\!]_{VARS})}$	(1)

Rule 8. Lifeline Out Event

$[\![\underline{(\text{out event} \rightarrow \text{actor})} : \texttt{LifelineEvents}]\!]_{LE} \mathrel{\widehat{=}}$

channel event	(1)

Rule 9. Typed Lifeline Out Event

$[\![\underline{(\text{out event} : \text{Type} \rightarrow \text{actor})} : \texttt{LifelineEvents}]\!]_{LE} \mathrel{\widehat{=}}$

channel event : $[\![\text{Type}]\!]_{TYPEEXP}$	(1)

Rule 10. Lifeline In Event

$[\![\underline{(\text{in event} \leftarrow \text{actor})} : \texttt{LifelineEvents}]\!]_{LE} \mathrel{\widehat{=}}$

channel event	(1)

Rule 11. Typed Lifeline In Event

$[\![\underline{(\text{in event} : \text{Type} \leftarrow \text{actor})} : \texttt{LifelineEvents}]\!]_{LE} \mathrel{\widehat{=}}$

channel event : $[\![\text{Type}]\!]_{TYPEEXP}$	(1)

Rule 12. Variable List

$[\![\texttt{varList} : \texttt{VariableList},\ \texttt{varLists} : \underline{\text{seq}}\ \texttt{VariableList}]\!]_{VARS} \mathrel{\widehat{=}}$

$[\![\texttt{varList}]\!]_{VAR}\ \underline{(\text{if}\ \texttt{varLists} \neq \text{null then}\ [\![\texttt{varLists}]\!]_{VARS})}$	(1)

Rule 13. Variable List

$[\![\texttt{var} : \texttt{Variable},\ \texttt{varList} : \texttt{VariableList}]\!]_{VAR} \mathrel{\widehat{=}}$

$[\![\texttt{var}]\!]_{V}\ \underline{(\text{if}\ \texttt{varList} \neq \text{null then}\ [\![\texttt{varList}]\!]_{VAR})}$	(1)

Rule 14. Variable Specification

$[\![\underline{(\text{var} : \text{Type})} : \texttt{Variable}]\!]_{V} \mathrel{\widehat{=}}$

channel *set*_var : $[\![\text{Type}]\!]_{TYPEEXP}$	(1)
channel *get*_var : $[\![\text{Type}]\!]_{TYPEEXP}$	(2)

Rule 15. Sequence Diagram Groups

$[\![$ **sdg** : **SequenceDiagramGroup**, **sdgs** : seq **SequenceDiagramGroup** $]\!]_{SDGS} =$

$[\![\texttt{sdg}]\!]_{SDG}$ (if sdgs $\neq$ null then $[\![\texttt{sdqs}]\!]_{SDGS}$) (1)

Rule 16. Sequence Diagram Group

$[\![$ **s** : **Sequence**, **sdg** : **SequenceDiagramGroup** $]\!]_{SDG} =$

$[\![\texttt{s}]\!]_S$ (if sdg $\neq$ null then $[\![\texttt{sdq}]\!]_{SDG}$) (1)

Rule 17. Sequence

$[\![$ **s** : **Sequence** $]\!]_S \mathrel{\widehat{=}}$

$((((\|$ a : actors • $[\![\texttt{a; s.fragment}]\!]_{\text{A}}$ $[\![\texttt{a; s.fragment}]\!]_{\text{LIF}})$ (1)

$|[$ {| *str*, *par*, *terminate* |} $]|$ $[\![\texttt{parFrags;strFrags}]\!]_{CTRL})$ (2)

$|[$ {| *alt*, *opt*, *loop*, *guard*, *terminate* |} $]|$
$[\![\texttt{altFrags;loopFrags;optFrags}]\!]_{GRD})$ (3)

$|[$ $[\![s.variable]\!]_{SVARS}$ $\cup$ {| *terminate* |}$]|$ $[\![s.variable]\!]_{MEMV})$ (4)

$\setminus$ {| *alt*, *opt*, *loop*, *guard*, *str*, *par* |} (5)

datatype IDs = *ID_ALT* | *ID_OPT* | *ID_LOOP* | *ID_PAR* | *ID_STR* (6)

where

actors = s.actor and parFrags = par(s.fragment) and (7)

strFrags = str(s.fragment) and altFrags = alt(s.fragment) and (8)

optFrags = opt(s.fragment) and loopFrags = loop(s.fragment) (9)

Rule 18. Actor Alphabet

$[\![$ **a** : **Actor**, **frags** : seq **InteractionFragment** $]\!]_A \mathrel{\widehat{=}}$

{| actorCaps(a) |} (1)

$\cup$ {| f : actFragsL(frags, a) • *loop.ID_LOOP*.id(f), *guard.ID_LOOP*.id(f) |} (2)

$\cup$ {| f : actFragsA(frags, a) • *alt.ID_ALT*.id(f), *guard.ID_ALT*.id(f) |} (3)

$\cup$ {| f : actFragsO(frags, a) • *opt.ID_OPT*.id(f), *guard.ID_OPT*.id(f) |} (4)

$\cup$ {| f : actFragsP(frags, a) • *par.ID_PAR*.id(f) |} (5)

$\cup$ {| f : actFragsS(frags, a) • *str.ID_STR*.id(f) |} (6)

$\cup$ {| *terminate* |} (7)

Rule 19. Actor Lifeline

$[\![\texttt{a} : \textbf{Actor},\ \textbf{frags} : \underline{\text{seq}}\ \textbf{InteractionFragment}]\!]_{LIF} \ \widehat{=}$

$$\mathop{;}_{9}\ \texttt{frag} : \underline{\text{actFrags}(\texttt{frags})} \bullet [\![\texttt{frag}]\!]^{F}_{(\texttt{a})} \tag{1}$$

Rule 20. Actor Control Fragments

$[\![\textbf{parFrag} : \textbf{ParFragment}]\!]^{F}_{(a)} =$

$$par.ID\_PAR.\underline{\text{id}(\texttt{parFrag})} \rightarrow \textbf{\textit{Skip}} \tag{1}$$

Rule 21. Actor Control Fragments

$[\![\textbf{strFrag} : \textbf{StrFragment}]\!]^{F}_{(a)} =$

$$str.ID\_STR.\underline{\text{id}(\texttt{strFrag})} \rightarrow \textbf{\textit{Skip}} \tag{1}$$

Rule 22. Actor Guarded Fragments

$[\![\textbf{altFrag} : \textbf{AltFragment}]\!]^{F}_{(\textbf{a})} =$

$$[\![\texttt{altFrag}, \texttt{a}]\!]_{AF} \tag{1}$$

Rule 23. Actor Guarded Fragments

$[\![\textbf{altFrag} : \textbf{AltFragment}]\!]^{F}_{(\textbf{a})} =$

$$[\![\texttt{optFrag}, \texttt{a}]\!]_{OF} \tag{1}$$

Rule 24. Actor Guarded Fragments

$[\![\textbf{altFrag} : \textbf{AltFragment}]\!]^{F}_{(\textbf{a})} =$

$$[\![\texttt{loopFrag}, \texttt{a}]\!]_{LF} \tag{1}$$

Rule 25. Control

$[\![\textbf{parFrags, strFrags}]\!]_{CTRL} \ \widehat{=}$

$$[\![\texttt{parFrags}]\!]_{PAR} \ |[\ \{terminate\}\ ]|\ [\![\texttt{strFrags}]\!]_{STR} \tag{1}$$

$$\textbf{\textit{channel}}\ par :\ IDs.Int \tag{2}$$

$$\textbf{\textit{channel}}\ str :\ IDs.Int \tag{3}$$

Rule 26. Parallel

$[\![\texttt{parFrags}]\!]_{PAR} \mathrel{\widehat{=}}$

let

$Parallel =$ (1)

$par.ID\_PAR?id \rightarrow$ (2)

$(\Box\ \texttt{parFrag} : \texttt{parFrags} \bullet$ (3)

$(id == \underline{\text{id}(\texttt{parFrag})})\&$ (4)

$(\Big|\Big|\ \texttt{branch} : \texttt{parFrag.branches} \bullet [\![\texttt{parFrag}, \texttt{branch}]\!]_{PT})$ (5)

$)\ {}^{\circ}_{9}\ Parallel$ (6)

$\Box\ terminate \rightarrow \textbf{\textit{Skip}}$ (7)

within

$Parallel$ (8)

Rule 27. Parallel Thread

$[\![\texttt{parFrag, branch}]\!]_{PT} \mathrel{\widehat{=}}$

$\Big|\Big|\ \texttt{actor} : \texttt{branch.actors} \bullet$ (1)

$[\![\texttt{actor}, \texttt{parFrag.branch.fragments}]\!]_{A}\ [\![\texttt{actor}, \texttt{parFrag}, \texttt{branch}]\!]_{PTA}$ (2)

Rule 28. Parallel Thread per Actor

$[\![\texttt{actor}, \texttt{parFrag}, \texttt{branch}]\!]_{PTA} \mathrel{\widehat{=}}$

$[\![\texttt{actor}, \texttt{parFrag.branch.fragments}]\!]_{LIF}$ (1)

Rule 29. Strict

$[\![\texttt{strFrags}]\!]_{STR} \mathrel{\widehat{=}}$

let

$Strict =$ (1)

$str.ID\_STR?id \rightarrow$ (2)

$(\Box\ \texttt{strFrag} : \texttt{strFrags} \bullet$ (3)

$(id == \underline{\text{id}(\texttt{strFrag})})\&[\![\texttt{strFrag}]\!]_{SF})$ (4)

$)\ {}^{\circ}_{9}\ Strict$ (5)

$\Box\ terminate \rightarrow \textbf{\textit{Skip}}$ (6)

within

$Strict$ (7)

Rule 30. Strict Fragment

$[\![\texttt{strFrag}]\!]_{SF} \mathrel{\widehat{=}}$

${}^{\circ}_{9}\ \texttt{frag} : \texttt{strFrag.fragments} \bullet [\![\texttt{frag}]\!]^{F}_{(\texttt{a})}$ (1)

Rule 31. Guard

$[\![$ **altFrags, loopFrags, optFrags** $]\!]_{GRD} \mathrel{\widehat{=}}$

$$([\![\texttt{altFrags, loopFrags, optFrags}]\!]_{EVAL} \tag{1}$$
$$\quad |[\ alphaCounters \cup \{|\ terminate\ |\}\ ]|\ [\![\texttt{loopFrags}]\!]_{CNT}) \setminus alphaCounters \tag{2}$$

$$\textbf{\textit{channel}}\ alt : IDs.Int \tag{3}$$
$$\textbf{\textit{channel}}\ opt : IDs.Int \tag{4}$$
$$\textbf{\textit{channel}}\ loop : IDs.Int \tag{5}$$
$$\textbf{\textit{channel}}\ guard : IDs.Int.Int.Bool \tag{6}$$

$$\textbf{\textit{channel}}\ getCount : IDs.Int.Int \tag{7}$$
$$\textbf{\textit{channel}}\ setCount : IDs.Int.Int \tag{8}$$
$$alphaCounters = \{|\ getCount, setCount\ |\} \tag{9}$$

Rule 32. Count

$[\![$ **loopFrags** $]\!]_{CNT} \mathrel{\widehat{=}}$

$$(\Big|\Big|\ \mathsf{loopID} : \mathsf{loopIDs} \bullet [\{|\ terminate\ |\}]\ [\![\mathsf{loopID}]\!]_{CNTID}) \tag{1}$$

where

$$\mathsf{loopIDs} = \underline{\mathrm{ids}}(\texttt{loopFrags}) \tag{2}$$

Rule 33. Count per Fragment

$[\![\mathsf{loopID}]\!]_{CNTID} \mathrel{\widehat{=}}$

let
$$Counter(count) = \tag{1}$$
$$\quad getCount.ID\_LOOP.\mathsf{loopID}!count \rightarrow Counter(count) \tag{2}$$
$$\quad \Box\ setCount.ID\_LOOP.\mathsf{loopID}?y \rightarrow Counter(y) \tag{3}$$
$$\quad \Box\ terminate \rightarrow \textbf{\textit{Skip}} \tag{4}$$
within
$$Counter(0) \tag{5}$$

Rule 34. Evaluation

$[\![$ **altFrags, loopFrags, optFrags** $]\!]_{EVAL} \mathrel{\widehat{=}}$

$$[\![\texttt{loopFrags}]\!]_{RES}\ \fatsemi\ [\![\texttt{altFrags, loopFrags, optFrags}]\!]_{GRSP} \tag{1}$$

Rule 35. Reset Counters

$[\![\texttt{loopFrags}]\!]_{RES} \mathrel{\widehat{=}}$

$\fatsemi$ loopID : loopIDs • $setCount.ID\_LOOP$.loopID!0 $\rightarrow$ ***Skip*** (1)

where

loopIDs = ids(`loopFrags`) (2)

Rule 36. Guard Response

$[\![\texttt{altFrags, loopFrags, optFrags}]\!]_{GRSP} \mathrel{\widehat{=}}$

let

$GuardResponse =$ (1)

$\Box$ $alt?ID\_ALT.id \rightarrow$ ((2)

$\Box$ `altFrag` : `altFrags` • $(id ==$ id(`altFrag`))&($[\![\texttt{altFrag}]\!]_{AEV}$) (3)

) $\fatsemi$ $GuardResponse$ (4)

$\Box$ $opt?ID\_OPT.id \rightarrow$ ((5)

$\Box$ `optFrag` : `optFrags` • $(id ==$ id(`optFrag`))&($[\![\texttt{optFrag}]\!]_{OEV}$) (6)

) $\fatsemi$ $GuardResponse$ (7)

$\Box$ $loop?ID\_LOOP.id \rightarrow$ ((8)

$\Box$ `loopFrag` : `loopFrags` • $(id ==$ id(`loopFrag`))&($[\![\texttt{loopFrag}]\!]_{LEV}$) (9)

) $\fatsemi$ $GuardResponse$ (10)

$\Box$ $terminate \rightarrow$ ***Skip*** (11)

within

$GuardResponse$ (12)

Rule 37. Alt Fragment Evaluation

$[\![\texttt{altFrag}]\!]_{AEV} \mathrel{\widehat{=}}$

($\rightarrow$ `variable` : guardVariables • ($get\_$`variable`)) (1)

$\rightarrow$ ($[\![\texttt{altFrag}]\!]_{G}$) (2)

where

guardVariables = getFragmentGuardVariables(`altFrag`) (3)

Rule 38. Opt Fragment Evaluation

optFrag$_{OEV} \widehat{=}$

(1) $(\rightarrow$ `variable` : guardVariables • $(get\_$`variable` $))$

(2) $\rightarrow ([\![$`optFrag`$]\!]_G)$

where

(3) guardVariables = getFragmentGuardVariables(`optFrag`)

Rule 39. Loop Fragment Evaluation

loopFrag$_{LEV} \widehat{=}$

(1) $(\rightarrow$ `variable` : guardVariables • $(get\_$`variable` $))$

(2) $\rightarrow ([\![$`loopFrag`$]\!]_G)$

where

(3) guardVariables = getFragmentGuardVariables(`loopFrag`)

Rule 40. Alt Fragment Guard

$[\![$(alt [g_1$_0$&g_1$_1$& . . . &g_1$_n$] x_1 else . . . else [g_n$_0$&g_n$_1$& . . . &g_n$_n$] x_n else x_m end) : **AltFragment**$]\!]_G$ =

(1) (g_1$_0$&g_1$_1$& . . . &g_1$_n$)&($guard.ID\_ALT$.id(`altFrag`).1!*true* → . . .

(2) → $guard.ID\_ALT$.id(`altFrag`).n!*false*)

(3) □

(4) . . .

(5) □

(6) (g_n$_0$&g_n$_1$& . . . &g_n$_n$)&($guard.ID\_ALT$.id(`altFrag`).1!*false* → . . .

(7) → $guard.ID\_ALT$.id(`altFrag`).n!*true*)

(8) □

(9) *not*((g_1$_0$&g_1$_1$&...&g_1$_n$) *or* . . . *or* (g_n$_0$&g_n$_1$&...&g_n$_n$))&

(10) ($guard.ID\_ALT$.id(`altFrag`).1!*false* → . . . →

(11) $guard.ID\_ALT$.id(`altFrag`).n!*false*)

Rule 41. Alt Fragment Guard

$[\![(\text{alt } [\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n]\ \mathrm{x}_1 \text{ else } \ldots \text{ else } [\mathsf{g\_n}_0\&\mathsf{g\_n}_1\&\ldots\&\mathsf{g\_n}_n]\ \mathrm{x}_n \text{ end}) : \mathbf{AltFragment}]\!]_G =$

$(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!true \rightarrow \ldots$	(1)
$\quad \rightarrow guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.\mathsf{n}!false)$	(2)
$\square$	(3)
$\ldots$	(4)
$\square$	(5)
$(\mathsf{g\_n}_0\&\mathsf{g\_n}_1\&\ldots\&\mathsf{g\_n}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!false \rightarrow \ldots$	(6)
$\quad \rightarrow guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.\mathsf{n}!true)$	(7)
$\square$	(8)
$not((\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&...\&\mathsf{g\_1}_n)\ or\ \ldots\ or\ (\mathsf{g\_n}_0\&\mathsf{g\_n}_1\&...\&\mathsf{g\_n}_n))\&$	(9)
$\quad (guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!false \rightarrow \ldots \rightarrow$	(10)
$\quad guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.n!false)$	(11)

Rule 42. Alt Fragment Guard

$[\![\underline{(\text{alt } \mathrm{x}_1 \text{ else } \ldots \text{ else } \mathrm{x}_n \text{ end})} : \mathbf{AltFragment}]\!]_G =$

$(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!true \rightarrow \ldots$	(1)
$\quad \rightarrow guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.\mathsf{n}!false)$	(2)
$\sqcap$	(3)
$\ldots$	(4)
$\sqcap$	(5)
$(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!false \rightarrow \ldots$	(6)
$\quad \rightarrow guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.\mathsf{n}!true)$	(7)[2pt]

Rule 43. Alt Fragment Guard

$[\![(\text{alt } [\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n]\ \mathrm{x}_1 \text{ else } \mathrm{x}_2 \text{ end}) : \mathbf{AltFragment}]\!]_G =$

$(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!true$	(1)
$\square$	(2)
$not(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&...\&\mathsf{g\_1}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!false$	(4)

Rule 44. Alt Fragment Guard

$[\![(\text{alt } [\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n]\ \mathrm{x}_1 \text{ end}) : \mathbf{AltFragment}]\!]_G =$

$(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!true$	(1)
$\square$	(2)
$not(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&...\&\mathsf{g\_1}_n)\&(guard.ID\_ALT.\underline{\mathrm{id}(\texttt{altFrag})}.1!false$	(4)

Rule 45. Opt Fragment Guard

$[\![\underline{(\text{opt } [\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n] \text{ x}_1 \text{ end})} : \texttt{OptFragment}]\!]_G \; =$

$(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&\ldots\&\mathsf{g\_1}_n)\&(guard.ID\_OPT.\underline{\text{id}(\texttt{optFrag})}.1!true$	(1)
$\square$	(2)
$not(\mathsf{g\_1}_0\&\mathsf{g\_1}_1\&...\&\mathsf{g\_1}_n)\&(guard.ID\_OPT.\underline{\text{id}(\texttt{optFrag})}.1!false$	(4)

Rule 46. Loop Fragment Guard

$[\![\underline{(\text{loop}(\mathsf{min},\mathsf{max})[\mathsf{condition}] \text{ x}_1 \text{ end})} : \texttt{LoopFragment}]\!]_G =$

$not(\mathsf{condition})\&$	(1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$	(2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$	(3)
$\square$	(4)
$((count < \mathsf{min})\ and\ \mathsf{condition})\&$	(5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count+1) \rightarrow$	(6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}})$	(7)
$\square$	(8)
$(count == \mathsf{max})\&$	(9)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$	(10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$	(11)
$\square$	(12)
$((count >= \mathsf{min})\ and\ (count < \mathsf{max})\ and\ \mathsf{condition})\&$	(13)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count+1) \rightarrow$	(14)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$	(15)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$	(16)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})))$	(17)

Rule 47. Loop Fragment Guard

$[\![\underline{(\text{loop}(\mathsf{min}, \mathsf{max})\ \mathsf{x}_1\ \text{end})} : \texttt{LoopFragment}]\!]_G =$

$(count < \mathsf{min})\&$ (1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}})$ (3)
$\square$ (4)
$(count == \mathsf{max})\&$ (5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (7)
$\square$ (8)
$((count >= \mathsf{min})\ and\ (count < \mathsf{max}))\&$ (9)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$ (11)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (12)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}}))$ (13)

Rule 48. Loop Fragment Guard

$[\![\underline{(\text{loop}(\mathsf{min})[\mathsf{condition}]\ \mathsf{x}_1\ \text{end})} : \texttt{LoopFragment}]\!]_G =$

$not(\mathsf{condition})\&$ (1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (3)
$\square$ (4)
$((count < \mathsf{min})\ and\ \mathsf{condition})\&$ (5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}})$ (7)
$\square$ (8)
$(count == \underline{\text{defaultMax}})\&$ (9)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (11)
$\square$ (12)
$((count >= \mathsf{min})\ and\ (count < \underline{\text{defaultMax}})\ and\ \mathsf{condition})\&$ (13)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (14)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$ (15)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (16)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}}))$ (17)

Rule 49. Loop Fragment Guard

$[\![\underline{(\text{loop}[\text{condition}]\ \text{x}_1\ \text{end})} : \texttt{LoopFragment}]\!]_G =$

$not(\text{condition})\&$ (1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (3)
$\Box$ (4)
$(count == \underline{\text{defaultMax}})\&$ (5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (7)
$\Box$ (8)
$((count < \underline{\text{defaultMax}})\ and\ \text{condition})\&$ (9)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$ (11)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (12)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}}))$ (13)

Rule 50. Loop Fragment Guard

$[\![\underline{(\text{loop}(\text{min})\ \text{x}_1\ \text{end})} : \texttt{LoopFragment}]\!]_G =$

$(count < \text{min})\&$ (1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}})$ (3)
$\Box$ (4)
$(count == \underline{\text{defaultMax}})\&$ (5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$ (7)
$\Box$ (8)
$((count >= \text{min})\ and\ (count < \underline{\text{defaultMax}}))\&$ (9)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$ (10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$ (11)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$ (12)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}}))$ (13)

Rule 51. Loop Fragment Guard

$[\![(\underline{\text{loop } \mathsf{x}_1 \text{ end}} : \texttt{LoopFragment}]\!]_G =$

$(count < \underline{\text{defaultMin}})\&$	(1)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$	(2)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}})$	(3)
$\square$	(4)
$(count == \underline{\text{defaultMax}})\&$	(5)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$	(6)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})$	(7)
$\square$	(8)
$((count >= \underline{\text{defaultMin}})\ and\ (count < \underline{\text{defaultMax}}))\&$	(9)
$((setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!(count + 1) \rightarrow$	(10)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!true \rightarrow \textbf{\textit{Skip}}) \sqcap$	(11)
$(setCount.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}!0 \rightarrow$	(12)
$guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1!false \rightarrow \textbf{\textit{Skip}})))$	(13)

Rule 52. Alt Fragment

$[\![(\underline{\text{alt } [\text{g\_1}_0\&\text{g\_1}_1\&\ldots\&\text{g\_1}_\text{n}]\ \mathsf{x}_1 \text{ else } \ldots \text{ else } [\text{g\_n}_0\&\text{g\_n}_1\&\ldots\&\text{g\_n}_\text{n}]\ \mathsf{x}_\mathsf{n} \text{ else } \mathsf{x}_\mathsf{m} \text{ end}} : \texttt{AltFragment},\ \texttt{a} : \texttt{Actor}]\!]_{AF} =$

$alt.ID\_ALT.\underline{\text{id}(\texttt{altFrag})} \rightarrow guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?id1$	(1)
$\rightarrow \ldots \rightarrow guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?idn$	(2)
$\rightarrow (id1\&([\![\text{a}, \mathsf{x}_1]\!]_{LIF}) \ \square \ \ldots \ \square \ idn\&([\![\text{a}, \mathsf{x}_\mathsf{n}]\!]_{LIF})$	(3)
$\square \ not(id1 \ or \ \ldots \ or \ idn)\&([\![\text{a}, \mathsf{x}_\mathsf{m}]\!]_{LIF}))$	(4)

Rule 53. Alt Fragment

$[\![(\underline{\text{alt } \mathsf{x}_1 \text{ else } \ldots \text{ else } \mathsf{x}_\mathsf{n} \text{ end}} : \texttt{AltFragment},\ \texttt{a} : \texttt{Actor}]\!]_{AF} =$

$alt.ID\_ALT.\underline{\text{id}(\texttt{altFrag})} \rightarrow guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?id1$	(1)
$\rightarrow \ldots \rightarrow guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?idn$	(2)
$\rightarrow (id1\&([\![\text{a}, \mathsf{x}_1]\!]_{LIF}) \ \square \ \ldots \ \square \ idn\&([\![\text{a}, \mathsf{x}_\mathsf{n}]\!]_{LIF})$	

Rule 54. Alt Fragment

$[\![(\underline{\text{alt } [\text{g\_1}_0\&\text{g\_1}_1\&\ldots\&\text{g\_1}_\text{n}]\ \text{x}_1 \text{ else } \text{x}_2 \text{ end}}) : \mathbf{AltFragment},\ \mathbf{a} : \mathbf{Actor}]\!]_{AF} \ =$

$alt.ID\_ALT.\underline{\text{id}(\texttt{altFrag})} \ \rightarrow \ guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, \text{x}_1]\!]_{LIF})$ (3)

$\quad \square \ not(id1)\&([\![\text{a}, \text{x}_2]\!]_{LIF}))$ (4)

Rule 55. Alt Fragment

$[\![(\underline{\text{alt } [\text{g\_1}_0\&\text{g\_1}_1\&\ldots\&\text{g\_1}_\text{n}]\ \text{x}_1 \text{ end}}) : \mathbf{AltFragment},\ \mathbf{a} : \mathbf{Actor}]\!]_{AF} \ =$

$alt.ID\_ALT.\underline{\text{id}(\texttt{altFrag})} \ \rightarrow \ guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, \text{x}_1]\!]_{LIF})$ (3)

$\quad \square \ not(id1)\&(\textbf{\textit{Skip}}))$ (4)

Rule 56. Alt Fragment

$[\![(\underline{\text{alt } [\text{g\_1}_0\&\text{g\_1}_1\&\ldots\&\text{g\_1}_\text{n}]\ \text{x}_1 \text{ else } \ldots \text{ else } [\text{g\_n}_0\&\text{g\_n}_1\&\ldots\&\text{g\_n}_\text{n}]\ \text{x}_\text{n} \text{ end}}) : \mathbf{AltFragment},\ \mathbf{a} : \mathbf{Actor}]\!]_{AF} \ =$

$alt.ID\_ALT.\underline{\text{id}(\texttt{altFrag})} \ \rightarrow \ guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?id1$ (1)

$\rightarrow \ \ldots \ \rightarrow \ guard.ID\_ALT.\underline{\text{id}(\texttt{altFrag})}.1?idn$ (2)

$\rightarrow \ (id1\&([\![\text{a}, \text{x}_1]\!]_{LIF}) \ \square \ \ldots \ \square \ idn\&([\![\text{a}, \text{x}_\text{n}]\!]_{LIF})$ (3)

$\quad \square \ not(id1 \ or \ \ldots \ or \ idn)\&(\textbf{\textit{Skip}}))$ (4)

Rule 57. Opt Fragment

$[\![(\underline{\text{opt } [\text{g\_1}_0\&\text{g\_1}_1\&\ldots\&\text{g\_1}_\text{n}]\ \text{x}_1 \text{ end}}) : \mathbf{OptFragment},\ \mathbf{a} : \mathbf{Actor}]\!]_{OF} \ =$

$opt.ID\_OPT.\underline{\text{id}(\texttt{optFrag})} \ \rightarrow \ guard.ID\_OPT.\underline{\text{id}(\texttt{optFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, \text{x}_1]\!]_{LIF}) \ \square \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 58. Loop Fragment

$[\![(\underline{\text{loop } \text{x}_1 \text{ end}}) : \mathbf{LoopFragment},\ \mathbf{a} : \mathbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, \text{x}_1]\!]_{LIF}) \ \square \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 59. Loop Fragment

$[\![(\underline{\text{loop[condition] } x_1 \text{ end}}) : \textbf{LoopFragment},\ \textbf{a} : \textbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, x_1]\!]_{LIF}) \ \Box \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 60. Loop Fragment

$[\![(\underline{\text{loop(min) } x_1 \text{ end}}) : \textbf{LoopFragment},\ \textbf{a} : \textbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, x_1]\!]_{LIF}) \ \Box \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 61. Loop Fragment

$[\![(\underline{\text{loop(min)[condition] } x_1 \text{ end}}) : \textbf{LoopFragment},\ \textbf{a} : \textbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, x_1]\!]_{LIF}) \ \Box \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 62. Loop Fragment

$[\![(\underline{\text{loop(min, max) } x_1 \text{ end}}) : \textbf{LoopFragment},\ \textbf{a} : \textbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, x_1]\!]_{LIF}) \ \Box \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 63. Loop Fragment

$[\![(\underline{\text{loop(min, max)[condition] } x_1 \text{ end}}) : \textbf{LoopFragment},\ \textbf{a} : \textbf{Actor}]\!]_{LF} \ =$

$loop.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})} \ \rightarrow \ guard.ID\_LOOP.\underline{\text{id}(\texttt{loopFrag})}.1?id1$ (1)

$\rightarrow \ (id1\&([\![\text{a}, x_1]\!]_{LIF}) \ \Box \ not(id1)\&(\textbf{\textit{Skip}}))$ (2)

Rule 64. Shared Variables List

$[\![\textbf{vars} : \underline{\text{seq}}\ \textbf{VariableList}]\!]_{SVARS} \ \widehat{=}$

$\bigcup\{\texttt{vs} : \underline{\text{ran}}\ \texttt{vars} \ \bullet \ [\![\texttt{vs.vars}]\!]_{SVAR}\}$ (1)

Rule 65. Shared Variables

$[\![\textbf{vars} : \underline{\text{seq}}\ \textbf{Variable}]\!]_{SVAR} \ \widehat{=}$

$\{|\ \underline{\texttt{v : vars} \ \bullet}\ get\_\underline{\text{id}(\texttt{v})}, set\_\underline{\text{id}(\texttt{v})}\ |\}$ (1)

Rule 66. Memory Variable List

$[\![\textbf{varList} : \textbf{VariableList},\ \textbf{varLists} : \underline{\text{seq}}\ \textbf{VariableList}]\!]_{MEMV} \ \widehat{=}$

$[\![\texttt{varList}]\!]_{MEM}\ (\underline{\text{if varLists} \neq \text{null then}}\ [\![\texttt{varLists}]\!]_{MEMV})$ (1)

Rule 67. Memory

$[\![\texttt{varList} : \texttt{VariableList}]\!]_{MEM} \mathrel{\widehat{=}}$

$(\Big\|\ \texttt{var} : \texttt{varList.vars} \bullet [\{\!|\ terminate\ |\!\}]\ [\![\texttt{var}]\!]_{MEMVARS})$ (1)

Rule 68. Variable Memory

$[\![\texttt{var}]\!]_{MEMVARS} \mathrel{\widehat{=}}$

let
$Memory(var) =$ (1)
$get\_\mathsf{id}!var \rightarrow Memory(var)$ (2)
$\Box\ set\_\mathsf{id}?x \rightarrow Memory(var)$ (3)
$\Box\ terminate \rightarrow \textbf{\textit{Skip}}$ (4)
within
$Memory(\underline{\text{initial}(\texttt{var})})$ (5)

where
$\mathsf{id} = \underline{\text{id}(\texttt{var})}$ (6)

Rule 69. Deterministic Wait

$[\![\underline{(\text{wait}(\mathsf{x})\ \text{on a})} : \texttt{WaitOccurrence}]\!]^F_{(a)} =$

$wait(\mathsf{x})$ (1)

Rule 70. Nondeterministic Wait

$[\![\underline{(\text{wait}([\mathsf{v},\mathsf{y}])\ \text{on a})} : \texttt{WaitOccurrence}]\!]^F_{(a)} =$

$(\sqcap\ x : \{\mathsf{v}..\mathsf{y}\} \bullet wait(x))$ (1)

Rule 71. Deadline

$[\![\underline{(\text{deadline}(\mathsf{x})\ \text{b})} : \texttt{DeadlineFragment}]\!]^F_{(a)} =$

$Deadline([\![\mathtt{a},\mathtt{b}]\!]_{LIF}, \mathsf{x})$ (1)

Rule 72. Destroy

$[\![(\underline{\text{destroy on a}}) : \texttt{DestroyOccurrence}]\!]^{F}_{(a)} \;=$

terminate $\rightarrow$ ***Skip***	(1)

Rule 73. Ignore

$[\![(\underline{\text{ignore}(\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_n)\ \mathsf{b}}) : \texttt{IgnoreFragment}]\!]^{F}_{(a)} \;=$

$(\mathsf{b}) \setminus \{\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_n\}$	(1)

Rule 74. Consider

$[\![(\underline{\text{consider}(\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_n)\ \mathsf{b}}) : \texttt{IgnoreFragment}]\!]^{F}_{(a)} \;=$

$(\mathsf{b}) \setminus \{ALL \setminus \{\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_n\}\}$	(1)
where	
$\mathsf{ALL} = \underline{\text{allFrags(b)}}$	(2)

Rule 75. Message - Source Actor

$[\![(\underline{\text{a} \rightarrow \text{a}_1 : \text{mID}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \;=$

$\mathsf{mID} \rightarrow$ ***Skip***	(1)

Rule 76. Message - Target Actor

$[\![(\underline{\text{a}_1 \rightarrow \text{a} : \text{mID}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \;=$

$\mathsf{mID} \rightarrow$ ***Skip***	(1)

Rule 77. Message With Output Constant - Source Actor

$[\![(\underline{\text{a} \rightarrow \text{a}_1 : \text{mID!var}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \;=$

$\mathsf{mID!var} \rightarrow$ ***Skip***	(1)

Rule 78. Message With Output Constant - Target Actor

$[\![(\underline{\mathrm{a}_1 \ \rightarrow \ \mathrm{a} : \mathrm{mID!var}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID?var $\rightarrow$ *set*_var!var $\rightarrow$ ***Skip*** (1)

Rule 79. Message With Output Literal - Source Actor

$[\![(\underline{\mathrm{a} \ \rightarrow \ \mathrm{a}_1 : \mathrm{mID!val}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID!val $\rightarrow$ ***Skip*** (1)

Rule 80. Message With Output Literal - Target Actor

$[\![(\underline{\mathrm{a}_1 \ \rightarrow \ \mathrm{a} : \mathrm{mID!val}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID?var $\rightarrow$ *set*_var!var $\rightarrow$ ***Skip*** (1)

where

var $= \underline{\mathrm{var}(\mathtt{val})}$ (2)

Rule 81. Message With Input - Source Actor

$[\![(\underline{\mathrm{a} \ \rightarrow \ \mathrm{a}_1 : \mathrm{mID?var}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID?var $\rightarrow$ *set*_var!var $\rightarrow$ ***Skip*** (1)

Rule 82. Message With Input - Target Actor

$[\![(\underline{\mathrm{a}_1 \ \rightarrow \ \mathrm{a} : \mathrm{mID?var}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID!var $\rightarrow$ ***Skip*** (1)

Rule 83. Self Message Actor

$[\![(\underline{\mathrm{a} \ \rightarrow \ \mathrm{a} : \mathrm{mID?var}}) : \texttt{InteractionFragment}]\!]^{F}_{(a)} \ =$

mID?var $\rightarrow$ *set*_var!var $\rightarrow$ ***Skip*** (1)