poST Language Transformational Semantics

I. Preliminaries

A. Transformation Relation

A transformation relation is defined as a family of relations $\mapsto_i \in (Po \times C) \times ((Pr \cup \{\Lambda\}) \times C)$, where Po is a set of fragments of poST programs, Pr is a set of fragments of Promela programs, C is a context in which a transformation is executed.

B. Kinds of Transformation Relations

The transformation relation is divided into the following indexed relations:

- $\mapsto_A$ transforms control algorithm based on a sequence of poST programs;
- $\mapsto_{Pg}$ transforms poST programs and sequences of poST programs;
- $\mapsto_T$ transforms types;
- $\mapsto_V$ transforms variable declarations;
- $\mapsto_E$ transforms expressions;
- $\mapsto_{St}$ transforms statements and statement sequences;
- $\mapsto_P$ transforms process declarations and process declaration sequences;
- $\mapsto_S$ transforms state declarations and state declaration sequences;
- $\mapsto_R$ generates Promela names for the corresponding names of variables, processes and process states of the poST program.

C. Context Attributes

The transformation context is specified by the following attributes:

- $VN(n, p, pg)$ returns the name in Promela program corresponding to the name n of a variable v in process p of program pg. If $p = \bot$, v is defined in pg outside processes;
- $PN(n, pg)$ returns the name in Promela program corresponding to the name n of a process in program pg;
- $SN(n, p, pg)$ returns the name in Promela program corresponding to the name n of a state of process p of program pg;
- CB is a fragment of Promela program that is a result of transformation of declarations of constant variables of poST programs;
- VB is a fragment of Promela program that is a result of transformation of declarations of non-constant variables of poST programs;
- CPg returns the name of the current program;
- $CP(pg)$ returns the name of the current process of program pg;
- $CS(p, pg)$ returns the name of the current state of process p of program pg;
- $FS(p, pg)$ returns the first state of process p of program pg;
- $NS(s, p, pg)$ returns the next state for state s of process p of program pg;
- $Timed(s, p, pg)$ returns true if s contains a timeout statement;
- $Interval$ returns the duration of a scan loop. We consider that it is constant.

II. Transformation Rules

Let functions VR, PR and SR specify renaming of variables, processes and states of poST programs in Promela program. They input a name of a variable, process and state, respectively and a transformation context.

Let $CSV(p, pg)$ specifies a Promela variable that stores the current state of process p of program pg.

Let $TV(p, pg)$ specifies a Promela variable that stores the local time of process p of program pg. If a current state s of process p is timed, the value of variable $TV(p, pg)$ is the number of iteration of scan loop during which process p has state s including the current iteration.

Let Λ denote an empty fragment.

Let $s_{stop}(p, pg)$ and $s_{error}(p, pg)$ be Promela names for poST states `STOP` and `ERROR` for process p of program pg.

Let $SS(p, pg, c)$ return a state sequence $\{s_1, ..., s_n, s_{stop}(p, pg), s_{error}(p, pg)\}$, where $s_{i+1} = c.NS(s_i, p, pg)$.

Let $ST(p, pg, c)$ be a Promela name for the enumeration type with values from $SS(p, pg, c)$ specifying all states of process p of program pg.

Let $PCS(c)$ return a sequence of codes

$$\begin{gathered}\{PC(pc_{11}), \ldots, PC(pc_{1m_1}),\\ \ldots,\\ PC(pc_{m1}), \ldots, PC(pc_{mn_m}),\\ PC(\texttt{Gremlin}), PC(\texttt{OutInput}), PC(\texttt{BOC})\}\end{gathered}$$

of processes corresponding processes of the control algorithm based on a sequence of poST programs $\{pg_1, ..., pg_m\}$, where PC is encoding function, p_{ij} is a process of program pc_i, $p_{ij+1} = c.NP(p_{ij}, pg)$ for $1 < j < m_i$.

The last three processes are called service processes. They are explicitly not represented in poST programs on which the control algorithm is based. The process `Gremlin` simulates an environment (in particular, control object) of the control algorithm. The process `OutInput` specifies exchange of values of input and output variables among processes of programs between iterations of the scan cycle of the control algorithm. The process BOC (Beginning Of Cycle) assigns the value `true` to the special variable `cycle_u` at the moment of transition from one iteration of the scan loop to the next iterations, allowing us to specifies the number of iterations in LTL requirements.

Let $PT(c)$ be a Promela name for the enumeration type with values from $PCS(c)$ specifying codes of all processes of the control algorithm.

Let $PN(pc, c)$ returns the next element of the sequence $PCS(c)$ for $pc \in PCS(c)$. If pc is the final element, it returns the first element of $PCS(c)$.

Let cpg denote cpg, cp denote $c.CP(cpg)$, and cs denote $c.CS(cp, cpg)$.

Let csv denote $CSV(cp, cpg)$, and tv denote $TV(cp, cpg)$.

Let letters u, v, w (possibly with indexes and primes) denote a fragment, a nonempty sequence of fragments and a sequence of fragments, respectively, of programs both poST and Promela

A. The Control Algorithm

Transformation of the control algorithm based on sequences of poST programs and constant scan cycle interval is defined by the following rules:

$$\frac{(v, c) \mapsto_R (w, c_1),\ (v, c_1[Interval := v]) \mapsto_{Pg} (v', c_2)}{\begin{array}{c}(\text{PROGRAMS } v \text{ INTERVAL } u, c) \mapsto_A \\ (c.CB\ c.VB \\ \text{mtype:}PT(c) = PCS(c) \\ \text{chan current = [1] of mtype:}PT(c) \\ \text{init\{ current ! } PC(\texttt{Gremlin})\text{; \}} \\ \text{active proctype Gremlin() \{...\}} \\ \text{active proctype OutInput() \{...\}} \\ \text{active proctype BOC() \{...\}} \\ v', c_2)\end{array}}$$

Channel `current` stores the code of the current process of the control algorithm.

B. Renaming

The relation $\mapsto_R$ generates Promela names for the corresponding names of variables, processes and process states of the poST program. it also fills the attributes VN, PN and SN of the transformation context. This relation is defined by the following rules:

$$\frac{\begin{array}{c}v_1 \in \{VAR, VAR\ CONSTANT, VAR\_INPUT, \\ VAR\_OUTPUT, VAR\_IN\_OUT\}, \\ n = VR(u, c), \\ c_1 = c[VN(u, cp, cpg) := n] \\ (w, c_1) \mapsto_R (w', c_2)\end{array}}{(v_1\ u\ v_2 \text{ END\_VAR } w, c) \mapsto_R (w', c_2)}$$

$$\frac{\begin{array}{c}n = PR(u, c), \\ c_1 = c[PN(u, cpg) := n] \\ (w_1, c_1[CP(cpg) := u]) \mapsto_R (w_1', c_2), \\ (w_2, c_2[CP(cpg) := \perp]) \mapsto_R (w_2', c_3)\end{array}}{(\text{PROCESS } u\ w_1 \text{ END\_PROCESS } w_2, c) \mapsto_R (w_2', c_3)}$$

$$\frac{n = SR(u, c),\ (w_2, c) \mapsto_R (w_2', c_1)}{(\text{STATE } u\ w_1 \text{ END\_STATE } w_2, c) \mapsto_R (w_2', c_1)}$$

$$\frac{\begin{array}{c}(w_1, c[CPg(cpg) := u]) \mapsto_R (w_1', c_1), \\ (w_2, c_1[CPg(cpg) := \perp]) \mapsto_R (w_2', c_2)\end{array}}{(\text{PROGRAM } u\ w_1 \text{ END\_PROGRAM } w_2, c) \mapsto_R (w_2', c_2)}$$

$$(\Lambda, c) \mapsto_R (\Lambda, c)$$

C. Examples of transformation of service processes

For simplicity, we consider transformation of service processes on examples of poST programs (see Table 1).

TABLE I
SERVICE PROCESSES

poST	**Promela**

PROCESS u
VAR_INPUT
u_1 : BOOL;
u_2 : USINT;
u_3 : SINT;
...
END_VAR
...
END_PROCESS
	active proctype Gremlin() {
	do :: current ? PC(Gremlin) ->
	atomic {
	if
	:: $c.VN(u_1, u, cpg)$ = **true**;
	:: $c.VN(u_1, u, cpg)$ = **false**;
	fi;
	select ($c.VN(u_2, u, cpg)$: 0..255);
	select ($c.VN(u_3, u, cpg)$: -128..127);
	current ! $PN(PC(\texttt{Gremlin}), c)$;}
	od;
	}

PROGRAM u_1
VAR_OUTPUT
u : t;
END_VAR
...
END_PROGRAM
...
PROGRAM u_2
VAR_INPUT
u : t;
END_VAR
...
END_PROGRAM
	active proctype OutInput() {
	do :: current ? PC(OutInput) ->
	atomic {
	$c.VN(u, \perp, u_2)$ = $c.VN(u, \perp, u_1)$;
	...
	current ! $PN(PC(\texttt{OutInp}), c)$;}
	od;
	}

	bool cycle_u;
	active proctype BOC () {
	do :: current ? PC(BOC) ->
	cycle_u = **true**;
	atomic {
	cycle_u = **false**;
	current ! $PN(PC(\texttt{BOC}), c)$; }
	od;
	}

D. Programs

Transformation of poST programs is defined by the following rules:

$$\frac{(v, c) \mapsto_{St} (v', c_1)}{\text{PROGRAM } u\ v \text{ END\_PROGRAM}, c) \mapsto_{Pg} (v', c_1)}$$

$$\frac{\begin{array}{c}u \text{ is a program declaration}, \\ (u, c) \mapsto_{Pg} (u', c_1),\ (w, c_1) \mapsto_{Pg} (w', c_2)\end{array}}{(u\ w, c) \mapsto_{Pg} (u'\ w', c_2)}$$

$$(\Lambda, c) \mapsto_{Pg} (\Lambda, c_2)$$

E. Processes

Transformation of poST processes is defined by the following rules:

$$\frac{(v, c) \mapsto_{St} (v', c_1),\ u' = c.PN(u, cpg)}{\begin{array}{c}(\text{PROCESS}\ u\ v\ \text{END\_PROCESS}, c) \mapsto_P \\ (\text{mtype:}ST(p, pg, c) = SS(p, pg, c) \\ \text{mtype:}ST(p, pg, c)\ csv = s_{stop}(p, pg); \\ \text{active proctype}\ u'()\ \{ \\ \text{do}\ \texttt{::}\ \text{current}\ \texttt{?}\ \mu(u')\ \texttt{->} \\ \text{atomic}\ \{\ \text{if}\ v'\ \texttt{::}\ \text{else}\ \texttt{-->}\ \text{skip; fi;} \\ \text{current}\ \texttt{!}\ NP(u, c);\} \\ \text{od;}\}, c_1)\end{array}}$$

$$\frac{\begin{array}{c}u \text{ is a process declaration,} \\ (u, c) \mapsto_P (u', c_1),\ (w, c_1) \mapsto_P (w', c_2)\end{array}}{(u\ w, c) \mapsto_P (u'\ w', c_2)}$$

$$(\Lambda, c) \mapsto_P (\Lambda, c_2)$$

F. States

Transformation of poST process states is defined by the following rules:

$$\frac{(v, c) \mapsto_{St} (v', c_1)}{\begin{array}{c}(\text{STATE}\ u\ v\ \text{END\_STATE}, c) \mapsto_S \\ (\texttt{::}\ csv\ \texttt{==}\ cs\ \texttt{->}\ \{v'\}, c_1)\end{array}}$$

$$\frac{\begin{array}{c}u \text{ is a state declaration,} \\ (u, c) \mapsto_S (u', c_1),\ (w, c_1) \mapsto_S (w', c_2)\end{array}}{(u\ w, c) \mapsto_S (u'\ w', c_2)}$$

$$(\Lambda, c) \mapsto_S (\Lambda, c_2)$$

G. Types

Transformation of poST types is defined by the following rules:

$$\frac{\begin{array}{c}u \in \{\text{DINT}, \text{LINT}, \text{UDINT}, \\ \text{ULINT}, \text{DWORD}, \text{LWORD}\}\end{array}}{(u, c) \mapsto_T (\text{int}, c)}$$

$$\frac{u \in \{\text{SINT}, \text{INT}, \text{WORD}\}}{(u, c) \mapsto_T (\text{short}, c)}$$

$$\frac{u \in \{\text{USINT}, \text{BYTE}\}}{(u, c) \mapsto_T (\text{byte}, c)}$$

$$\frac{u \in \{\text{UINT}, \text{TIME}\}}{(u, c) \mapsto_T (\text{unsigned}, c)}$$

$$\frac{\begin{array}{c}u_3 \neq TIME, (u_1, c) \mapsto_E (w_1, c_1), \\ (u_2, c_1) \mapsto_E (w_2, c_2), (u_3, c_2) \mapsto_T (w_3, c_3)\end{array}}{(\text{ARRAY}[u_1\text{:}u_2]\ \text{OF}\ u_3, c) \mapsto_T (w_3[\ ], c_3)}$$

$$(\text{BOOL}, c) \mapsto_T (\text{bool}, c)$$

The real types `REAL` and `LREAL` as well as the string types `STRING` and `WSTRING` are not supported.

H. Expressions

Transformation rules for expressions are divided into groups of rules for boolean operators, relation operators, arithmetic operators, and state-handling operators

1) Boolean Operators: Transformation of boolean expressions is defined by the following rules:

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \text{XOR}\ u_2, c) \mapsto_E (u_1'\ \texttt{\^{}}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \text{OR}\ u_2, c) \mapsto_E (u_1'\ \texttt{||}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \text{AND}\ u_2, c) \mapsto_E (u_1'\ \texttt{\&}\ u_2', c)}$$

$$\frac{(u, c) \mapsto_E (u', c')}{(\text{NOT}\ u, c) \mapsto_E (\ \texttt{!}\ u', c')}$$

2) Comparison Operators: Transformation of comparison expressions is defined by the following rules:

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{=}\ u_2, c) \mapsto_E (u_1'\ \texttt{==}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{<>}\ u_2, c) \mapsto_E (u_1'\ \texttt{!=}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{<}\ u_2, c) \mapsto_E (u_1'\ \texttt{<}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{>}\ u_2, c) \mapsto_E (u_1'\ \texttt{>}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{<=}\ u_2, c) \mapsto_E (u_1'\ \texttt{<=}\ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{>=}\ u_2, c) \mapsto_E (u_1'\ \texttt{>=}\ u_2', c)}$$

3) Arithmetic Operators: Transformation of arithmetic expressions is defined by the following rules:

$$\frac{\begin{array}{c}\circ \in \{+, -, *, /\}, \\ (u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)\end{array}}{(u_1 \circ u_2, c) \mapsto_E (u_1' \circ u_2', c)}$$

$$\frac{(u_1, c) \mapsto_E (u_1', c_1), (u_2, c_1) \mapsto_E (u_2', c_2)}{(u_1\ \texttt{MOD}\ u_2, c) \mapsto_E (u_1'\ \texttt{\%}\ u_2', c)}$$

$$\frac{(u, c) \mapsto_E (u', c')}{(-\ u, c) \mapsto_E (\ -\ u', c')}$$

The operator (`**`) of exponentiation is not supported.

4) State-handling Operators: Transformation of state-handling expressions is defined by the following rules:

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{PROCESS } u \text{ IN STATE ACTIVE}, c) \mapsto_E \\ (n \texttt{ != } s_{stop}(u, cpg) \texttt{ \& } n \texttt{ != } s_{error}(u, cpg), c)\end{array}}$$

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{PROCESS } u \text{ IN STATE INACTIVE}, c) \mapsto_E \\ (n \texttt{ = } s_{stop}(u, cpg) \texttt{ || } n \texttt{ = } s_{error}(u, cpg), c)\end{array}}$$

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{PROCESS } u \text{ IN STATE STOP}, c) \mapsto_E \\ (n \texttt{ = } s_{stop}(u, cpg), c)\end{array}}$$

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{PROCESS } u \text{ IN STATE ERROR}, c) \mapsto_E \\ (n \texttt{ = } s_{error}(u, cpg), c)\end{array}}$$

I. Variable Declarations

Transformation of variable declarations is defined by the following rules:

$$\frac{\begin{array}{c} v \in \{VAR, VAR\_INPUT, \\ VAR\_OUTPUT, VAR\_IN\_OUT\}, \\ n = c.VN(u_1, cp, cpg), \\ (u_2, c) \mapsto_t (u_2', c_1)\end{array}}{\begin{array}{c}(v\ u_1 : u_2 \text{ END\_VAR}, c) \mapsto_{St} \\ (\Lambda, c_1[VB := c_1.VB\ u_2'\ n;]\end{array}}$$

$$\frac{\begin{array}{c} v \in \{VAR, VAR\_INPUT, \\ VAR\_OUTPUT, VAR\_IN\_OUT\}, \\ n = c.VN(u_1, cp, cpg), \\ (u_2, c) \mapsto_t (u_2', c_1), (u_3, c_1) \mapsto_e (u_3', c_2)\end{array}}{\begin{array}{c}(v\ u_1 : u_2 \texttt{ = } u_3 \text{ END\_VAR}, c) \mapsto_{St} \\ (\Lambda, c_2[VB := c_2.VB\ u_2'\ n \texttt{ = } u_3';]\end{array}}$$

$$\frac{n = c.VN(u_1, cp, cpg),\ (u_3, c) \mapsto_e (u_3', c_1)}{\begin{array}{c}(\text{VAR CONST } u_1 : u_2 \texttt{ = } u_3 \text{ END\_VAR}, c) \mapsto_{St} \\ (\Lambda, c_1[CB := c_1.CB \text{ \#define } n\ u_3']\end{array}}$$

J. Statement Sequences

Transformation of statement sequences is defined by the following rules:

$$\frac{\begin{array}{c} u \text{ is a statement}, \\ (u, c) \mapsto_{St} (u', c_1),\ (w, c_1) \mapsto_{St} (w', c_2)\end{array}}{(u\ w, c) \mapsto_{St} (u'\ w', c_2)}$$

$$(\Lambda, c) \mapsto_{St} (\Lambda, c_2)$$

K. ST Statements

Transformation rules for ST statements are divided into groups of rules for assignment statements, selection statements (if statements, case statements), and iteration statements (while statements, repeat statements, for statements).

1) Assignment: Transformation of assignment statements is defined by the following rules:

$$\frac{n = c.VN(u_1, cp, cpg),\ (u_2, c) \mapsto_e (u_2', c_1)}{(u_1 \texttt{ := } u_2, c) \mapsto_{St} (n \texttt{ = } u_2', c_1)}$$

2) If Statements: Transformation of if statements is defined by the following rules:

$$\frac{(u, c) \mapsto_e (u', c_1), (v, c_1) \mapsto_{st,s} (v', c_2)}{\begin{array}{c}(\text{IF } u \text{ THEN } v \text{ END\_IF}, c) \mapsto_{St} \\ (\text{if :: } u' \texttt{ -> } \{v'\} \text{ :: else } \texttt{->} \text{ skip; fi;}, c_2)\end{array}}$$

$$\frac{\begin{array}{c}(u, c) \mapsto_e (u', c_1), (v_1, c_1) \mapsto_{st,s} (v_1', c_2), \\ (v_2, c_2) \mapsto_{st,s} (v_2', c_3)\end{array}}{\begin{array}{c}(\text{IF } u \text{ THEN } v_1 \text{ ELSE } v_2 \text{ END\_IF}, c) \mapsto_{St} \\ (\text{if :: } u' \texttt{ -> } \{v_1'\} \text{ :: else } \texttt{->} \{v_2'\} \text{ fi;}, c_3)\end{array}}$$

$$\frac{\begin{array}{c}(u, c) \mapsto_e (u', c_1), (v_1, c_1) \mapsto_{st,s} (v_1', c_2), \\ (\text{IF } v_2, c_2) \mapsto_{st,s} (v_2', c_3)\end{array}}{\begin{array}{c}(\text{IF } u \text{ THEN } v_1 \text{ ELSEIF } v_2, c) \mapsto_{St} \\ (\text{if :: } u' \texttt{ -> } \{v_1'\} \text{ :: else } \texttt{->} v_2' \text{ fi;}, c_3)\end{array}}$$

3) Case Statements: Transformation of case statements is defined by the following rules:

$$\frac{\begin{array}{c}(u, c) \mapsto_e (u', c_1), (v_1, c_1) \mapsto_{st,s} (v_1', c_2), \\ n \text{ is a fresh name}, (v_2, c_2[caseVal := n]) \mapsto_{cases} (v_2', c_3)\end{array}}{\begin{array}{c}(\text{CASE } u \text{ OF } v_1 \text{ ELSE } v_2 \text{ END\_CASE}, c) \mapsto_{St} \\ (\text{int } n \texttt{ = } v_1'; \text{ if } v_1' \text{ :: else } \texttt{->} \{v_2'\} \text{ fi;}, c_3)\end{array}}$$

The intermediate transformation relations $\mapsto_{cases}$ and $\mapsto_{labels}$ specify transformation of case branches and case labels, respectively.

$$\frac{\begin{array}{c}v_1 : v_2 \text{ is a case branch}, n = c.caseVal, \\ (v_1, c) \mapsto_{labels} (v_1', c_1), (v_2, c_1) \mapsto_{st,s} (v_2', c_2), \\ (v_3, c_2[caseVal := n]) \mapsto_{cases} (v_3', c_3)\end{array}}{(v_1 : v_2\ v_3, c) \mapsto_{cases} (\text{:: } v_1' \texttt{ -> } \{v_2'\}\ v_3', c_3)}$$

$$\frac{\begin{array}{c}v_1 : v_2 \text{ is a case branch}, (v_1, c) \mapsto_{labels} (v_1', c_1), \\ (v_2, c_1) \mapsto_{st,s} (v_2', c_2) \mapsto_{cases} (v_3', c_3)\end{array}}{(v_1 : v_2, c) \mapsto_{cases} (\text{:: } v_1' \texttt{ -> } \{v_2'\}, c_3)}$$

$$\frac{u \text{ is a label}, n = c.caseVal, (v, c) \mapsto_{labels} (v', c')}{(u\ v, c) \mapsto_{labels} (n \texttt{ == } u \texttt{ || } v', c')}$$

$$\frac{u \text{ is a label}, n = c.caseVal}{(u, c) \mapsto_{labels} (n \texttt{ == } u, c')}$$

4) While Statements: Transformation of while statements is defined by the following rules:

$$\frac{(u, c) \mapsto_e (u', c_1), (v, c_1) \mapsto_{st,s} (v', c_2)}{\begin{array}{c}(\text{WHILE } u \text{ DO } v \text{ END\_WHILE}, c) \mapsto_{St} \\ (\text{do :: } u' \texttt{ -> } \{v'\} \text{ :: else } \texttt{->} \text{ break; od;}, c_2)\end{array}}$$

5) Repeat Statements: Transformation of repeat statements is defined by the following rules:

$$\frac{(v,c) \mapsto_{st,s} (v',c_1), (u,c_1) \mapsto_e (u',c_2)}{\begin{array}{c}(\text{REPEAT } v \text{ UNTIL } u \text{ END\_REPEAT}, c) \mapsto_{St} \\ (v'\text{do} :: u' \text{ -> } \{v'\} :: \text{else -> break; od;}, c_2)\end{array}}$$

6) For Statements: Transformation of for statements is defined by the following rules:

$$\frac{\begin{array}{c}n = c.VN(u, cpg, c.CP), (u_1,c) \mapsto_{st,s} (u_1',c_1), \\ (u_2,c_1) \mapsto_e (u_2',c_2), (v,c_2) \mapsto_e (v',c_3)\end{array}}{\begin{array}{c}(\text{FOR } u \text{ := } u_1 \text{ TO } u_2 \text{ DO } v \text{ END\_FOR}, c) \mapsto_{St} \\ (n = u_1'; \text{ do :: } n \text{ <= } u_2' \text{ -> } \{v'\ n = n + 1;\} \\ \text{:: else -> break; od;}, c_3)\end{array}}$$

$$\frac{\begin{array}{c}n = c.VN(u, cpg, c.CP), (u_1,c) \mapsto_{st,s} (u_1',c_1), \\ (u_2,c_1) \mapsto_e (u_2',c_2), (u_3,c_2) \mapsto_e (u_3',c_3), \\ (v,c_3) \mapsto_e (v',c_4)\end{array}}{\begin{array}{c}(\text{FOR } u \text{ := } u_1 \text{ TO } u_2 \text{ BY } u_3 \text{ DO } v \text{ END\_FOR}, c) \mapsto_{St} \\ (n = u_1'; \text{ do :: } n \text{ <= } u_2' \text{ -> } \{v'\ n = n + u_3';\} \\ \text{:: else -> break; od;}, c_4)\end{array}}$$

L. Process-handling Statements

Transformation rules for process-handling statements are divided into groups of rules for start statements, stop statements, error statements, set statements, and timeout statements.

1) Start statements: Transformation of start statements is defined by the following rules:

$$\frac{\begin{array}{c}n = CSV(u, cpg), \\ s = FS(c.CS(u, cpg), u, cpg), \\ s' = c.SN(s, u, cpg), Timed(s, u, cpg) = false\end{array}}{\begin{array}{c}(\text{START PROCESS } u, c) \mapsto_{St} \\ (n = s';, c[CS(u, cpg) := s])\end{array}}$$

$$\frac{\begin{array}{c}n = CSV(u, cpg), \\ s = FS(c.CS(u, cpg), u, cpg), \\ s' = c.SN(s, u, cpg), Timed(s, u, cpg) = true, \\ t = CSV(u, cpg),\end{array}}{\begin{array}{c}(\text{START PROCESS } u, c) \mapsto_{St} \\ (n = s';\ t = 1;, c[(u, cpg) := s])\end{array}}$$

$$\frac{\begin{array}{c}s = FS(c.CS(cp, cpg), cp, cpg), \\ s' = c.SN(s, cp, cpg), Timed(s, cp, cpg) = false\end{array}}{(\text{RESTART}, c) \mapsto_{St} (csv = s';, c[CS(cp, cpg) := s])}$$

$$\frac{\begin{array}{c}s = FS(c.CS(cp, cpg),\ s' = c.SN(s, cp, cpg), \\ Timed(s, cp, cpg) = true\end{array}}{(\text{RESTART}, c) \mapsto_{St} (csv = s';\ tv = 1;, c[CS(cp, cpg) := s])}$$

2) Stop statements: Transformation of stop statements is defined by the following rules:

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{STOP PROCESS } u, c) \mapsto_{St} \\ (n = s;, c[CS(u, cpg) := STOP])\end{array}}$$

$$\begin{array}{c}(\text{STOP}, c) \mapsto_{St} \\ (csv = s;, c[CS(cp, cpg) := STOP])\end{array}$$

3) Error statements: Transformation of error statements is defined by the following rules:

$$\frac{n = CSV(u, cpg)}{\begin{array}{c}(\text{ERROR PROCESS } u, c) \mapsto_{St} \\ (n = s;, c[CS(u, cpg) := ERROR])\end{array}}$$

$$\begin{array}{c}(\text{ERROR}, c) \mapsto_{St} \\ (csv = s;, c[CS(cp, cpg) := ERROR])\end{array}$$

4) Set statements: Transformation of set statements is defined by the following rules:

$$\frac{u' = c.SN(u, cp, cpg),\ Timed(u, cp, cpg) = false}{(\text{SET STATE } u, c) \mapsto_{St} (csv = u';, c[CS(cp, cpg) := u])}$$

$$\frac{\begin{array}{c}u' = c.SN(u, cp, cpg),\ Timed(u, cp, cpg) = true, \\ t = CSV(u, cp, cpg),\end{array}}{\begin{array}{c}(\text{SET STATE } u, c) \mapsto_{St} \\ (csv = u';\ t = 1;, c[CS(cp, cpg) := u])\end{array}}$$

$$\frac{\begin{array}{c}u = NS(c.CS, cp, cpg), \\ u' = c.SN(u, cp, cpg), Timed(u, cp, cpg) = false\end{array}}{(\text{SET NEXT}, c) \mapsto_{St} (csv = u';, c[CS(cp, cpg) := u])}$$

$$\frac{\begin{array}{c}u = NS(c.CS, cp, cpg),\ u' = c.SN(u, cp, cpg), \\ Timed(u, cp, cpg) = true, t = CSV(u, cp, cpg),\end{array}}{\begin{array}{c}(\text{SET NEXT}, c) \mapsto_{St} \\ (csv = u';\ t = 1;, c[CS(cp, cpg) := u])\end{array}}$$

5) Timeout Statements: Transformation of timeout statements is defined by the following rules:

$$\frac{Timed(c.CS(cp, cpg), cp, cpg) = false}{(\text{RESET TIMER}, c) \mapsto_{St} (\Lambda, c)}$$

$$\frac{Timed(c.CS(cp, cpg), cp, cpg) = true}{(\text{RESET TIMER}, c) \mapsto_{St} (tv = 1;\ , c)}$$

$$\frac{\begin{array}{c}u' = \lceil u /\ c.Interval \rceil, \\ (v,c) \mapsto_{st,s} (v',c_1)\end{array}}{\begin{array}{c}(\text{TIMEOUT } u \text{ THEN } v \text{ END\_TIMEOUT}, c) \mapsto_{St} \\ (\text{if :: } tv \text{ > } u' \text{ -> } \{tv = 1;\ v'\} \\ \text{:: else -> } tv = tv + 1; \text{ fi;}, c_1)\end{array}}$$