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 application (a sequence of poST programs);
- $\mapsto_{PgS}$ transforms sequences of poST programs;
- $\mapsto_{Pg}$ transforms poST programs;
- $\mapsto_T$ transforms types;
- $\mapsto_V$ transforms variable declarations;
- $\mapsto_E$ transforms expressions;
- $\mapsto_{St}$ transforms statements;
- $\mapsto_{StS}$ transforms sequences of statements;
- $\mapsto_P$ transforms process declarations;
- $\mapsto_{PS}$ transforms sequences of process declarations;
- $\mapsto_S$ transforms state declarations;
- $\mapsto_{SS}$ transforms sequences of state declarations;
- $\mapsto_R$ 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 $p = \perp$, 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;
- $NP(p, pg)$ returns the next process for process p of program pg;
- $NPg(pg)$ returns the next program for program p;
- $Timed(s, p, pg)$ 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. More exactly, they use only components vn, pn and sn of the context, respectively.

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

Let $timer(p, pg)$ 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 $timer(p)$ is the number of iteration of scan loop during 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 $SST(p, pg, c)$ be a Promela name for the enumeration type with values from $SS(p, pg, c)$.

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. Programs

$$\frac{(u, c) \mapsto_{Pg} (w, c'), (v, c') \mapsto_{Pg,s} (w', c'')}{(u\ v, c) \mapsto_{Pg,s} (w\ w', c'')}$$

$$\frac{(u, c) \mapsto_{pg} (w, c')}{(u, c) \mapsto_{PgS} (w, c'')}$$

B. Control Application

$$\frac{(app, c) \mapsto_R (w_1, c_1), (v, c_1) \mapsto_{PgS} (f, c_2)}{(v, c) \mapsto_A (f, c_2)}$$

C. Types

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

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

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

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

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

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

The string types `STRING` and `WSTRING` are not supported.

D. Expressions

1) Boolean Operators:

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

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

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

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

2) Relation Operators:

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

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

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

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

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

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

3) Arithmetic Operators:

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

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

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

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

4) State-handling Operators:

$$\frac{n = cstate(u, c.CPg)}{\begin{gathered}(\text{PROCESS } u \text{ IN STATE ACTIVE}, c) \mapsto_E\\ (n \texttt{ != } s_{stop}(u, c.CPg) \texttt{ \& } n \texttt{ != } s_{error}(u, c.CPg), c)\end{gathered}}$$

$$\frac{n = cstate(u, c.CPg)}{\begin{gathered}(\text{PROCESS } u \text{ IN STATE INACTIVE}, c) \mapsto_E\\ (n \texttt{ = } s_{stop}(u, c.CPg) \texttt{ || } n \texttt{ = } s_{error}(u, c.CPg), c)\end{gathered}}$$

$$\frac{n = cstate(u, c.CPg)}{\begin{gathered}(\text{PROCESS } u \text{ IN STATE STOP}, c) \mapsto_E\\ (n \texttt{ = } s_{stop}(u, c.CPg), c)\end{gathered}}$$

$$\frac{n = cstate(u, c.CPg)}{\begin{gathered}(\text{PROCESS } u \text{ IN STATE ERROR}, c) \mapsto_E\\ (n \texttt{ = } s_{error}(u, c.CPg), c)\end{gathered}}$$

E. 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.

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

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

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

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

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

F. Variable Declarations

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

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

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

G. ST Statements

1) Assignment:

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

2) If Statements:

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

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

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

3) Case Statements:

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

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

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

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

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

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

4) While Statements:

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

5) Repeat Statements:

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

6) For Statements:

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

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

H. Process-handling Statements

1) Start statements:

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

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

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

$$\frac{\begin{array}{c} cp = c.CP(c.CPg), n = cstate(cp, c.CPg), \\ s = FS(c.CS(cp, c.CPg), cp = c.CP(c.CPg), , c.CPg), \\ s' = c.SN(s, cp, c.CPg), Timed(s, cp, c.CPg) = true, \\ t = timer(cp, c.CPg), \end{array}}{(\text{RESTART}, c) \mapsto_{St} (n = s';\ t = 1;, c[CS(cp, c.CPg) := s])}$$

2) Stop statements:

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

$$\frac{n = cstate(c.CP(c.CPg), c.CPg)}{\begin{array}{c} (\text{STOP}, c) \mapsto_{St} \\ (n = s;, c[CS(c.CP(c.CPg), c.CPg) := STOP]) \end{array}}$$

3) Error statements:

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

$$\frac{n = cstate(c.CP(c.CPg), c.CPg)}{\begin{array}{c} (\text{ERROR}, c) \mapsto_{St} \\ (n = s;, c[CS(c.CP(c.CPg), c.CPg) := ERROR]) \end{array}}$$

4) Set statements:

$$\frac{\begin{array}{c} cp = c.CP(c.CPg),\ n = cstate(cp, c.CPg), \\ u' = c.SN(u, cp, c.CPg),\ Timed(u, cp, c.CPg) = false \end{array}}{(\text{SET STATE } u, c) \mapsto_{St} (n = u';, c[CS(cp, c.CPg) := u])}$$

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

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

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

5) Time-handling Statements:

$$\frac{\begin{array}{c} cp = c.CP(c.CPg), \\ Timed(c.CS(cp, c.CPg), cp, c.CPg) = false \end{array}}{(\text{RESET TIMER}, c) \mapsto_{St} (\Lambda, c)}$$

$$\frac{\begin{array}{c} cp = c.CP(c.CPg), \\ Timed(c.CS(cp, c.CPg), cp, c.CPg) = true, \\ t = timer(cp, c.CPg) \end{array}}{(\text{RESET TIMER}, c) \mapsto_{St} (t = 1;\ , c)}$$

$$\frac{\begin{array}{c} t = timer(c.CP(c.CPg), c.CPg), \\ u' = \lceil u /\ c.Interval \rceil, \\ (v, c) \mapsto_{st,s} (v', c_1) \end{array}}{\begin{array}{c} (\text{TIMEOUT } u \text{ THEN } v \text{ END\_TIMEOUT}, c) \mapsto_{St} \\ (\text{if :: } t > u' \text{ -> } \{t = 1;\ v'\} \text{ :: else -> } t = t + 1;\ \text{fi};, c_1) \end{array}}$$