A Supplemental Material for A Token-Based Distributed Group Mutual Exclusion Algorithm with Quorums

Hirotsugu Kakugawa, *Member, IEEE*, and Sayaka Kamei, *Member, IEEE*, and Toshimitsu Masuzawa, *Member, IEEE*

***Abstract*— In this material, we show complete proof of correctness of TQGmx, and model checking of TQGmx.**

I. Formal Proof of Correctness

A. Invariants

The proposed algorithm is designed to maintain the following *invariants*. Symbols used in invariants are shown in Figure 1.

- InvA: The main-token is maintained so that it is unique in the system.

$$V_{tok} + C_{\mathsf{token}} = 1 \tag{1}$$

- InvB: *tok*.*gSize* counts the number of tokens in use.

$$T_{\textsc{Main}} + C_{\mathsf{subtoken}} + T_{\textsc{Sub}} + C_{\mathsf{release}} = \mathit{gSize} \tag{2}$$

- InvC: Maintenance of requests and timestamp values.

$$\begin{aligned}
&\forall P_i : \mathit{mode}_i = \textsc{Idle} \Rightarrow \\
&\qquad (\mathit{ts}_i = \mathit{tsReq}[i]) \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P \neq P_i) \\
&\qquad\quad \wedge (C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0) &(3)\\
&\forall P_i : \mathit{mode}_i = \textsc{Trying} \Rightarrow \\
&\qquad (\mathit{ts}_i = \mathit{tsReq}[i] + 1) \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P \neq P_i) \\
&\qquad\quad \wedge (C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0) \\
&\qquad \vee (\mathit{ts}_i = \mathit{tsReq}[i]) \wedge (\exists \langle P, t, g\rangle \in \mathit{reqQ} : P = P_i) \\
&\qquad\quad \wedge (C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0) \\
&\qquad \vee (\mathit{ts}_i = \mathit{tsReq}[i]) \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P \neq P_i) \\
&\qquad\quad \wedge (C_{\mathsf{token},*,i} = 1 \wedge C_{\mathsf{subtoken},*,i} = 0) \\
&\qquad \vee (\mathit{ts}_i = \mathit{tsReq}[i]) \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P \neq P_i) \\
&\qquad\quad \wedge (C_{\mathsf{token},*,i} = 0 \wedge C_{\mathsf{subtoken},*,i} = 1) &(4)\\
&\forall P_i : \mathit{mode}_i = \textsc{InCS} \Rightarrow \\
&\qquad (\mathit{ts}_i = \mathit{tsReq}[i]) \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P \neq P_i) \\
&\qquad \wedge (C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0) &(5)\\
&\forall P_i : \\
&\qquad (\forall \langle P, t, g\rangle \in \mathcal{M}_{\mathsf{request}} : P = P_i \Rightarrow t \leq \mathit{ts}_i) &(6)\\
&\quad \wedge (\forall \langle P, t, g\rangle \in \mathit{tmpQ} : P = P_i \Rightarrow t \leq \mathit{ts}_i) &(7)\\
&\quad \wedge (|\{\langle P, t, g\rangle \in \mathit{reqQ} : P = P_i\}| \leq 1) &(8)
\end{aligned}$$

- InvD: Current group is maintained by *gName*.

$$\begin{aligned}
&\mathit{gSize} = 0 \Leftrightarrow \mathit{gName} = \perp &(9)\\
&\wedge \forall P_i : \\
&\qquad (\mathit{mode}_i \neq \textsc{Idle} \Leftrightarrow \mathit{grp}_i \neq \perp) &(10)\\
&\quad \wedge (\forall \langle P, t, g\rangle \in \mathit{reqQ} : P = P_i \Rightarrow g = \mathit{grp}_i) &(11)\\
&\quad \wedge (C_{\mathsf{subtoken},*,i} = 1 \Rightarrow \mathit{gName} = \mathit{grp}_i) &(12)\\
&\quad \wedge (\mathit{mode}_i = \textsc{InCS} \Rightarrow \mathit{gName} = \mathit{grp}_i) &(13)
\end{aligned}$$

- InvE: Process mode and local variables.

$$\begin{aligned}
&\forall P_i : \\
&\qquad (\mathit{mode}_i = \textsc{InCS} \Leftrightarrow \\
&\qquad\quad (\mathit{type}_i = \textsc{Main} \vee \mathit{type}_i = \textsc{Sub})) &(14)\\
&\qquad \wedge (\mathit{type}_i = \textsc{Main} \\
&\qquad\quad \Rightarrow (\mathit{tok}_i \neq \perp \wedge \mathit{home}_i = \perp)) &(15)\\
&\qquad \wedge (\mathit{type}_i = \textsc{Sub} \Rightarrow \mathit{tok}_i = \perp) &(16)\\
&\qquad \wedge (\mathit{type}_i = \textsc{Sub} \Leftrightarrow \mathit{home}_i \neq \perp) &(17)\\
&\qquad \wedge (\mathit{home}_i \neq P_i) &(18)
\end{aligned}$$

- InvF: Transfer of the main-token.

$$\begin{aligned}
&(C_{\mathsf{token}} = 1) \Rightarrow \\
&\qquad (\mathit{gSize} = 0) &(19)\\
&\qquad \wedge (\forall P_i : \mathit{acqs}_i = \emptyset) &(20)\\
&\qquad \wedge (\forall P_i : \mathit{acks}_i = \emptyset) &(21)
\end{aligned}$$

- InvG: Management of the main-token and sub-tokens.

$$\begin{aligned}
&\forall P_i : \\
&\qquad (\forall P_k : 0 \leq S_{\mathsf{subtoken},i,k} - R_{\mathsf{release},k,i}) &(22)\\
&\qquad \wedge (\forall P_k : C_{\mathsf{subtoken},i,k} + B_{\mathit{type},k,\textsc{Sub}} + C_{\mathsf{release},k,i} \\
&\qquad\qquad = S_{\mathsf{subtoken},i,k} - R_{\mathsf{release},k,i}) &(23)\\
&\qquad \wedge (\forall P_k : S_{\mathsf{subtoken},i,k} - R_{\mathsf{release},k,i} > 0 \\
&\qquad\qquad \Rightarrow \mathit{tok}_i \neq \perp) &(24)\\
&\wedge \forall P_i : \\
&\qquad (\forall P_k : 0 \leq R_{\mathsf{subtoken},k,i} - S_{\mathsf{release},i,k} \leq 1) &(25)\\
&\qquad \wedge \forall P_k : (R_{\mathsf{subtoken},k,i} - S_{\mathsf{release},i,k} = 1 \\
&\qquad\qquad \Leftrightarrow \mathit{home}_i = P_k) &(26)
\end{aligned}$$

- V_{tok} is the number of process P_i such that $tok_i \neq \bot$.
- *gSize* is the value of *tok.gSize*.
- *gName* is the value of *tok.gName*.
- $tsReq[i]$ is the value of $tok.tsReq[i]$.
- *reqQ* is the set of all requests in *tok.reqQ*.
- *tmpQ* is the set of all requests in $tmpQ_i$ for some P_i.
- T_{MAIN} (resp. T_{SUB}) is the number of process P_i such that $type_i = \text{MAIN}$ (resp. $= \text{SUB}$).
- C_t is the total number of messages of type t in transit.
- $C_{t,i,k}$ (resp. $C_{t,*,k}$) is the number of messages of type t in transit from P_i (resp. any process) to P_k.
- $\mathcal{M}_t$ is the set of all messages of type t in transit.
- $S_{t,i,k}$ is the total number of **send**s of message of type t from P_i to P_k since the system startups.
- $R_{t,i,k}$ is the total number of receipts of message of type t from P_i to P_k since the system startups.
- $B_{acqs,i,j}$ is 1 if $acqs_i \ni P_j$ and 0 otherwise.
- $B_{acks,i,j}$ is 1 if $acks_i \ni P_j$ and 0 otherwise.
- $B_{type,k,\text{SUB}}$ is 1 if $type_k = \text{SUB}$ and 0 otherwise.

Fig. 1. Symbols used in invariants.

- InvH: Sending acquired, leave, ack messages.

$$\forall P_i \forall P_k :$$
$$(0 \leq S_{\text{acquired},i,k} - S_{\text{leave},i,k} \leq S_{\text{acquired},i,k} - R_{\text{ack},k,i} \leq 1) \tag{27}$$
$$\wedge (S_{\text{acquired},i,k} - S_{\text{leave},i,k} = B_{acqs,i,k}) \tag{28}$$
$$\wedge (S_{\text{leave},i,k} - R_{\text{ack},k,i} = B_{acks,i,k}) \tag{29}$$
$$\wedge (tok_i = \bot \Rightarrow acqs_i = acks_i = \emptyset) \tag{30}$$

- InvI: Receiving acquired, leave, ack messages and maintenance of $holder_i$.

$$\forall P_i \forall P_k :$$
$$(R_{\text{leave},k,i} = S_{\text{ack},i,k}) \tag{31}$$
$$\wedge (R_{\text{acquired},k,i} - R_{\text{leave},k,i} = 1 \Leftrightarrow holder_i = P_k) \tag{32}$$

- InvJ: Local variables.

$$\forall P_i :$$
$$(tok_i = \bot \Rightarrow \neg leaving_i) \tag{33}$$
$$\wedge (tok_i = \bot \Rightarrow acks_i = \emptyset) \tag{34}$$
$$\wedge (acks_i \neq \emptyset \Rightarrow acqs_i = \emptyset) \tag{35}$$
$$\wedge (leaving_i \Leftrightarrow acks_i \neq \emptyset) \tag{36}$$
$$\wedge (leaving_i \Rightarrow gSize = 0) \tag{37}$$

B. Preliminary lemmas

First, only for the purpose of simplicity of invariant description, we show the following lemma.

Lemma 1: Invariant InvA is maintained for any execution.

Proof: When the system is initialized, $tok_0 \neq \bot$, $tok_i = \bot$ for each $P_i (\neq P_0)$ and $C_{\text{token}} = 0$ hold. Thus, invariant InvA is maintained initially.

A token message is sent at lines 4.13 and 12.14 by process P_i such that $tok_i \neq \bot$ by conditions of line 4.1 and 12.4. Since the value of tok_i becomes $\bot$ just after a token message is sent, the invariant is maintained. When a token message is received by a process, say P_k, the main-token is held by P_k (line 5.1), and hence the invariant is maintained.

Because there is no other assignment statement for local variable tok_i, the invariant is maintained. ■

For simplicity of description of invariants, the (unique) main-token is denoted by *tok*, which may be held by a process or may be in a token message in transit. We denote, by simply *gSize* in invariant description, $tok_i.gSize$ at some process P_i such that $tok_i \neq \bot$ without ambiguity since the main-token is exactly one. Similarly, we use notation *gName* and *reqQ*.

Below, by induction, we show that invariants are maintained. First, we show a base step.

Lemma 2: (Base step) When the system is initialized, all the invariants are satisfied, and precondition of *requestEvent* is satisfied at each process.

Proof: It is easy to check from initialization procedure. ■

Next, we start induction step.

Lemma 3: (Induction step, precondition of message receipt) Suppose that a message arrives at P_i provided that (1) precondition of a handler that sends the message is satisfied, and (2) all the invariants are satisfied just before the message arrives at P_i. Then, precondition of corresponding handler is satisfied.

Proof: For each message type, we check corresponding precondition just before a message arrives at P_i.

- A $\langle \text{token}, tok, t \rangle$ message from P_k:
  - $tok_i = \bot$: Since a token message is in transit, we have $C_{\text{token}} = 1$. By InvA (1), we have the condition.
  - $gSize = 0$: $C_{\text{token}} = 1$ implies $gSize = 0$ by InvF (19).
  - $type_i = \bot$: By InvC (5), we have $mode_i \neq \text{INCS}$. By InvE (14), the condition holds.
  - $acks_i = \emptyset$ holds by InvH (30) and $tok_i = \bot$.
  - $acqs_i = \emptyset$ holds by InvH (30) and $tok_i = \bot$.
  - $\neg leaving_i$ holds by InvJ (33) and $tok_i = \bot$.
  - $home_i = \bot$ holds by $type_i = \bot$ and InvE (17).
- A $\langle \text{subtoken}, t \rangle$ message from P_k:
  - $tok_i = \bot$: When P_k sends the message, $tok_k \neq \bot$ holds. Then, $gSize > 0$ becomes true when P_k sends the message. By InvA (1), InvB (2) and InvF (19), $tok_k \neq \bot$ is true when P_i receives the message. Thus, the condition holds by InvA (1).
  - $type_i = \bot$: By InvC (5), we have $mode_i \neq \text{INCS}$. By InvE (14), the condition holds.
  - $home_i = \bot$ holds by $type_i = \bot$ and InvE (17).
  - $\neg leaving_i$ holds by $tok_i = \bot$ and InvJ (33).
- An $\langle \text{acquired} \rangle$ message from P_k:
  - $holder_i = \bot$: Since $S_{\text{t},j,\ell} = C_{\text{t},j,\ell} + R_{\text{t},j,\ell}$ for any message t and any process j and ℓ, we have a relation $S_{\text{leave},k,i} - R_{\text{ack},i,k} = C_{\text{leave},k,i} + C_{\text{ack},i,k}$ by InvI (31). Just before the message is sent by P_k, $S_{\text{leave},k,i} - R_{\text{ack},i,k} = 0$ holds by InvH (29) since $acks_k = \emptyset$ holds. Thus, we have $C_{\text{leave},k,i} = C_{\text{ack},i,k} = 0$. At the same time, $S_{\text{acquired},k,i} - S_{\text{leave},k,i} = 0$ holds by InvH (28) since $acqs_k = \emptyset$ holds. By FIFO property of a channel, we have $C_{\text{acquired},k,i} = 0$ because $S_{\text{acquired},k,i} - S_{\text{leave},k,i} = 0$ and $C_{\text{leave},k,i} = 0$ hold. This implies $R_{\text{acquired},k,i} = R_{\text{leave},k,i}$.

    Just after the message is sent by P_k, we have $S_{\text{acquired},k,i} - S_{\text{leave},k,i} = 1$. By FIFO property of a channel, $R_{\textsf{acquired},k,i} = R_{\textsf{leave},k,i}$ is maintained until P_i receives the message. This implies the condition by InvI (32).

- A $\langle\textsf{request}\rangle$ message: Precondition is trivially satisfied.
- A $\langle\textsf{release}\rangle$ message from P_k:
  - *tok*$_i \neq \perp$: Just before the message arrives at P_i, $S_{\textsf{subtoken}i,k} - R_{\textsf{release},k,i} > 0$ holds by InvG (23). By InvG (24), the condition holds.
  - *gSize* > 0: Just before the message is received, $C_{\textsf{release}} > 0$ holds. This implies *gSize* > 0 by InvB (2).
  - $\neg$*leaving*$_i$: Since *gSize* > 0, the condition is implied by InvJ (37).
- A $\langle\textsf{leave}\rangle$ message from P_k:
  - *holder*$_i = P_k$: Just before the message is sent by P_k, $S_{\textsf{acquired},k,i} - S_{\textsf{leave},k,i} = 1$ holds by InvH (27). By FIFO property of a channel, we have $R_{\textsf{acquired},k,i} - 1 = R_{\textsf{leave},k,i}$ just before the message arrives at P_i. Hence, the condition is implied by InvI (32).
- An $\langle\textsf{ack}\rangle$ message from P_k:
  - $P_k \in$ *acks*$_i \neq \emptyset$: By InvH (29), it is easy to see that $0 \leq S_{\textsf{leave},i,k} - R_{\textsf{ack},k,i} \leq 1$ holds. By relation $S_{\textsf{leave},i,k} - R_{\textsf{ack},k,i} = C_{\textsf{leave},i,k} + C_{\textsf{ack},k,i}$ from InvI (31), we have $S_{\textsf{leave},i,k} - R_{\textsf{ack},k,i} = 1$ since $C_{\textsf{ack},k,i} > 0$ holds. By InvH (29), we have $B_{acks,i,k} = 1$ and hence *acks*$_i \ni P_k$.
  - *acqs*$_i = \emptyset$ holds by InvJ (35).
  - *leaving*$_i$ is implied by *acks*$_i \neq \emptyset$ and InvJ (36).
  - *tok*$_i \neq \perp$ is implied by *leaving*$_i$ and InvJ (33).
  - *gSize* $= 0$ holds by *leaving*$_i$ and InvJ (37).

Thus, for each message type, precondition of a message handler is satisfied. ■

Lemma 4: (Induction step, precondition of procedure call) Suppose that a handler is invoked with precondition being satisfied when all the invariants are maintained. Then, precondition of any procedure invoked from the handler is also satisfied.

Proof: We check each invocation of procedure in each handler and procedure.

- In procedure *handlePendingRequests*: Procedure *beginTokenTransfer* is invoked at line 4.10.
  - *tok*$_i \neq \perp$ holds by precondition.
  - *gSize* $= 0$ holds by line 4.1.
  - $\neg$*leaving*$_i$ holds by line 4.1.
  - *acqs*$_i \neq \emptyset$ holds by line 4.9.
  - *acks*$_i = \emptyset$ holds by $\neg$*leaving*$_i$ and InvJ (36).
- In procedure *beginTokenTransfer*: This invokes no procedure.
- In *requestEvent* hander: It invokes procedure *handlePendingRequests* at line 2.5:
  - *tok*$_i \neq \perp$ holds by line 2.2.
- In *releaseEvent* hander: Procedure *handlePendingRequests* is invoked at line 3.6.
  - *tok*$_i \neq \perp$ holds by *type*$_i$ = MAIN at line 3.1 and precondition.
- In token message handler: Procedure *handlePendingRequests* is invoked at line 5.14.
  - *tok*$_i \neq \perp$: Assume that the message is sent by P_k. The message is sent at lines 4.13 or 12.14. Then, *tok*$_k \neq \perp$ holds when P_k sends the message by condition at line 4.1 (in case of line 4.13) and condition at line 12.4 (in case of line 12.14). Thus, the value in *tok* in the token is not $\perp$. Thus, by line 5.1, the precondition *tok*$_i \neq \perp$ is satisfied.
- In subtoken, acquired and leave message handlers: No procedure is invoked.
- In request message handler: Procedure *handlePendingRequests* is invoked at line 8.5.
  - *tok*$_i \neq \perp$ holds by line 8.1.
- In release message handler: Procedure *handlePendingRequests* is invoked at line 9.4.
  - *tok*$_i \neq \perp$ holds by precondition.
- In ack message hander: Procedure *handlePendingRequests* is invoked at line 12.11.
  - *tok*$_i \neq \perp$ holds by line 12.4.

■

Lemma 5: (Induction step, invariants) Suppose that a handler is invoked with precondition being satisfied and all the invariants are satisfied just before the handler is invoked. Then, execution of a handler maintains all the invariants.

Proof: First, we check if each procedure maintains the invariants or not by assuming that precondition is satisfied on invocation.

- Procedure *handlePendingRequests*:
  - Lines 4.4–4.6. The value of *type*$_i$, *gSize*, *gName*, *mode*$_i$ and *reqQ* are changed. We check only invariants in which these values appear.
    * InvB (2): Before the statements are executed, *mode*$_i$ = TRYING holds by InvC (3, 4 and 5) since the request of P_i is in *reqQ*. By InvE (14), we have *type*$_i = \perp$. Thus, by the assignment statements, both T_{MAIN} and *gSize* are incremented by one, respectively, and hence the invariant is maintained.
    * InvC (5): Before the assignment statements, we have exactly one item of P_i's request in *reqQ* and *ts*$_i$ = *tsReq*$[i]$ by InvC (8), and $C_{\textsf{token},*,i} = C_{\textsf{subtoken},*,i} = 0$ by InvC (3, 4 and 5). After the assignment statements are executed, we have *ts*$_i$ = *tsReq*$[i]$, no item of P_i's request in *reqQ*, and $C_{\textsf{token},*,i} = C_{\textsf{subtoken},*,i} = 0$. Thus the invariant is maintained.
    * InvC (3 and 4): Since we have *mode*$_i$ = INCS, these invariants are trivially maintained.
    * InvC (8): Since we have no item of P_i's request in *reqQ*, the invariant is maintained.
    * InvD (9): Before the assignment statements, we have *mode*$_i$ = TRYING by InvC (3,4 and 5) since a request item of P_i is in *reqQ*. This implies that *grp*$_i \neq \perp$ by InvD (10). After the statements are executed, we have *gSize* $= 1$ and *gName* = *grp*$_i \neq \perp$. Thus, the invariant is maintained.
    * InvD (10): Before the assignment statements, we have *grp*$_i \neq \perp$. After the statements are executed, we have *mode*$_i$ = INCS. Thus, the invariant is maintained.
    * InvD (12): Before the assignment statements, we have *mode*$_i$ = TRYING. By condition *gSize* $= 0$ at line 4.1, we have $C_{\textsf{subtoken}} = 0$ by InvB (2), and hence $C_{\textsf{subtoken},*,i} = 0$ holds. Thus, the invariant is trivially maintained.
    * InvD (13): Since *gSize* $= 0$, *mode*$_j \neq$ INCS for any $P_j (\neq P_i)$. Thus, by assignment statement, the invariant is clearly maintained.

* InvE (17): Before the assignment statements, we have $home_i = \bot$ because $gSize = 0$ holds by line 4.1 and by InvB (2) and InvE (17). Thus, the assignment maintains the invariant.
* InvE (14): It is obvious by line 4.6.
* InvE (15): It is obvious since $home_i = \bot$ holds and line 4.6.
* InvJ (37): Since we have $\neg leaving_i$ by line 4.1, the invariant is maintained.

– Line 4.10. As we will see shortly, invocation of *beginTokenTransfer* maintains all the invariants.

– Lines 4.12 – 4.13. The value of V_{token}, C_{token} and tok_i are changed.

* InvA (1) is obviously maintained.
* InvE (15): Because $gSize = 0$ (line 4.1) implies $type_i = \bot$ by InvB (2), the invariant is maintained.
* InvC (4): Before the statement is executed, we have $mode_j = \text{TRYING}$ and $ts_j = tsReq[j]$ by InvC (3, 4 and 5) since there is an item of P_j's request in *reqQ*. Specifically, the second term of InvC (4) holds for P_j. By InvC (8), there is exactly one item of P_j's request in *reqQ*. By InvC (4), we have $C_{\mathsf{token},*,j} = C_{\mathsf{subtoken},*,j} = 0$.
  After execution of the statements, we have $ts_j = tsReq[j]$, no item of P_j's request in *reqQ*, $C_{\mathsf{token},*,j} = 1$, and $C_{\mathsf{subtoken},*,j} = 0$, and the third term of InvC (4) holds for P_j. Thus, the invariant is maintained.
* InvC (3 and 5): Since we have $mode_j = \text{TRYING}$, these invariants are trivially maintained.
* InvF (19) is maintained since $gSize = 0$ holds by line 4.1.
* InvF (20 and 21): Before P_i sends the main-token, we have $tok_i \neq \bot$, and thus $tok_j = \bot$ for each $P_j (\neq P_i)$ holds. Hence $acqs_j = acks_j = \emptyset$ holds for each $P_j (\neq P_i)$ by InvH (30). Because $acqs_i = \emptyset$ holds by line 4.9, InvF (20) is maintained. After the main-token is sent, since we have $\neg leaving_i$ by line 4.1, $acks_i = \emptyset$ holds by InvJ (36), and thus InvF (21) is maintained.
* InvJ (33) is maintained because $\neg leaving_i$ holds by line 4.1.
* InvJ (34) is maintained because $\neg leaving_i$ holds by line 4.1 and InvJ (36).

– Lines 4.21 and 4.23 – 4.24. The value of *gSize*, $type_i$, $mode_i$ and *reqQ* are changed.

* InvB (2): Before the assignments, $mode_i = \text{TRYING}$ holds by InvC (3, 4 and 5) since there is an item of P_i's request in *reqQ*. By InvE (14), we have $type_i = \bot$. Thus, by execution of the statements, T_{MAIN} and *gSize* are incremented by one, respectively. Thus, the invariant is maintained.
* InvC (5): Before the statements are executed, we have $mode_i = \text{TRYING}$, $ts_i = tsReq[i]$ and $C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0$ by InvC (3, 4 and 5). By InvC (8), there is exactly one item of P_i's request in *reqQ*. After execution of the statements, we have $mode_i = \text{INCS}$ and no item of P_i's request in *reqQ*. Thus, the invariant is maintained.
* InvC (3 and 4): Since we have $mode_i = \text{INCS}$, these invariants are trivially maintained.
* InvC (8): Since there is no item of P_i's request in *reqQ*, the invariant is maintained.
* InvD (9): Before the statements are executed, statements at lines 4.10 and 4.13 are not executed so that the condition of the **while** statement at line 4.17 is satisfied, because their executions yield $leaving_i$ or $tok_i = \bot$. This implies that the condition of **if** statement at line 4.3 is satisfied. Therefore, we have $gSize = 1$ and $gName \neq \bot$ by line 4.5. Thus, execution of the statements maintains the invariant.
* InvD (10): Before the statements are executed, we have $mode_i = \text{TRYING}$ by InvC (3, 4 and 5) since a request item of P_i is in *reqQ*. By InvD (11), we have $grp_i \neq \bot$. After the statements are executed, we have $mode_i = \text{INCS}$ and grp_i is unchanged. Thus, the invariant is maintained.
* InvD (11): Removing a request item of P_i trivially maintains the invariant.
* InvD (13): By condition at line 4.19, the invariant is maintained.
* InvE (14) is clearly maintained.
* InvE (15): $tok_i \neq \bot$ holds by condition at line 4.17, and $home_i = \bot$ holds by $tok_i \neq \bot$ and InvE (15 and 17). Thus, the invariant is maintained.
* InvE (17) is maintained since $home_i = \bot$ holds.
* InvJ (37) is maintained since $\neg leaving_i$ holds by line 4.17.

– Lines 4.21 and 4.27 – 4.28. The value of *gSize*, C_{subtoken} and $S_{\mathsf{subtoken},i,j}$ are changed.

* InvB (2) is obviously maintained.
* InvC (4): Before execution of the statements, by the same discussion for lines 4.22 – 4.23, we have $mode_j = \text{TRYING}$, $ts_j = tsReq[j]$, $C_{\mathsf{token},*,j} = C_{\mathsf{subtoken},*,j} = 0$, and there is exactly one item of P_j's request in *reqQ*. After the execution of the statements, we have no item of P_j's request in *reqQ* and $C_{\mathsf{subtoken},*,j} = 1$. Thus the invariant is maintained.
* InvC (3 and 5): Since we have $mode_j = \text{TRYING}$, these invariants are trivially maintained.
* InvC (8): Since we have no item of P_j's request in *reqQ*, the invariant is maintained.
* InvD (12): Before the statements are executed, we have $grp_i = g$ where g is the group such that $\langle P_j, t, g \rangle \in reqQ$. By condition at line 4.19, we have $grp_i = g = gName$. Thus, the invariant is maintained.
* InvF (19) is maintained because $C_{\mathsf{token}=}0$ holds by $tok_i \neq \bot$ and InvA (1).
* InvG (22 and 23) is maintained since $S_{\mathsf{subtoken},i,j}$ and $C_{\mathsf{subtoken},i,j}$ are simply incremented by one, respectively.
* InvG (24) is maintained since $tok_i \neq \bot$ holds.
* InvJ (37) is maintained since $\neg leaving_i$ holds by line 4.17.

• Procedure *beginTokenTransfer*:

– Line 10.1 – 10.3. The values of $leaving_i$, C_{leave}, S_{leave}, $acks_i$ and $acqs_i$ are changed.

* InvF (20 and 21): Because $tok_i \neq \bot$ holds by precondition, $C_{\mathsf{token}} = 0$ holds. Thus, the invariants

are maintained.

  * InvH (27): Before the statements are executed, $S_{\mathsf{acquired},i,j} - S_{\mathsf{leave},i,j} = B_{acqs,i,j} = 1$ holds for each $P_j \in acqs_i$ by InvH (28). At the same time, $S_{\mathsf{leave},i,k} - R_{\mathsf{acks},k,i} = B_{acks,i,k} = 0$ for any P_k by precondition $acks_i = \emptyset$ and InvH (29).
    After the statements are executed, we have $S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = B_{acqs,i,k} = 0$ for any P_k, $acqs_i = \emptyset$, and $S_{\mathsf{acquired},i,k} - R_{\mathsf{acks},k,i} = B_{acks,i,k}$ for any P_k. Thus the invariant is maintained.
  * InvH (28) is maintained because $acqs_i = \emptyset$ holds after the assignment statements are executed.
  * InvH (29) is maintained because $S_{\mathsf{leave},i,j} - R_{\mathsf{ack},j,i} = 1$ holds for each $P_j \in acks_i$ after the assignment statements are executed.
  * InvH (30) is maintained by precondition $tok_i \neq \perp$.
  * InvJ (33) is maintained by precondition $tok_i \neq \perp$.
  * InvJ (34) is maintained since $tok_i \neq \perp$.
  * InvJ (35) is maintained since $acqs_i = \emptyset$ and $acks_i \neq \emptyset$ hold by the assignment statements.
  * InvJ (36) is maintained since $acks_i \neq \emptyset$ holds by the assignment statements.
  * InvJ (37) is maintained by precondition $gSize = 0$.

Next, we show that each handler maintains invariants.

- *requestEvent* hander:
  - Lines 2.1 and 2.3 – 2.4. The values of $mode_i$, ts_i, grp_i, $tsReq[i]$ and $reqQ$ are changed.
    * InvC (4): Before the statements are executed, we have no item of P_i's request in $reqQ$ and $C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0$ by InvC (3) with precondition $mode_i = \text{IDLE}$. By execution of the statements, we have $mode_i = \text{TRYING}$, $ts_i = tsReq[i]$ and there is exactly one item of P_i's request in $reqQ$. Thus the invariant is maintained.
    * InvC (3 and 5): Since we have $mode_i = \text{TRYING}$, these invariants are trivially maintained.
    * InvC (8): Since there is exactly one item of P_i's request in $reqQ$ after execution of the statements, the invariant is maintained.
    * InvD (10): Since $g_i \neq \perp$, the invariant is clearly maintained.
    * InvD (11): Before the statements are executed, we have $mode_i = \text{IDLE}$ by precondition. By InvC (3), there is no item of P_i's request in $reqQ$. After the statements are executed, we have only one item of P_i's request such that $\langle P_i, ts_i, grp_i \rangle$ in $reqQ$. Thus, the invariant is maintained.
    * InvD (12): We have $C_{\mathsf{subtoken},*,i} = 0$ by InvC (3) and by $mode_i = \text{IDLE}$. Thus the invariant is trivially maintained.
    * InvD (13): Since we have $mode_i = \text{TRYING}$ after the statements are executed, the invariant is trivially maintained.
    * InvE (14) is maintained by precondition $type_i = \perp$.
  - Lines 2.1 and 2.7. The values of $mode_i$, ts_i, grp_i and $\mathcal{M}_{\mathsf{request}}$ are changed.
    * InvC (4): Before execution of the statements, we have $ts_i = tsReq[i]$, no item of P_i's request in $reqQ$ and $C_{\mathsf{token},*,i} = C_{\mathsf{subtoken},*,i} = 0$. After execution of the statements, we have $ts_i = tsReq[i] + 1$ and $mode_i = \text{TRYING}$. Thus, the invariant is maintained.
    * InvC (3 and 5) are trivially maintained since we have $mode_i = \text{TRYING}$.
    * InvC (6): Since the value of ts_i is incremented by one, and a $\mathsf{request}$ message issued contains the value. Thus, the invariant is maintained.
    * InvD (10, 12 and 13) are maintained by the same proof for the case of lines 2.1 and 2.3 – 2.4 shown above.
    * InvD (11): Before the statements are executed, there is no request item of P_i. Thus, the invariant is trivially maintained because $reqQ$ is unchanged.
    * InvE (14) is maintained since $type_i = \perp$ by precondition.
  - Lines 2.1 and 2.9. The value of $mode_i$, ts_i, grp_i and $\mathcal{M}_{\mathsf{request}}$ are changed. This case is shown by simply applying the case of lines 2.1 and 2.7 for each $P_j \in (q_i - \{P_i\})$.
  - Lines 2.1 and 2.10 – 2.13. The value of $tmpQ_i$ is changed.
    * InvC (7): Since the value of ts_i is incremented by one at line 2.1, $\forall \langle P_i, t, g \rangle \in reqQ_i : t < ts_i$ holds. Thus, deleting such items does not violate the invariant since an item with timestamp value ts_i is enqueued.
- *releaseEvent* hander:
  - Lines 3.2 – 3.5. The values of $type_i$, $mode_i$, grp_i, $gSize$ and $gName$ are changed.
    * InvB (2) is maintained since both T_{MAIN} and $gSize$ are decremented by one, respectively.
    * InvC (3): Before the statements are executed, $mode_i = \text{INCS}$ holds by precondition. Since InvC (5) holds before execution of the statements, the invariant is maintained by assignment statement at line 3.2.
    * InvC (4 and 5) are trivially maintained since $mode_i = \text{IDLE}$ holds.
    * InvD (9) is maintained because the value of $gName$ becomes $\perp$ iff the value of $gSize$ becomes zero.
    * InvD (10) is clearly maintained.
    * InvD (11): Before the statements are executed, there is no request item of P_i in $reqQ$ by InvC (5). Thus, the invariant is trivially maintained.
    * InvD (12): Before the statements are executed, we have $C_{\mathsf{subtoken},*,i} = 0$ by precondition $mode_i = \text{INCS}$ and InvC (5). Thus, the invariant is trivially maintained.
    * InvD (13) is trivially maintained because we have $mode_i = \text{IDLE}$ by execution of the statements.
    * InvE (14) is maintained since we have $mode_i = \text{IDLE}$ and $type_i = \perp$ by the assignment statements.
    * InvE (17): Before the statements are executed, we have $home_i = \perp$ by InvE (15) and $type_i = \text{MAIN}$ by line 3.1. After the statements are executed, we have $type_i = \perp$ and $home_i = \perp$. Thus the invariant is maintained.
    * InvF (19) is maintained because we have $C_{\mathsf{token}} = 0$ by InvA (1) and precondition $tok_i \neq \perp$.
    * InvJ (37) is maintained by precondition $\neg leaving_i$.
  - Line 3.6. Because all invariants are maintained before

invocation of procedure *handlePendingRequests*, all invariants are maintained.

– Lines 3.9 – 3.11. The values of S_{release}, C_{release}, type_i, mode_i, grp_i and home_i are changed.
  * InvB (2) is maintained since T_{SUB} is decremented by one and C_{release} is incremented by one.
  * InvC (3): Before the statements are executed, $\mathit{mode}_i = \text{INCS}$ holds by precondition. Since InvC (5) holds before execution of the statements, the invariant is maintained by assignment statement at line 3.11.
  * InvC (4 and 5) are trivially maintained since $\mathit{mode}_i = \text{IDLE}$ holds.
  * InvD (10) is clearly maintained by the assignments.
  * InvD (11): Before the statements are executed, there is no item of P_i's request in *reqQ* by InvC (5). Thus, the invariant is trivially maintained.
  * InvD (12) Before the statements are executed, we have $C_{\text{subtoken},*,i} = 0$ by precondition $\mathit{mode}_i = \text{INCS}$ and InvC (5). Thus, the invariant is trivially maintained.
  * InvD (13) is trivially maintained because we have $\mathit{mode}_i = \text{IDLE}$ by execution of the statements.
  * InvE (14) is maintained since we have $\mathit{mode}_i = \text{IDLE}$ and $\mathit{type}_i = \bot$ by the statements.
  * InvE (17): We have $\mathit{type}_i = \bot$ and $\mathit{home}_i = \bot$ by execution of the statements. Thus the invariant is maintained.
  * InvE (18) is obviously maintained.
  * InvG (25): Before sending a release message, we have $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 1$ by InvG (26) and $\mathit{home}_i \neq \bot$ by precondition. By sending the message, we have $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 0$. Thus the invariant is maintained.
  * InvG (26): Since we have $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 0$ and $\mathit{home}_i = \bot$ by execution, the invariant is maintained.

- token message handler:
  – Receipt of the message and Line 5.1. The value of C_{token}, V_{tok} and tok_i are changed.
    * InvA (1): Before the message arrives, $\mathit{tok}_i = \bot$ holds by InvA (1). By receiving the message, V_{tok} is incremented by one and C_{token} is decremented by one. Hence the invariant is maintained.
    * InvC (5): Before the message is received, we have $C_{\text{token},*,i} = 1$. Thus, by InvC (3, 4 and 5), we have $\mathit{mode}_i = \text{TRYING}$ and $\mathit{ts}_i = \mathit{tsReq}[i]$. After the statement is executed, we have $C_{\text{token},*,i} = 0$ and $\mathit{mode}_i = \text{INCS}$ at line 5.4. Thus, the invariant is maintained.
    * InvC (3 and 4): Since we have $\mathit{mode}_i = \text{INCS}$ at line 5.4, these invariants are trivially maintained.
    * InvE (15 and 16) are maintained by precondition $\mathit{type}_i = \bot$.
    * InvF (19, 20 and 21) are maintained since we have $C_{\text{token}} = 0$ by the assignment statement.
    * InvH (30) is maintained since we have $\mathit{tok}_i \neq \bot$ by the assignment statement.
    * InvJ (33 and 34) are maintained since we have $\mathit{tok}_i \neq \bot$.
  – Lines 5.2 – 5.3. The values of acqs_i and $S_{\text{acquired},i,j}$ are changed.
    * InvF (20) is maintained since we have $C_{\text{token}} = 0$.
    * InvH (27): Before the statements are executed, we have $\mathit{acqs}_i = \emptyset$ by precondition, which implies $B_{\mathit{acqs},i,k} = 0$ for any P_k. By InvH (28), for any P_k, we have $S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 0$. By precondition $\mathit{acks}_i = \emptyset$, we have $S_{\text{leave},i,k} - R_{\text{ack},k,i} = 0$ for for any P_k by InvH (29). Thus, we have $S_{\text{acquired},i,k} - R_{\text{ack},k,i} = 0$ by InvH (27).
    After the statements are executed, we have $S_{\text{acquired},i,j} - S_{\text{leave},i,j} = 1$ and $S_{\text{acquired},i,j} - R_{\text{ack},j,i} = 1$. Thus, the invariant is maintained.
    * InvH (28): Before the statements are executed, $S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 0$ holds for any P_k. Thus, after the statements are executed, we have $S_{\text{acquired},i,j} - S_{\text{leave},i,j} = 1$ iff $P_j \in \mathit{acqs}_i$, and hence the invariant is maintained.
    * InvH (30) is trivially maintained since we have $\mathit{tok}_i \neq \bot$ by line 5.1.
  – Lines 5.4 – 5.5. The values of type_i, mode_i, *gSize* and *gName* are changed.
    * InvB (2): Before the statements, $\mathit{type}_i = \bot$ holds by precondition. Since T_{MAIN} and *gSize* are incremented by one, respectively, the invariant is maintained.
    * InvD (9): Since $\mathit{mode}_i = \text{TRYING}$ holds before the statements are executed, we have $\mathit{grp}_i \neq \bot$ by InvD (10). Thus, execution of the statements maintains the invariant.
    * InvD (10): Since we have $\mathit{mode}_i = \text{INCS}$ and $\mathit{gName} = \mathit{grp}_i \neq \bot$, the invariant is maintained.
    * InvD (13): Before the statements are executed, $\mathit{gName} = \bot$ holds by precondition $\mathit{gSize} = 0$ and InvD (9). Thus, $\mathit{mode}_j \neq \text{INCS}$ for any $P_j (\neq P_i)$. Thus, the invariant is maintained by execution of the statements.
    * InvE (14) is clearly maintained by the assignments.
    * InvE (15) is maintained since we have $\mathit{tok}_i \neq \bot$ by line 5.1 and $\mathit{home}_i = \bot$ by precondition.
    * InvE (17) is maintained since $\mathit{home}_i = \bot$ and $\mathit{type}_i = \text{MAIN}$.
    * InvF (19) is obviously maintained since $C_{\text{token}} = 0$ holds.
    * InvJ (37) is maintained by precondition $\neg \mathit{leaving}_i$.
  – Lines 5.7 – 5.13. The values of tmpQ_i, *tsReq* and *reqQ* are changed.
    * InvC (8): We claim that there is no item of P_j's request in *reqQ* if a request of P_j is enqueued into *reqQ* at line 5.11. Suppose contrary that there exists a request item $\langle P_j, t, g \rangle \in \mathit{reqQ}$. Since P_j's request is in *reqQ*, we have $\mathit{mode}_j = \text{TRYING}$ and $\mathit{ts}_j = \mathit{tsReq}[j]$ by InvC (3, 4 and 5). By InvC (7), we have $\forall \langle P_j, t', g' \rangle : t' \leq \mathit{ts}_j$. By condition of enqueue at line 5.9, each request item of P_j in tmpQ_i is never enqueued into *reqQ*; a contradiction. Therefore, the invariant is maintained.
    * InvC (4): Request items that are not enqueued into *reqQ* never violate the invariant. Thus, we consider each request item $\langle P_j, t, g \rangle$ which is enqueued.

Consider just before the request item of P_j is dequeued. Since *tsReq*$[j] < t$ holds, by InvC (3, 4 and 5), we have *mode*$_j$ = TRYING and $C_{\mathsf{token},*,j} = C_{\mathsf{subtoken},*,j} = 0$. Thus, after execution of lines 5.10 and 5.11 for P_j, we have *tsReq*$[j] = t$ and an request item of P_j in *reqQ*. Thus, InvC (4) is maintained for P_j.

  - ∗ InvC (3 and 5) are trivially maintained since *mode*$_j$ = TRYING holds for each P_j whose request item is enqueued into *reqQ*$_i$.
  - ∗ InvC (7) is maintained since *tmpQ*$_i$ becomes empty.
  - ∗ InvD (11): Let P_j be any process whose request item is enqueued at line 5.11. Then, as discussed above, *mode*$_j$ = TRYING holds. Let $\langle P_j, t, g\rangle$ be the request item of P_j in question. By InvC (3, 4, 5 and 7), *tok.tsReq*$[j] < t$ implies *ts*$_j = t$. Thus, we have $g =$ *grp*$_j$ by execution of the statements, and hence the invariant is maintained.

  Since all invariants are maintained just before procedure *handlePendingRequests* is invoked, all invariants are maintained.

- subtoken message handler:
  - – Receipt of the message and Line 6.1. The values of *type*$_i$, *mode*$_i$, *home*$_i$, $R_{\mathsf{subtoken},k,i}$ and $C_{\mathsf{subtoken},k,i}$ are changed. Just before the statements are executed, we have $R_{\mathsf{subtoken},k,i} - S_{\mathsf{release},i,k} = 0$ holds by InvG (25 and 26) since *home*$_i = \perp$ holds by precondition.
    - ∗ InvB (2): Before the message is received, *type*$_i = \perp$ holds by precondition. By receiving the message and the assignment statements, C_{subtoken} is decremented by one and T_{SUB} is incremented by one. Thus the invariant is maintained.
    - ∗ InvC (5): Before the message is received, we have $C_{\mathsf{subtoken},*,i} > 0$. By InvC (3, 4 and 5), we have *mode*$_i$ = TRYING, *ts*$_i$ = *tsReq*$[i]$, no request item of P_i in *reqQ* and $C_{\mathsf{subtoken},*,i} = 1$. After executing the statements, we have *mode*$_i$ = INCS and $C_{\mathsf{subtoken},*,i} = 0$. Thus the invariant is maintained.
    - ∗ InvC (3 and 4): Since we have *mode*$_i$ = INCS, these invariants are trivially maintained.
    - ∗ InvD (10): Before the statements are executed, *grp*$_i \neq \perp$ holds by InvD (10) since *mode*$_i$ = TRYING holds. Thus, execution of the statements maintains the invariant.
    - ∗ InvD (12): Since $C_{\mathsf{subtoken},*,i}$ is decremented by one, the invariant is trivially maintained.
    - ∗ InvD (13): Since *gName* = *grp*$_i$ holds before the message is received, the invariant is maintained.
    - ∗ InvE (14) is obviously maintained by the assignments.
    - ∗ InvE (16) is maintained by precondition *tok*$_i = \perp$.
    - ∗ InvE (17) is obviously maintained.
    - ∗ InvE (18) is maintained since $P_k \neq P_i$ by lines 4.22 and 4.28.
    - ∗ InvG (25) is maintained because $R_{\mathsf{subtoken},k,i} - S_{\mathsf{release},i,k} = 1$ holds by receiving the message.
    - ∗ InvG (26) is maintained since we have *home*$_i = P_k$ by the assignment statements.
- acquired message handler:
  - – Receipt of the message and Line 7.1. The values of *holder*$_i$, $C_{\mathsf{acquired},k,i}$ and $R_{\mathsf{acquired},k,i}$ are changed.
    - ∗ InvI (32): Before the statements are executed, *holder*$_i = \perp$ holds by precondition, and hence $R_{\mathsf{acquired},j,i} - R_{\mathsf{leave},j,i} = 0$ holds for any P_j by InvI (32). After the statements are executed, we have $R_{\mathsf{acquired},j,i} - R_{\mathsf{leave},j,i} = 1$ and *holder*$_i = P_k$, which implies the invariant.
- request message handler:
  - – Receipt of the message and Lines 8.2 – 8.4. The value of $\mathcal{M}_{\mathsf{request}}$, *tsReq*$[k]$ and *reqQ* are changed.
    - ∗ InvC (8): In case *tsReq*$[k] \geq t$ holds, the invariant is maintained. Thus, we consider a case that *tsReq*$[k] < t$ holds. We claim that there is no request item of P_k in *reqQ*. Suppose contrary that there exists a request item of P_k in *reqQ*. Before the statements are executed, by InvC (4), we have *ts*$_k$ = *tsReq*$[k]$ holds. Since the value of *ts*$_k$ is non-decreasing, we have $t \leq$ *ts*$_k$. Thus, we have $t \leq$ *ts*$_k$ = *tsReq*$[k]$. This is a contradiction because *tsReq*$[k] < t$ holds by assumption. Therefore, there is no request item of P_k in *reqQ*, and hence the invariant is maintained.
    - ∗ InvC (4): Before the statements are executed, there is no request item of P_k in *reqQ* and *tsReq*$[k] < t \leq$ *ts*$_k$ holds. By InvC (4), we have *mode*$_k$ = TRYING, $C_{\mathsf{token},*,k} = C_{\mathsf{subtoken},*,k} = 0$ and *ts*$_k$ = *tsReq*$[k] + 1$. Thus, we have $t =$ *ts*$_k$. After the statements are executed, we have *tsReq*$[k] = t =$ *ts*$_k$, and a request item of P_k is enqueued into *reqQ*. Thus, the invariant is maintained.
    - ∗ InvC (3 and 5): Since we have *mode*$_k$ = TRYING, the invariants are trivially maintained.
    - ∗ InvC (6) is maintained since an item is removed from $\mathcal{M}_{\mathsf{request}}$.
    - ∗ InvC (8) is maintained since there is no request item of P_k before execution of the statements.
    - ∗ InvD (11): In case the request item of P_k is not enqueued into *reqQ*, the invariant is trivially maintained. Consider a case that it is enqueued. Then, as discussed above, *mode*$_k$ = TRYING holds. Let $\langle P_k, t, g\rangle$ be the request item in question. By InvC (3, 4, 5 and 6), *tok.tsReq*$[k] < t$ implies *ts*$_k = t$. Thus, we have $g =$ *grp*$_k$ by execution of the statements, and hence the invariant is maintained.
  - – Line 8.5. All invariants are maintained because all invariants are satisfied before invocation of procedure *handlePendingRequests*.
  - – Receipt of the message and Line 8.8. The value of $\mathcal{M}_{\mathsf{request}}$ is changed.
    - ∗ InvC (6): Since a request message is simply enqueued into a queue, the invariant is maintained.
  - – Receipt of the message and Lines 8.10 – 8.11. The values of $\mathcal{M}_{\mathsf{request}}$ and *tmpQ* are changed.
    - ∗ InvC (6): Since a request message is simply forwarded, the invariant is maintained.
    - ∗ InvC (7): By non-decreasing property of *ts*$_k$, we have $t \leq t_k$. Thus, the invariant is maintained.
- release message handler: Let P_k be the process that sent the message.

- Receipt of the message and Lines 9.1 – 9.3. The value of $C_{\mathsf{release},k,i}$, $R_{\mathsf{release},k,i}$, *gSize* and *gName* are changed.
  - InvB (2) is maintained since $C_{\mathsf{release},k,i}$ and *gSize* are decremented by one, respectively.
  - InvF (19) is maintained since we have $C_{\mathsf{token}} = 0$ by InvA (1) and precondition $tok_i \neq \perp$.
  - InvG (22): Before the message is received, $C_{\mathsf{release},k,i} > 0$ holds, which implies $S_{\mathsf{subtoken},i,k} - R_{\mathsf{release},k,i} > 0$ by InvG (23). By receiving the message, we have $S_{\mathsf{subtoken},i,k} - R_{\mathsf{release},k,i} \geq 0$, and hence the invariant is maintained.
  - InvG (23) is maintained because, by receiving the message, $C_{\mathsf{release},k,i}$ is decremented by one, and $R_{\mathsf{release},k,i}$ is incremented by one.
  - InvG (24) is maintained by precondition $tok_i \neq \perp$.
  - InvD (9) is maintained because the value of *gName* becomes $\perp$ iff the value of *gSize* becomes zero.
  - InvD (12): Before the message is received, we have $gSize = 1$ and $C_{\mathsf{release}} = 1$, and by InvB (2), we have $C_{\mathsf{subtoken},*,i} = 0$. Thus, the invariant is trivially maintained.
  - InvD (13) is maintained because $mode_i \neq$ INCS holds before the message is received by InvB (2).
  - InvJ (37) is maintained by precondition $\neg leaving_i$.
- Line 9.4. All invariants are maintained because all of them are maintained just before procedure *handlePendingRequests* is called.

- leave message handler: Let P_k be the process that sent the message.
  - Receipt of the message and Lines 11.1 – 11.2. The values of $R_{\mathsf{leave},k,i}$, $S_{\mathsf{ack},i,k}$ and $holder_i$ are changed.
    - InvI (31) is maintained because, by receiving the message, $R_{\mathsf{leave},k,i}$ and $S_{\mathsf{ack},i,k}$ are incremented by one, respectively.
    - InvI (32): Before the message is received, we have $R_{\mathsf{acquired},k,i} - R_{\mathsf{leave},k,i} = 1$ by InvI (32) because $holder_i = P_k$ by precondition. After receiving the message and execution of the statements, $R_{\mathsf{acquired},k,i} - R_{\mathsf{leave},k,i} = 0$ and $holds_i = \perp$ hold. Thus the invariant is maintained.

- ack message handler: Let P_k be the process that sends the message.
  - Receipt of the message and Lines 12.1 – 12.3. The values of $R_{\mathsf{ack},k,i}$, $acks_i$, $B_{acks,i,k}$ and $leaving_i$ are changed.
    - InvF (20) is maintained because $C_{\mathsf{token}} = 0$ holds by InvA (1) and $tok_i \neq \perp$ by precondition.
    - InvH (27): Before the message is received, $B_{acks,i,k} = 1$ holds by precondition $P_k \in acks_i$. By InvH (29), we have $S_{\mathsf{leave},i,k} - R_{\mathsf{ack},k,i} = 1$. By precondition $acqs_i = \emptyset$, we have $B_{acqs,i,k} = 0$, which implies $S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = 0$ by InvH (28). By receiving the message, we have $S_{\mathsf{leave},i,k} - R_{\mathsf{ack},k,i} = 0$, and hence we have $S_{\mathsf{acquired},i,k} - R_{\mathsf{ack},k,i} = S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = 0$. Thus the invariant is maintained.
    - InvH (29) is maintained because we have $S_{\mathsf{leave},i,k} - R_{\mathsf{ack},k,i} = B_{acks,i,k} = 0$ by receiving the message and execution of the assignments.
    - InvH (30) is trivially maintained by precondition $tok_i \neq \perp$.
    - InvJ (33 and 34) are maintained by precondition $tok_i \neq \perp$.
    - InvJ (35) is maintained by precondition $acqs_i = \emptyset$.
    - InvJ (36): We have $\neg leaving_i$ (line 12.3) when $acks_i = \emptyset$ (line 12.2), and $leaving_i$ by precondition is kept unchanged when $acks_i \neq \emptyset$. Thus, the invariant is maintained.
    - InvJ (37) is maintained by precondition $gSize = 0$.
  - Line 12.5. Execution of this line is discussed in conjunction with lines 12.9 – 12.10 and 12.14. See below.
  - Lines 12.7 – 12.8. The values of $acqs_i$ and $S_{\mathsf{acquire},i,j}$ are changed.
    - InvH (27): Before the message is received, $acks_i = \{P_k\}$ holds by condition at line 12.2. Thus, after the message is received. we have $S_{\mathsf{acquired},i,\ell} - R_{\mathsf{ack},\ell,i} = 0$ for any P_ℓ by InvH (27).

      Before the statements are executed, we have $B_{acqs,i,\ell} = 0$ for any P_ℓ by precondition $acqs_i = \emptyset$. Thus, by InvH (28), $S_{\mathsf{acquired},i,\ell} - S_{\mathsf{leave},i,\ell} = 0$ holds for any P_ℓ.

      After the statements are executed, we have $S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = S_{\mathsf{acquired},i,k} - R_{\mathsf{ack},k,i} = 1$ iff $P_k \in acqs_i$. Thus the invariant is maintained.
    - InvH (28): After the statements are executed, we have $S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = 1$ if and only if $P_k \in acqs_i$. This is equivalent to $S_{\mathsf{acquired},i,k} - S_{\mathsf{leave},i,k} = B_{acks,i,k}$ for each $P_k \in acqs_i$, and hence the invariant is maintained.
    - InvH (30) is maintained by precondition $tok_i \neq \perp$.
    - InvJ (35) is maintained since $acks_i = \emptyset$ by line 12.2.
  - Line 12.9 – 12.10 with line 12.5. The values of *reqQ*, $type_i$, $mode_i$, *gSize* and *gName* are changed.
    - InvB (2): Before the statements are executed, we have $type_i = \perp$ by InvB (2) because $gSize = 0$ holds by precondition, and hence $T_{\text{MAIN}} = 0$ holds. By execution of the statements, T_{MAIN} and *gSize* are incremented by one, respectively. Thus the invariant is maintained
    - InvC (5): Before the statements are executed, we have $mode_i =$ TRYING, $ts_i = tsReq[i]$ and $C_{\mathsf{token}} = C_{\mathsf{subtoken}} = 0$ by InvC (3, 4 and 5) because a request item of P_i is in *reqQ*. After the statements are executed, $mode_i =$ INCS holds and the request item is removed. Thus, the invariant is maintained.
    - InvC (3 and 4) are trivially maintained since we have $mode_i =$ INCS.
    - InvC (8) is clearly maintained by dequeue operation on *reqQ*.
    - InvD (9): Before the statements are executed, a request item of P_i is in *reqQ*. This implies $mode_i =$ TRYING by InvC (3, 4 and 5). By InvD (10), we have $grp_i \neq \perp$ because $mode_i =$ TRYING holds. Thus, the assignment statements maintain the invariant.
    - InvD (10) is maintained since we have $mode_i =$ INCS and $grp_i \neq \perp$.
    - InvD (11) is trivially maintained since a request item of P_i is simply dequeued.
    - InvD (12): Before the statements are executed,

$C_{\mathsf{subtoken},*,i} = 0$ holds by precondition $gSize = 0$ and InvB (2). Thus, the invariant is trivially maintained.

  * InvD (13): Before the statements are executed, $mode_j \neq$ INCS holds for any P_j by precondition $gSize = 0$ and InvB (2). After the statements are executed, P_i is the only process such that $mode_i =$ INCS. Thus, the invariant is maintained.
  * InvE (14) is maintained since we have $mode_i =$ INCS and $type_i =$ MAIN by execution of the statements.
  * InvE (15): We have $type_i = \perp$ before the statements are executed. By InvE (15 and 17), $type_i = \perp$ implies $home_i = \perp$. Because we have $type_i =$ MAIN by execution of the statements and $tok_i \neq \perp$ by precondition, the invariant is maintained.
  * InvE (17) is maintained because we have $type_i =$ MAIN and $tok_i \neq \perp$.
  * InvJ (37) is maintained since $\neg leaving_i$ holds by line 12.3.
- Line 12.11. All the invariants are maintained since all of them are satisfied before procedure *handlePendingRequests* is invoked.
- Line 12.14 with line 12.5. The values of C_{token}, V_{tok} and tok_i are changed.
  * InvA (1): Before execution of the statements, we have $V_{tok} = 1$ by precondition, and hence $C_{\mathsf{token}} = 0$ holds. Thus, by execution of the statements, we have $V_{tok} = 0$ and $C_{\mathsf{token}} = 1$.
  * InvC (4): Before the statements are executed, we have $mode_i =$ TRYING, $ts_i = tsReq[i]$ and $C_{\mathsf{token}} = C_{\mathsf{subtoken}} = 0$, as discussed for the case of line 12.9 – 12.10. After the statements are executed, we have $C_{\mathsf{token}} = 1$ and the request item is removed. Thus, InvC (4) is maintained.
  * InvC (3 and 5) are trivially maintained since we have $mode_i =$ TRYING.
  * InvC (8) is clearly maintained by dequeue operation on *reqQ*.
  * InvE (15): Before the statements are executed, we have $type_i = \perp$ by InvB (2) because of precondition $gSize = 0$. Thus, the invariant trivially holds.
  * InvH (30) is maintained since $acqs_i = \emptyset$ holds by precondition and $acks_i = \emptyset$ holds by line 12.2.
  * InvJ (33 and 34) are maintained since $tok_i \neq \perp$ holds. ■

Lemma 6: Suppose that $requestDone_i$ event is triggered assuming that (1) precondition of message handler or procedure, in which the event is triggered, is satisfied, and (2) all the invariants are satisfied on invocation of message handler or procedure. Then, precondition of *releaseEvent* handler becomes true, and it remains true until *releaseEvent* handler is invoked.

Proof: First, let us observe when $requestDone_i$ is triggered. The event triggered at lines 4.7, 4.25, 5.6, 6.2 and 12.12.

- Line 4.7 (in procedure *handlePendingRequests*):
  - $mode_i =$ INCS and $type_i =$ MAIN by line 4.6.
  - $tok_i \neq \perp$ and $\neg leaving_i$ by line 4.1.
  - $gSize > 0$ by line 4.5.
  - When procedure *handlePendingRequests* is invoked, $gSize = 0$ holds, and hence we have $acks_i = \emptyset$ by InvJ (36) and $\neg leaving_i$. The value $acks_i$ is unchanged in the procedure.
- Line 4.25 (in procedure *handlePendingRequests*):
  - $mode_i =$ INCS and $type_i =$ MAIN by line 4.24.
  - $tok_i \neq \perp$ and $\neg leaving_i$ by line 4.17.
  - $gSize > 0$ by line 4.23.
  - By the same reason as the case for line 4.7, we have $acks_i = \emptyset$.
- Line 5.6 (in token message handler):
  - $mode_i =$ INCS and $type_i =$ MAIN by line 5.4.
  - $tok_i \neq \perp$ by line 5.1.
  - $\neg leaving_i$ by precondition.
  - $gSize > 0$ by line 5.5.
  - $acks_i = \emptyset$ by precondition.
- Line 6.2 (in subtoken message handler):
  - $mode_i =$ INCS and $type_i =$ SUB by line 6.1.
  - $tok_i = \perp$ by precondition.
  - $\neg leaving_i$ by precondition.
  - $acks_i = \emptyset$ by InvJ (36).
- Line 12.12 (in ack message handler)
  - $mode_i =$ INCS and $type_i =$ MAIN by line 12.9.
  - $tok_i \neq \perp$ by line 12.4 (and precondition).
  - $\neg leaving_i$ by line 12.3.
  - $gSize > 0$ by line 12.10.
  - $acks_i = \emptyset$ by line 12.2.

Thus, when $requestDone_i$ event is triggered, precondition of *releaseEvent* handler is satisfied.

Next, we show that precondition of *releaseEvent* handler is maintained until *releaseEvent* handler is triggered.

- $mode_i =$ INCS is maintained because the value of $mode_i$ becomes IDLE only in *releaseEvent* handler (lines 3.2 and 3.11). Since *requestEvent* handler is not invoked before *releaseEvent* handler is invoked, the value of $mode_i$ is maintained.
- $type_i =$ MAIN $\vee$ $type_i =$ SUB is maintained by InvE (14).
- When $type_i =$ MAIN, $tok_i \neq \perp$ holds by InvE (15).
- When $type_i =$ MAIN, $gSize > 0$ holds InvB (2).
- When $type_i =$ SUB, $tok_i = \perp$ holds by InvE (16).
- When $type_i =$ SUB, $home_i \neq \perp$ holds by InvE (17).
- $\neg leaving_i$ holds by InvJ (33) and InvJ (37).
- $acks_i = \emptyset$ holds by $\neg leaving_i$ and InvJ (36).

Thus, the precondition of *releaseEvent* handler is maintained by invariants until *releaseEvent* handler is invoked. ■

Lemma 7: Suppose that $releaseDone_i$ event is triggered assuming that (1) precondition of message handler or procedure, in which the event is triggered, is satisfied, and (2) all the invariants are satisfied on invocation of message handler or procedure. Then, precondition of *requestEvent* handler becomes true and it remains true until *requestEvent* handler is invoked.

Proof: Note that $releaseDone_i$ event is triggered at line 3.13 only. Since we have $type_i = \perp$ and $mode_i =$ IDLE at lines 3.2 and 3.11 when $releaseDone_i$ event is triggered. Thus, precondition of *requestEvent* handler is satisfied when $releaseDone_i$ event is triggered.

Next, we show that the precondition is maintained until *requestEvent* handler is invoked.

- By InvC (6), each request message in transit issued by P_i has timestamp value less than or equal to ts_i. Thus, each request message issued by P_i is not enqueued into *reqQ* because $ts_i = tsReq[i]$ holds.

- There is no request item of P_i in *reqQ* by InvC (3).
- There is no token and subtoken message in transit to P_i by InvC (3).

Thus, P_i never receive any token and subtoken message until P_i invokes *requestEvent* handler. Because the value of $mode_i$ is not changed, the value of $mode_i$ is maintained. By InvE (14), we have $type_i = \perp$. Since the value of $mode_i$ is maintained, the value of $type_i$ is also maintained.

Therefore, the precondition of *requestEvent* handler is maintained until *requestEvent* handler is invoked. ■

Theorem 1: For any execution, the proposed algorithm maintains all the invariants, and each handler is invoked with precondition being satisfied. ■

C. Safety property

Theorem 2: (*Safety*) For any execution, no two processes in different groups are in their critical sections simultaneously.

Proof: Assume contrary that there exists an execution and two processes P_i and P_j such that

$$(mode_i = mode_j = \text{InCS}) \wedge (grp_i \neq grp_j).$$

By InvB (2), we have $gSize > 0$. This implies $gName \neq \perp$ by InvD (9). By InvD (13), we have $gName = grp_i = grp_j$; a contradiction. ■

D. Liveness property

Theorem 3: (*Liveness*) A process that makes a request eventually enters its critical section, provided that priority scheme of the queue of the main-token is non-starving.

Proof: Assume contrary that there exists an execution such that a requesting process P_i does not enter its critical section forever. Let grp_i be the group that P_i is requesting.

First, we consider a case that the request in question is enqueued into *reqQ* but the request is not granted forever. Because priority scheme of *reqQ* is non-starving, eventually the priority of the request becomes the highest among any requests. Thus, any execution eventually reaches a point such that **peek** (resp., **dequeue**) operation always returns (resp., extracts) the request of P_i for any future execution.

Consider after the priority of the request of P_i becomes the highest. Then, no more token and subtoken message is issued. Since each process in its critical section eventually exits, the value of *gSize* eventually becomes zero by InvB (2) and InvF (19).

- Case 1, the value of *gSize* becomes zero when the request is in *reqQ* and the priority of the request is the highest: The value of *gSize* is decremented at lines 3.3 or 9.1. In any case, procedure *handlePendingRequests* is invoked in which P_i is granted.
- Case 2, otherwise, i.e., the value of *gSize* is zero when the request is enqueued into *reqQ* with the highest priority: The request is enqueued into *reqQ* at lines 2.4, 5.11 or 8.4. In any case, procedure *handlePendingRequests* is invoked in which P_i is granted.

Therefore, in any case, the request of P_i is eventually granted.

Next, we consider a case that the request of P_i is never enqueued into *reqQ*. There are three cases when P_i makes a request, i.e., when P_i invokes *requestEvent* handler.

- Case A1, $tok_i \neq \perp$ (line 2.2): The request is always enqueued in the main-token (line 2.4).
- Case A2, $tok_i = \perp$ and $holder_i \neq \perp$ (line 2.6): P_i sends a request message directly to P_k, where $P_k = holder_i$.
  We claim that $tok_k \neq \perp$ holds when the request message arrives at P_k. By InvI (32) and $holder_i = P_k$, we have $R_{\mathsf{acquired},k,i} - R_{\mathsf{leave},k,i} = 1$. Starting from this formula, by InvI (31), we have

$$\begin{aligned} & R_{\mathsf{acquired},k,i} - R_{\mathsf{leave},k,i} \\ = \; & R_{\mathsf{acquired},k,i} - S_{\mathsf{ack},i,k} \\ = \; & (S_{\mathsf{acquired},k,i} - C_{\mathsf{acquired},k,i}) - (C_{\mathsf{ack},i,k} + R_{\mathsf{ack},i,k}) \\ = \; & S_{\mathsf{acquired},k,i} - R_{\mathsf{ack},i,k} - C_{\mathsf{ack},i,k} - C_{\mathsf{acquired},k,i} \\ = \; & 1. \end{aligned}$$

  Because $0 \leq S_{\mathsf{acquired},k,i} - R_{\mathsf{ack},i,k} \leq 1$ by InvH (27), we must have $S_{\mathsf{acquired},k,i} - R_{\mathsf{ack},i,k} = 1$ and $C_{\mathsf{acquired},k,i} = C_{\mathsf{ack},i,k} = 0$.
  - In case $S_{\mathsf{acquired},k,i} - S_{\mathsf{leave},k,i} = 0$: Since $S_{\mathsf{leave},k,i} - R_{\mathsf{ack},i,k} = 1$ holds, $acks_k \neq \emptyset$ by InvH (29).
  - Otherwise, i.e., in case $S_{\mathsf{acquired},k,i} - S_{\mathsf{leave},k,i} = 1$: $acqs_k \neq \emptyset$ by InvH (28).

  Thus, in any case, we have $tok_k \neq \perp$ by InvH (30), and the claim is shown.
  Therefore, the request of P_i is enqueued into *reqQ* when the request message arrives at P_k.
- Case A3, $tok_i = \perp$ and $holder_i = \perp$ (line 2.8): In this case, P_i sends a request message for each process in a quorum q_i except P_i itself (line 2.9), and it enqueues the request into $tmpQ_i$ (line 2.11) if $P_i \in q_i$.
  There are two cases to consider.
  - There exists $P_j \in (q_i - \{P_i\})$ such that $holder_j \neq \perp$ holds when P_j receives the request message from P_i. Then, P_j forwards the request to P_k, where $P_k = holder_j$. As claimed in case A2, $tok_k \neq \perp$ holds when the request message arrives at P_k.
  - Otherwise, i.e., $holder_j = \perp$ holds for any $P_j \in (q_i - \{P_i\})$ when P_j receives the request message from P_i. Suppose that this condition remains true forever. Then, by intersection property of coterie and a property of message delivery in finite time, there is no acquire and leave messages in transit in the system, i.e., $C_{\mathsf{acquired}} = C_{\mathsf{leave}} = 0$, and $holder_\ell = \perp$ holds for any P_ℓ. By InvI (32), we have $R_{\mathsf{acquired},\ell,i} = R_{\mathsf{leave},\ell,i}$ for any P_ℓ. At the same time, by InvI (31) and InvH (27), eventually we have $S_{\mathsf{acquired},\ell,i} = S_{\mathsf{leave},\ell,i} = R_{\mathsf{ack},i,\ell}$ for any P_ℓ. Consider when $S_{\mathsf{acquired},\ell,i} = S_{\mathsf{leave},\ell,i} = R_{\mathsf{ack},i,\ell}$ becomes true for any P_ℓ. By InvH (27), the formula becomes true by receiving an ack message at P_ℓ. By receiving the message, we have $acqs_\ell = acks_\ell = \emptyset$ by InvH (28 and 29). This implies that
    - ∗ acquired messages are sent (line 12.8), or
    - ∗ a token message is sent to other process, say P_k, and acquired messages are sent on receipt of the message (line 5.3).

    Note that $P_k = P_i$ never holds in case a token message is sent, because otherwise, a request item of P_i must be in *reqQ*.
    Let P_k be a process that sends the acquired messages. By intersection property of coterie, there exists $P_j \in q_i \cap q_k$ such that $holder_j \neq \perp$ eventually becomes true. Thus, the condition assumed cannot be true forever.

TABLE I
CONSUMED RESOURCE FOR MODEL CHECKING.

Property	# states of model	time (sec)	memory (Mbyte)
Safety	1.37×10^6	14	418
Liveness	3.46×10^6	168	3103

Then, the request of P_i in *tmpQ*$_j$ is forwarded to P_k on receipt of acquired message (line 7.4), and the request item is eventually enqueued as claimed in case A2.

Thus, there exists no execution such that a requesting process P_i never enter its critical section forever. ■

II. VERIFICATION BY MODEL CHECKING

We carried out a model checking [1] with model checker SPIN [2], [3] to verify the proposed algorithm. Because a model checking verifies correctness properties for all asynchronous execution patterns of a system exhaustively, it is a useful tool for finding a bug in a concurrent system.

Note that there is an execution such that (1) the value of timestamp is unbounded, and (2) the value of C_{release}, the number of release messages in transit, is unbounded. Because a system to be verified must be finite by limitation of current model checking technology, in our model, (1) abstraction is introduced such that timestamps are in a bounded range, and (2) asynchrony is restricted such that C_{release} is bounded.

We verified the following correctness properties.

- Safety: No two different groups are accessed simultaneously at any time.
- Liveness: A process that makes a request eventually enters its critical section.

Model checking is performed for a distributed system with the following settings.

- The number of processes is 3.
- The number of groups is 3.
- Coterie used is a majority coterie C_{maj}.
- Each process non-deterministically selects a quorum on initialization.
- Each process non-deterministically selects a group when it makes a request.
- Priority scheme of the queue of the main-token is FCFS.

Our computing environment is as follows.

- IBM IntelliStation A Pro with dual Opteron 245 (2.8GHz clock) and 10G byte memory
- RedHat Enterprise Linux WS4 for AMD64/EM64T
- Model checker: SPIN version 4.2.5
- C compiler: GCC version 3.4.2

Our algorithm is successfully verified by model checker, and resource consumed for verification is shown in Table I.

REFERENCES

[1] Edmund M. Clarke, Jr., Orna Grumberg, and Dron A. Peled, *Model Checking*, The MIT Press, 1999.
[2] Gerard J. Holzmann, “The model checker Spin,” *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.
[3] Gerard J. Holzmann, *The SPIN Model Checker*, Addison Wesley, 2003.