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

Hirotugu 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.
  \[ V_{tok} + C_{token} = 1 \]  
  (1)

- InvB: \(tok.gSize\) counts the number of tokens in use.
  \[ T_{MAIN} + C_{subtoken} + T_{SUB} + C_{release} = gSize \]  
  (2)

- InvC: Maintenance of requests and timestamp values.
  \[ \forall P_i : mode_i = IDLE \Rightarrow \]
  \[ (ts_i = ts_{Req[i]} ) \land (\forall (P, t, g) \in reqQ : P \neq P_i ) \land (C_{token},+,i = C_{subtoken},+,i = 0) \]  
  (3)

  \[ \forall P_i : mode_i = TRYING \Rightarrow \]
  \[ (ts_i = ts_{Req[i]} + 1 ) \land (\forall (P, t, g) \in reqQ : P \neq P_i ) \land (C_{token},+,i = C_{subtoken},+,i = 0) \]
  \[ \land (C_{token},i = 1 ) \land (C_{subtoken},+,i = 0) \land (C_{token},+,i = 0 ) \land (C_{subtoken},+,i = 0) \land (C_{token},+,i = 0 ) \land (C_{subtoken},+,i = 0) \]  
  (4)

  \[ \forall P_i : mode_i = INCS \Rightarrow \]
  \[ (ts_i = ts_{Req[i]} ) \land (\forall (P, t, g) \in reqQ : P \neq P_i ) \land (C_{token},+,i = 0 ) \land (C_{subtoken},+,i = 0) \]  
  (5)

- InvD: Current group is maintained by \(gName\).
  \[ gSize = 0 \Leftrightarrow gName = \bot \]
  \[ \land \forall P_i : (mode_i \neq IDLE \Leftrightarrow grp_i \neq \bot) \]  
  (6)

- InvE: Process mode and local variables.
  \[ \forall P_i : (mode_i = INCS \Leftarrow (type_i = MAIN \lor type_i = SUB)) \]  
  (7)

  \[ \land (type_i = MAIN \Rightarrow (tok_i \neq \bot \land home_i = \bot)) \]  
  (8)

  \[ \land (type_i = SUB \Rightarrow tok_i = \bot) \]  
  (9)

  \[ \land (home_i \neq P_i) \]  
  (10)

- InvF: Transfer of the main-token.
  \[ (C_{token} = 1 ) \Rightarrow \]
  \[ (gSize = 0 ) \land \forall P_i : (acqs_i = \emptyset) \]  
  (11)

  \[ \land \forall P_i : (acks_i = \emptyset) \]  
  (12)

- InvG: Management of the main-token and sub-tokens.
  \[ (\forall P_k : 0 \leq S_{subtoken,i,k} = R_{release,i,k} ) \]  
  (13)

  \[ \land (\forall P_k : C_{subtoken,i,k} + P_{type,k,sub} + C_{release,i,k} = S_{subtoken,i,k} - R_{release,k,i} ) \]  
  (14)

  \[ \land (\forall P_k : S_{subtoken,i,k} = R_{release,i,k} > 0 \Rightarrow tok_i \neq \bot) \]  
  (15)

  \[ \land \forall P_i : (\forall P_k : 0 \leq R_{subtoken,k,i} \leq S_{release,i,k} \leq 1) \]  
  (16)

  \[ \land \forall P_i : (R_{subtoken,k,i} = S_{release,i,k} = 1 \Rightarrow home_i = P_k) \]  
  (17)
\[ V_{i,k} \] 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 \( tmpQ, \) for some \( P_i \).  
\( T_{\text{MAIN}} \) (resp. \( T_{\text{SUB}} \)) is the number of process \( P_i \) such that \( type_i = \text{MAIN} \) (resp. \( \text{SUB} \)).  
\( C_i \) is the total number of messages of type \( t \) in transit.  
\( C_{i,k,i} \) (resp. \( C_{i,t,k} \)) is the number of messages of type \( t \) in transit from \( P_i \) (resp. any process) to \( P_k \).  
\( M_i \) is the set of all messages of type \( t \) in transit.  
\( S_{t,i,k} \) is the set of all requests in \( P_i \) to \( P_k \) since the system startups.  
\( R_{i,k} \) is the total number of \texttt{send} of message of type \( t \) from \( P_i \) to \( P_k \) since the system startups.  
\( B_{acqs,i,j} \) is 1 if \( acqs_i \supseteq P_j \) and 0 otherwise.  
\( B_{acks,i,j} \) is 1 if \( acks_i \supseteq P_j \) and 0 otherwise.  
\( B_{type.k,Sub} \) is 1 if \( type_k = \text{SUB} \) and 0 otherwise.

Fig. 1. Symbols used in invariants.

- **InvH**: Sending acquired, leave, ack messages.
  \[
  \forall P_i \forall P_k : \
  \begin{align*}
  & (0 \leq S_{\text{acquired},i,k} - S_{\text{leave},i,k} \\
  & \quad \leq S_{\text{acquired},i,k} - R_{\text{ack},k,i} \leq 1) \\
  & \land (S_{\text{acquired},i,k} - S_{\text{leave},i,k} - B_{\text{acqs},i,k}) \\
  & \land (S_{\text{leave},i,k} - R_{\text{ack},k,i} = B_{\text{acqs},i,k}) \\
  & \land (tok_i = \bot \Rightarrow acqs_i = acks_i = \emptyset)
  \end{align*}
  \]  
  \[
  \forall P_i \forall P_k : \n  \begin{align*}
  & (R_{\text{leave},k,i} = S_{\text{ack},i,k}) \\
  & \land (R_{\text{acquired},k,i} - R_{\text{leave},k,i} = 1) \Leftrightarrow \text{holder}_i = P_k
  \end{align*}
  \]

- **InvI**: Receiving acquired, leave, ack messages and maintenance of \texttt{holder}_i.
  \[
  \forall P_i : \n  \begin{align*}
  & (tok_i = \bot \Rightarrow -\text{leaving}_i) \\
  & \land (tok_i = \bot \Rightarrow acks_i = \emptyset) \\
  & \land (acks_i = \emptyset \Rightarrow acqs_i = \emptyset) \\
  & \land (\text{leaving}_i \Leftrightarrow acks_i \neq \emptyset) \\
  & \land (\text{leaving}_i \Rightarrow gSize = 0)
  \end{align*}
  \]

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 \textit{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 \textit{token} message is sent, the invariant is maintained. When a \textit{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 \textit{token} message in transit. We denote, by simply \( gSize \) in invariant description, \( tok.gSize \) 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 \texttt{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 \textit{(token, tok, t)} message from \( P_k \):
  - \( tok_i = \bot \): Since a \textit{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.
      - \( acks_i = 0 \) holds by InvH (30) and \( tok_i = \bot \).
      - \( acqs_i = 0 \) holds by InvH (30) and \( tok_i = \bot \).
    - \( \neg \text{leaving}_i \) holds by InvJ (33) and \( tok_i = \bot \).
    - \( \text{home}_i = \bot \) holds by \( type_i = \bot \) and InvE (17).

- A \textit{(subtoken, t)} message from \( P_k \):
  - \( tok_i = \bot \): When \( P_k \) sends the message, \( tok_k \neq \bot \) holds. Then, \( gSize \geq 0 \) 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.
      - \( \text{home}_i = \bot \) holds by \( type_i = \bot \) and InvE (17).
      - \( \neg \text{leaving}_i \) holds by \( tok_i = \bot \) and InvJ (33).

- An \textit{(acquired)} message from \( P_k \):
  - \( \text{holder}_i = \bot \): Since \( S_{t,i,\ell} = C_{t,j,\ell} + R_{t,j,\ell} \) for any message \( t \) and any process \( j \) and \( \ell \), we have a relation \( S_{\text{leave},k,i} - R_{\text{ack},k,i} = C_{\text{leave},k,i} + C_{\text{ack},k,i} \) by InvA (31). Just before the message is sent by \( P_k \), \( S_{\text{leave},k,i} - R_{\text{ack},k,i} \) holds by InvH (29) since \( acks_k = 0 \) holds. Thus, we have \( C_{\text{leave},k,i} = C_{\text{ack},k,i} = 0 \). At the same time, \( S_{\text{acquired},k,i} - S_{\text{leave},k,i} = 0 \) holds by InvH (28) since \( acqs_k = 0 \) 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, \( R_{\text{acquired},k,i} = R_{\text{leave},k,i} \) is maintained until \( P_i \) receives the message. This implies the condition by InvI (32).

- A (request) message: Precondition is trivially satisfied.

- A (release) message from \( P_k \):
  - \( tok_i \neq \bot \): Just before the message arrives at \( P_k \), \( S_{\text{subtoken},k,i} - R_{\text{release},k,i} \geq 0 \) holds by InvG (23). By InvG (24), the condition holds.
  - \( g\text{Size} > 0 \): Just before the message is received, \( C_{\text{release}} \geq 0 \) holds. This implies \( g\text{Size} > 0 \) by InvB (2).
  - \( \neg \text{leaving} \): Since \( g\text{Size} > 0 \), the condition is implied by InvJ (37).

- A (leave) message from \( P_k \):
  - \( hold_{i} = P_{k} \): Just before the message is sent by \( P_k \), \( S_{\text{acquired},k,i} - S_{\text{leave},k,i} = 1 \) holds by InvH (27). By FIFO property of a channel, we have \( R_{\text{acquired},k,i} - 1 = R_{\text{leave},k,i} \) just before the message arrives at \( P_i \). Hence, the condition is implied by InvJ (32).

- An (ack) message from \( P_k \):
  - \( P_{k} \in \text{acks}_{i} \neq 0 \): By InvH (29), it is easy to see that \( \leq S_{\text{leave},i,k} - R_{\text{ack},k,i} \leq 1 \) holds. By relation \( S_{\text{leave},i,k} - R_{\text{ack},k,i} = C_{\text{leave},i,k} + C_{\text{ack},k,i} \) from InvJ (31), we have \( S_{\text{leave},i,k} - R_{\text{ack},k,i} = 1 \) since \( C_{\text{ack},k,i} > 0 \) holds. By InvH (29), we have \( B_{\text{ack},i,k} = 1 \) and hence \( \text{acks}_{i} \supseteq P_{k} \).
  - \( \text{acqs}_{i} = 0 \) holds by InvJ (35).
  - \( \text{leaving}_{i} \) is implied by \( \text{acqs}_{i} \neq 0 \) and InvJ (36).
  - \( tok_{i} \neq \bot \) is implied by \( \text{leaving}_{i} \) and InvJ (33).
  - \( g\text{Size} = 0 \) holds by \( \text{leaving}_{i} \) 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 satisfied. 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.
  - \( tok_{i} \neq \bot \) holds by precondition.
  - \( g\text{Size} = 0 \) holds by line 4.1.
  - \( \neg \text{leaving} \), holds by line 4.1.
  - \( \text{acqs}_{i} \neq 0 \) holds by line 4.9.
  - \( \text{acqs}_{i} = 0 \) holds by \( \neg \text{leaving} \), and InvJ (36).

- In procedure beginTokenTransfer: This invokes no procedure.

- In requestEvent handler: It invokes procedure handlePendingRequests at line 2.5:
  - \( tok_{i} \neq \bot \) holds by line 2.2.

- In releaseEvent handler: Procedure handlePendingRequests is invoked at line 3.6:
  - \( tok_{i} \neq \bot \) holds by \( \text{type}_{i} = \text{MAIN} \) at line 3.1 and precondition.

- In token message handler: Procedure handlePendingRequests is invoked at line 5.14:
  - \( tok_{i} \neq \bot \): Assume that the message is sent by \( P_k \). The message is sent at lines 4.13 or 12.14. Then, \( tok_k \neq \bot \) 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 \( \bot \). Thus, by line 5.1, the precondition \( tok_{i} \neq \bot \) 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.
  - \( tok_{i} \neq \bot \) holds by line 8.1.

- In release message handler: Procedure handlePendingRequests is invoked at line 9.4.
  - \( tok_{i} \neq \bot \) holds by precondition.

- In ack message handler: Procedure handlePendingRequests is invoked at line 12.11.
  - \( tok_{i} \neq \bot \) 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 \( \text{type}_{i} \), \( g\text{Size} \), \( g\text{Name} \), \( \text{mode}_{i} \) and \( \text{reqQ} \) are changed. We check only invariants in which these values appear.

  * InvB (2): Before the statements are executed, \( \text{mode}_{i} = \text{TRYING} \) holds by InvC (3, 4 and 5) since the request of \( P_i \) is in \( \text{reqQ} \). By InvE (14), we have \( \text{type}_{i} = \bot \). Thus, by the assignment statements, both \( T_{\text{MAIN}} \) and \( g\text{Size} \) 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 \( \text{reqQ} \) and \( ts_{i} = ts\text{Req}[i] \) by InvC (8), and \( C_{\text{token},+,i} = C_{\text{subtoken},+,i} = 0 \) by InvC (3, 4 and 5). After the assignment statements are executed, we have \( ts_{i} = ts\text{Req}[i] \), no item of \( P_i \)'s request in \( \text{reqQ} \), and \( C_{\text{token},+,i} = C_{\text{subtoken},+,i} = 0 \). Thus the invariant is maintained.

  * InvC (3 and 4): Since we have \( \text{mode}_{i} = \text{INCS} \), these invariants are trivially maintained.

  * InvC (8): Since we have no item of \( P_i \)'s request in \( \text{reqQ} \), the invariant is maintained.

  * InvD (9): Before the assignment statements, we have \( \text{mode}_{i} = \text{TRYING} \) by InvC (3, 4 and 5) since a request item of \( P_i \) is in \( \text{reqQ} \). This implies that \( grp_{i} \neq \bot \) by InvD (10). After the statements are executed, we have \( g\text{Size} = 1 \) and \( g\text{Name} = grp_{i} \neq \bot \). Thus, the invariant is maintained.

  * InvD (10): Before the assignment statements, we have \( grp_{i} \neq \bot \). After the statements are executed, we have \( \text{mode}_{i} = \text{INCS} \). Thus, the invariant is maintained.

  * InvD (12): Before the assignment statements, we have \( \text{mode}_{i} = \text{TRYING} \). By condition \( g\text{Size} = 0 \) at line 4.1, we have \( C_{\text{subtoken}} = 0 \) by InvB (2), and hence \( C_{\text{subtoken},+,i} = 0 \) holds. Thus, the invariant is trivially maintained.

  * InvD (13): Since \( g\text{Size} = 0 \), \( \text{mode}_{i} \neq \text{INCS} \) 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 \(~leaving_i \) 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 (4), we have \( C_{token,*j} = C_{subtoken,*j} = 0 \).

- Line 4.1. As we will see shortly, invocation of beginTokenTransfer maintains all the invariants.
- Line 4.1 and InvJ (36).

After execution of the statements, we have \( ts_j = tsReq[j] \), no item of \( P_j \)’s request in \( reqQ \), \( C_{token,*j} = 0 \), and \( C_{subtoken,*j} = 0 \), 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 \( P_j \) sends the main-token, we have \( tok_k \neq \bot \), and thus \( tok_k = \bot \) 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 \( ~leaving_i \) by line 4.1, \( acks_i = \emptyset \) holds by InvJ (36), and thus InvF (21) is maintained.
* InvJ (33) is maintained because \( ~leaving_i \) holds by line 4.1.
* InvJ (34) is maintained because \( ~leaving_i \) 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 \( P_j \)’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 \( C_{token,*i} = C_{subtoken,*i} = 0 \) by InvC (3, 4 and 5). By InvC (8), there is exactly one item of \( P_j \)’s request in \( reqQ \).

After execution of the statements, we have \( mode_i = I N C S \) and no item of \( P_j \)’s request in \( reqQ \). Thus, the invariant is maintained.
* InvC (3 and 4): Since we have \( mode_i = I N C S \), 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 \( P_j \) is in \( reqQ \). By InvD (11), we have \( grp_i \neq \bot \). After the statements are executed, we have \( mode_i = I N C S \) 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): \( tok_k \neq \bot \) by condition at line 4.17, and \( home_i = \bot \) holds by \( tok_k \neq \bot \) and InvE (15 and 17). Thus, the invariant is maintained.
* InvE (17) is maintained since \( home_i = \bot \) holds.
* InvJ (37) is maintained since \( ~leaving_i \) holds by line 4.17.

- Lines 4.21 and 4.27 – 4.28. The value of \( gSize, C_{subtoken} \) and \( S_{subtoken,i,j} \) 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 \( mode_j = \text{TRYING}, ts_j = tsReq[j], C_{token,*j} = C_{subtoken,*j} = 0 \), 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 \( C_{subtoken,*j} = 1 \). 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 \( (P_j,t,g) \in reqQ \). By condition at line 4.19, we have \( grp_i = g = gName \). Thus, the invariant is maintained.
* InvF (19) is maintained because \( C_{token,0} = 0 \) holds by \( tok_i \neq \bot \) and InvA (1).
* InvG (22 and 23) is maintained since \( S_{subtoken,i,j} \) and \( C_{subtoken,i,j} \) are simply incremented by one, respectively.
* InvG (24) is maintained since \( tok_i \neq \bot \) holds.
* InvJ (37) is maintained since \( ~leaving_i \) 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 \( tok_k \neq \bot \) by precondition, \( C_{token} = 0 \) holds. Thus, the invariants
are maintained.

• **InvH (27):** Before the statements are executed, \( S_{\text{acquired},i,j} - S_{\text{leave},i,j} = B_{\text{acqs},i,j} = 1 \) holds for each \( P_j \in \text{acqs}_i \) by InvH (28). At the same time, \( S_{\text{leave},i,k} - R_{\text{acks},k,i} = B_{\text{acks},i,k} \) is 0 for any \( P_k \) by precondition \( \text{acks}_i = \emptyset \) and InvH (29).

After the statements are executed, we have \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = B_{\text{acqs},i,k} = 0 \) for any \( P_k \), \( \text{acqs}_i = \emptyset \), and \( S_{\text{acquired},i,k} - R_{\text{acks},k,i} = B_{\text{acks},i,k} \) for any \( P_k \). Thus the invariant is maintained.

• **InvH (28) is maintained because \( \text{acqs}_i = \emptyset \) holds after the assignment statements are executed.**

• **InvH (29) is maintained because \( S_{\text{leave},i,j} - R_{\text{acks},j,i} = 1 \) holds for each \( P_j \in \text{acks}_i \) after the assignment statements are executed.**

• **InvH (30) is maintained by precondition \( \text{tok}_i \neq \bot \).**

• **InvJ (33) is maintained by precondition \( \text{tok}_i \neq \bot \).**

• **InvJ (34) is maintained since \( \text{tok}_i \neq \bot \).**

• **InvJ (35) is maintained since \( \text{acqs}_i = \emptyset \) and \( \text{acks}_i \neq \emptyset \) hold by the assignment statements.**

• **InvJ (36) is maintained since \( \text{acks}_i \neq \emptyset \) holds by the assignment statements.**

• **InvJ (37) is maintained by precondition \( g\text{Size} = 0 \).**

Next, we show that each handler maintains invariants.

**requestEvent handler:**

- **Lines 2.1 and 2.3 – 2.4.** The values of \( \text{mode}_i, \text{ts}_i, \text{grp}_i, \) \( \text{tsReq}[i] \) and \( \text{reqQ} \) are changed.

- **InvC (4):** Before the statements are executed, we have no item of \( P_i \)'s request in \( \text{reqQ} \) and \( C_{\text{token},*,i} = C_{\text{subtoken},*,i} = 0 \) by InvC (3) with precondition \( \text{mode}_i = \text{IDLE} \). By execution of the statements, we have \( \text{mode}_i = \text{TRYING}, \text{ts}_i = \text{tsReq}[i] \) and there is exactly one item of \( P_i \)'s request in \( \text{reqQ} \). Thus the invariant is maintained.

- **InvC (3 and 5):** Since we have \( \text{mode}_i = \text{TRYING} \), these invariants are trivially maintained.

- **InvC (8):** Since there is exactly one item of \( P_i \)'s request in \( \text{reqQ} \) after execution of the statements, the invariant is maintained.

- **InvD (10):** Since \( g_i \neq \bot \), the invariant is clearly maintained.

- **InvD (11):** Before the statements are executed, we have \( \text{mode}_i = \text{IDLE} \) by precondition. By InvC (3), there is no item of \( P_i \)'s request in \( \text{reqQ} \). After the statements are executed, we have only one item of \( P_i \)'s request such that \( \langle P_i, \text{ts}_i, \text{grp}_i \rangle \) in \( \text{reqQ} \). Thus, the invariant is maintained.

- **InvD (12):** We have \( C_{\text{subtoken},*,i} = 0 \) by InvC (3) and by \( \text{mode}_i = \text{IDLE} \). Thus the invariant is trivially maintained.

- **InvD (13):** Since we have \( \text{mode}_i = \text{TRYING} \) after the statements are executed, the invariant is trivially maintained.

- **InvE (14) is maintained by precondition \( \text{type}_i = \bot \).**

- **Lines 2.1 and 2.7.** The values of \( \text{mode}_i, \text{ts}_i, \text{grp}_i \) and \( M_{\text{request}} \) are changed.

- **InvC (4):** Before execution of the statements, we have \( \text{ts}_i = \text{tsReq}[i] \), no item of \( P_i \)'s request in \( \text{reqQ} \) and \( C_{\text{token},*,i} = C_{\text{subtoken},*,i} = 0 \). After execution of the statements, we have \( \text{ts}_i = \text{tsReq}[i] + 1 \) and \( \text{mode}_i = \text{TRYING} \). Thus, the invariant is maintained.

- **InvV (3 and 5) are trivially maintained since we have \( \text{mode}_i = \text{TRYING} \).**

- **InvC (6):** Since the value of \( \text{ts}_i \) is incremented by one, and a 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 \( \text{reqQ} \) is unchanged.

- **InvE (14) is maintained since \( \text{type}_i = \bot \) by precondition.**

- **Lines 2.1 and 2.9.** The value of \( \text{mode}_i, \text{ts}_i, \text{grp}_i \) and \( M_{\text{request}} \) 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 \( \text{tmpQ}_i \) is changed.

- **InvC (7):** Since the value of \( \text{ts}_i \) is incremented by one at line 2.1, \( \forall (P_i, t,g) \in \text{reqQ}_i : t < \text{ts}_i \) holds. Thus, deleting such items does not violate the invariant since an item with timestamp value \( \text{ts}_i \) is enquired.

**releaseEvent handler:**

- **Lines 3.2 – 3.5.** The values of \( \text{type}_i, \text{mode}_i, \text{grp}_i, \text{gSize} \) and \( \text{gName} \) are changed.

- **InvB (2) is maintained since both \( T_{\text{MAIN}} \) and \( g\text{Size} \) are decremented by one, respectively.**

- **InvC (3):** Before the statements are executed, \( \text{mode}_i = \text{INCS} \) 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 \( \text{mode}_i = \text{IDLE} \).**

- **InvD (9) is maintained because the value of \( \text{gName} \) becomes \( \bot \) iff the value of \( \text{gSize} \) becomes zero.**

- **InvD (10) is clearly maintained.**

- **InvD (11):** Before the statements are executed, there is no request item of \( P_i \) in \( \text{reqQ} \) 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 \( \text{mode}_i = \text{INCS} \) and InvC (5). Thus, the invariant is trivially maintained.

- **InvD (13) is trivially maintained because we have \( \text{mode}_i = \text{IDLE} \) by execution of the statements.**

- **InvE (14) is maintained since we have \( \text{mode}_i = \text{IDLE} \) and \( \text{type}_i = \bot \) by the assignment statements.**

- **InvE (17):** Before the statements are executed, we have \( \text{home}_i = \bot \) by InvE (15) and \( \text{type}_i = \text{MAIN} \) by line 3.1. After the statements are executed, we have \( \text{type}_i = \bot \) and \( \text{home}_i = \bot \). Thus the invariant is maintained.

- **InvF (19) is maintained because we have \( C_{\text{token}} = 0 \) by InvA (1) and precondition \( \text{tok}_i \neq \bot \).**

- **InvJ (37) is maintained by precondition \( \text{~leaving}_i \).**

- **Line 3.6.** Because all invariants are maintained before
Lines 3.9 – 3.11. The values of $S_{\text{release}}$, $C_{\text{release}}$, $\text{type}_i$, $\text{mode}_i$, $\text{grp}_i$, $\text{home}_i$ are changed.
- InvB (2) is maintained since $T_{\text{SUB}}$ is decremented by one and $C_{\text{release}}$ is incremented by one.
- InvC (3): Before the statements are executed, $\text{mode}_i = \text{INCS}$ 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 $\text{mode}_i = \text{DLE}$ 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 $\text{reqQ}$ by InvC (5). Thus, the invariant is trivially maintained.
- InvD (12) Before the statements are executed, we have $C_{\text{subtoken},i,j} = 0$ by precondition $\text{mode}_i = \text{INCS}$ and InvC (5). Thus, the invariant is trivially maintained.
- InvD (13) is trivially maintained because we have $\text{mode}_i = \text{DLE}$ by execution of the statements.
- InvE (14) is maintained since we have $\text{mode}_i = \text{DLE}$ and $\text{type}_i = \bot$ by the statements.
- InvE (17): We have $\text{type}_i = \bot$ and $\text{home}_i = \bot$ by execution of the statements. Thus the invariant is maintained.
- InvE (18) is obviously maintained.
- InvG (25): Before sending a $\text{release}$ message, we have $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 1$ by InvG (26) and $\text{home}_i \neq \bot$ 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 $\text{home}_i = \bot$ by execution, the invariant is maintained.

*token* message handler:
- Receipt of the message and Line 5.1. The value of $C_{\text{token}}$, $V_{\text{tok}}$ and $tok_i$ are changed.
- InvA (1): Before the message arrives, $tok_i = \bot$ holds by InvA (1). By receiving the message, $V_{\text{tok}}$ is incremented by one and $C_{\text{token}}$ 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 $\text{mode}_i = \text{TRYING}$ and $ts_i = ts_{\text{req[i]}}$. After the statement is executed, we have $C_{\text{token},*,i} = 0$ and $\text{mode}_i = \text{INCS}$ at line 5.4. Thus, the invariant is maintained.
- InvC (3 and 4): Since we have $\text{mode}_i = \text{INCS}$ at line 5.4, these invariants are trivially maintained.
- InvE (15 and 16) are maintained by precondition $\text{type}_i = \bot$.
- 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 $tok_i \neq \bot$ by the assignment statement.
- InvJ (33 and 34) are maintained since we have $tok_i \neq \bot$.

Lines 5.2 – 5.3. The values of $\text{acqs}_i$ and $\text{acquired},i,j$ are changed.
- InvF (20) is maintained since we have $C_{\text{token}} = 0$.
- InvH (27): Before the statements are executed, we have $\text{acqs}_i = \emptyset$ by precondition, which implies $R_{\text{acqs},i,k} = 0$ 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 $\text{acks}_i = 0$, we have $S_{\text{leave},i,k} + R_{\text{ack},k,i} = 0$ 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 $P_j \in \text{acqs}_i$, and hence the invariant is maintained.
- InvH (30) is trivially maintained since we have $tok_i \neq \bot$ by line 5.1.

Lines 5.4 – 5.5. The values of $\text{type}_i$, $\text{mode}_i$, $\text{gSize}$ and $\text{gName}$ are changed.
- InvB (2): Before the statements, $\text{type}_i = \bot$ holds by precondition. Since $T_{\text{MAIN}}$ and $\text{gSize}$ are incremented by one, respectively, the invariant is maintained.
- InvD (9): Since $\text{mode}_i = \text{TRYING}$ holds before the statements are executed, we have $\text{grp}_i \neq \bot$ by InvD (10). Thus, execution of the statements maintains the invariant.
- InvD (10): Since we have $\text{mode}_i = \text{INCS}$ and $\text{gName} = \text{grp}_i \neq \bot$, the invariant is maintained.
- InvD (13): Before the statements are executed, $\text{gName} = \bot$ holds by precondition $\text{gSize} = 0$ and InvD (9). Thus, $\text{mode}_i \neq \text{INCS}$ 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 $tok_i \neq \bot$ by line 5.1 and $\text{home}_i = \bot$ by precondition.
- InvE (17) is maintained since $\text{home}_i = \bot$ and $\text{type}_i = \text{MAIN}$.
- InvF (19) is obviously maintained since $C_{\text{token}} = 0$ holds.
- InvJ (37) is maintained by precondition $\text{acqs}_i$.  

Lines 5.7 – 5.13. The values of $\text{tmpQ}_i$, $ts_{\text{req}}$ and $\text{reqQ}$ are changed.
- InvC (8): We claim that there is no item of $P_j$’s request in $\text{reqQ}$ if a request of $P_j$ is enqueued into $\text{reqQ}$ at line 5.11. Suppose contrary that there exists a request item $(P_j,t,g) \in \text{reqQ}$. Since $P_j$’s request is in $\text{reqQ}$, we have $\text{mode}_j = \text{TRYING}$ and $ts_j = ts_{\text{req[j]}}$ by InvC (3, 4 and 5). By InvC (7), we have $\forall (P_j,t',g') : t' \leq ts_j$. By condition of enqueue at line 5.9, each request item of $P_j$ in $\text{tmpQ}_i$ is never enqueued into $\text{reqQ}$; a contradiction. Therefore, the invariant is maintained.
- InvC (4): Request items that are not enqueued into $\text{reqQ}$ never violate the invariant. Thus, we consider each request item $(P_j,t,g)$ 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 $mode_j = \text{TRYING}$ and $C_{\text{token},*} = C_{\text{subtoken},*} = 0$. 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 $mode_j = \text{TRYING}$ holds for each $P_j$ whose request item is enqueued into reqQ.
- InvC (7) is maintained since $tmpQ$ becomes empty.
- InvD (11): Let $P_j$ be any process whose request item is enqueued at line 5.11. Then, as discussed above, $mode_j = \text{TRYING}$ holds. Let $(P_j, t, g)$ 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 $g = \text{grp}_j$ by execution of the statements, and hence the invariant is maintained.

Since all invariants are maintained just before procedure $\text{handlePendingRequests}$ is invoked, all invariants are maintained.

- **subtoken message handler:**
  - Receipt of the message and Line 6.1. The values of $type,\, mode,\, home,\, R_{\text{subtoken},k,i} \text{ and } C_{\text{subtoken},k,i}$ are changed. Just before the statements are executed, we have $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 0$ 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, $C_{\text{subtoken}}$ is decremented by one and $T_{\text{sub}}$ is incremented by one. Thus the invariant is maintained.
  - InvC (5): Before the message is received, we have $C_{\text{subtoken},*,i} > 0$. By InvC (3, 4 and 5), we have $mode_i = \text{TRYING}$, $ts_i = tsReq[i]$, no request item of $P_i$ in reqQ and $C_{\text{subtoken},*,i} = 1$. After executing the statements, we have $mode_i = \text{INCS}$ and $C_{\text{subtoken},*,i} = 0$. Thus the invariant is maintained.
  - InvC (3 and 4): Since we have $mode_i = \text{INCS}$, 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 $C_{\text{subtoken},*,i}$ 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 $R_{\text{subtoken},k,i} - S_{\text{release},i,k} = 1$ 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 $holder,\, C_{\text{acquired},k,i} \text{ and } R_{\text{acquired},k,i}$ are changed.
  - InvI (32): Before the statements are executed, $holder_i = \perp$ holds by precondition, and hence $R_{\text{acquired},j,i} - R_{\text{leave},j,i} = 0$ holds for any $P_j$ by InvI (32). After the statements are executed, we have $R_{\text{acquired},j,i} - R_{\text{leave},j,i} = 1$ 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 $\mathcal{M}_{\text{request}},\, tsReq[k]$ 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 $mode_k = \text{TRYING}$, $C_{\text{token},*,k} = C_{\text{subtoken},*,k} = 0$ and $ts_k = tsReq[k] + 1$. Thus, we have $t \geq ts_k$. 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 $mode_k = \text{TRYING}$, the invariants are trivially maintained.
  - InvC (6) is maintained since an item is removed from $\mathcal{M}_{\text{request}}$.
  - 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, $mode_k = \text{TRYING}$ holds. Let $(P_k, t, g)$ be the request item in question. By InvC (3, 4, 5 and 6), $tok.tsReq[k] < t$ implies $ts_k = t$. Thus, we have $g = \text{grp}_k$ 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 $\text{handlePendingRequests}$.
- Receipt of the message and Line 8.8. The value of $\mathcal{M}_{\text{request}}$ 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 $\mathcal{M}_{\text{request}}$ and $\text{tmpQ}$ are changed.
  - InvC (6): Since a request message is simply forwarded, the invariant is maintained.
  - InvC (7): By non-decreasing property of $ts_i$, 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 \( C_{\text{release},k,i} \), \( R_{\text{release},k,i} \), \( gSize \) and \( gName \) are changed.

- InvB (2) is maintained when \( C_{\text{token}} = 0 \) holds by InvA (1) and precondition \( tok_i \neq \perp \).
- InvF (19) is maintained when we have \( C_{\text{token}} = 0 \) by InvA (1) and precondition \( tok_i \neq \perp \).
- InvG (22): Before the message is received, \( C_{\text{release},k,i} > 0 \) holds, which implies \( S_{\text{subtoken},i,k} - R_{\text{release},k,i} > 0 \) by InvG (23). By receiving the message, we have \( S_{\text{subtoken},i,k} - R_{\text{release},k,i} \geq 0 \), and hence the invariant is maintained.
- InvG (23) is maintained because, by receiving the message, \( C_{\text{release},k,i} \) is decremented by one, and \( R_{\text{release},k,i} \) is incremented by one.
- InvG (24) is maintained by precondition \( tok_i \neq \perp \).
- InvD (9) is maintained because the value of \( gName \) becomes \( \perp \) ifff the value of \( gSize \) becomes zero.
- InvD (12): Before the message is received, we have \( gSize = 1 \) and \( C_{\text{release}} = 1 \), and by InvB (2), we have \( C_{\text{subtoken},*} = 0 \). Thus, the invariant is trivially maintained.
- InvD (13) is maintained because \( mode_i \neq \text{INCS} \) holds before the message is received by InvB (2).
- InvJ (37) is maintained by precondition \( \text{~leaving}_j \).

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 \( R_{\text{leave},k,i} \), \( S_{\text{ack},i,k} \) and \( \text{holder}_i \) are changed.
  - InvI (31) is maintained because, by receiving the message, \( R_{\text{leave},k,i} \) and \( S_{\text{ack},i,k} \) are incremented by one, respectively.
  - InvI (32): Before the message is received, we have \( R_{\text{acquired},k,i} - R_{\text{leave},k,i} = 1 \) by InvI (32) because \( \text{holder}_i = P_k \) by precondition. After receiving the message and execution of the statements, \( R_{\text{acquired},k,i} - R_{\text{leave},k,i} = 0 \) and \( \text{hold}_i = \perp \) 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 \( R_{\text{ack},k,i} \), \( \text{ack}_{i,k} \), \( B_{\text{ack},i,k} \) and \( \text{leaving}_i \) are changed.
  - InvF (20) is maintained because \( C_{\text{token}} = 0 \) holds by InvA (1) and \( tok_i \neq \perp \) by precondition.
  - InvH (27): Before the message is received, \( B_{\text{ack},i,k} = 1 \) holds by precondition \( P_k \in \text{ack}_{i,k} \). By InvH (29), we have \( S_{\text{leave},i,k} - R_{\text{ack},k,i} = 1 \). By precondition \( \text{acqs}_{i,k} = 0 \), we have \( B_{\text{acqs},i,k} = 0 \), which implies \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 0 \) by InvH (28). By receiving the message, we have \( S_{\text{leave},i,k} - R_{\text{ack},k,i} = 0 \) and hence we have \( S_{\text{acquired},i,k} - R_{\text{ack},k,i} = S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 0 \). Thus the invariant is maintained.
  - InvH (29) is maintained because we have \( S_{\text{leave},i,k} - R_{\text{ack},k,i} = B_{\text{ack},i,k} = 0 \) 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 \( \text{acqs}_{i,k} = \emptyset \).
  - InvJ (36): We have \( \text{~leaving}_i \) (line 12.3) when \( \text{ack}_{i,k} = 0 \) (line 12.2), and \( \text{leaving}_i \) by precondition is kept unchanged when \( \text{ack}_{i,k} \neq 0 \). 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 \( \text{acqs}_{i,k} \) and \( S_{\text{acquire},i,j} \) are changed.

- InvH (27): Before the message is received, \( \text{ack}_{i,k} = \{ P_k \} \) holds by condition at line 12.2. Thus, after the message is received, we have \( S_{\text{acquired},i,k} - R_{\text{ack},i,k} = 0 \) for any \( P_i \) by InvH (27).
  - Before the statements are executed, we have \( B_{\text{acqs},i,k} = 0 \) for any \( P_i \) by precondition \( \text{acqs}_{i,k} = \emptyset \). Thus, by InvH (28), \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 0 \) holds for any \( P_i \).
  - After the statements are executed, we have \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 1 \) ifff \( P_k \in \text{acqs}_{i,k} \). Thus the invariant is maintained.
  - InvH (28): After the statements are executed, we have \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = 1 \) if and only if \( P_k \in \text{acqs}_{i,k} \). This is equivalent to \( S_{\text{acquired},i,k} - S_{\text{leave},i,k} = B_{\text{ack},i,k} \) for each \( P_k \in \text{acqs}_{i,k} \), and hence the invariant is maintained.
  - InvH (30) is maintained by precondition \( tok_i \neq \perp \).
  - InvJ (35) is maintained since \( \text{ack}_{i,k} = 0 \) by line 12.2.

Line 12.9 – 12.10 with line 12.5. The values of \( \text{reqQ} \), \( \text{type}_i \), \( \text{mode}_i \), \( gSize \) and \( gName \) are changed.

- InvB (2): Before the statements are executed, we have \( \text{mode}_i = \text{TURING}, \text{ts}_i = \text{tsReq}[i] \) and \( C_{\text{token}} = C_{\text{subtoken}} = 0 \) by InvC (3, 4 and 5) because a request item of \( P_i \) is in \( \text{reqQ} \). After the statements are executed, \( \text{mode}_i = \text{INCS} \) holds and the request item is removed. Thus, the invariant is maintained.
  - InvC (3 and 4) are trivially maintained since we have \( \text{mode}_i = \text{INCS} \).
  - InvC (8) is clearly maintained by dequeue operation on \( \text{reqQ} \).
  - InvD (9): Before the statements are executed, a request item of \( P_i \) is in \( \text{reqQ} \). This implies \( \text{mode}_i = \text{TURING} \) by InvC (3, 4 and 5). By InvD (10), we have \( \text{grp}_{i,k} \neq \perp \) because \( \text{mode}_i = \text{TURING} \) holds. Thus, the assignment statements maintain the invariant.
  - InvD (10) is maintained since we have \( \text{mode}_i = \text{INCS} \) and \( \text{grp}_{i,k} \neq \perp \).
  - InvD (11) is trivially maintained since a request item of \( P_i \) is simply dequeued.
  - InvD (12): Before the statements are executed,
are satisfied on invocation of message handler or procedure. Then, releaseEvent
suming that (1) precondition of message handler or procedure, in
The event triggered at lines 4.7, 4.25, 5.6, 6.2 and 12.12.

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

Lemma 6: Suppose that requestDone event is triggered assum-
ing 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
ture until releaseEvent handler is invoked.

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

• Line 4.7 (in procedure handlePendingRequests):
  - modei = INCs and typei = MAIN by line 4.6.
  - toki ≠ ⊥ and ~leavingi by line 4.1.
  - gSize > 0 by line 4.5.
  - When procedure handlePendingRequests is invoked, gSize = 0 holds, and hence we have acksi = ∅ by InvJ (36) and ~leavingi. The value acksi is unchanged in the procedure.
  - Line 4.25 (in procedure handlePendingRequests):
    - modei = INCs and typei = MAIN by line 4.24.
    - toki ≠ ⊥ and ~leavingi by line 4.17.
    - gSize > 0 by line 4.23.
    - By the same reason as the case for line 4.7, we have acksi = ∅.

• Line 5.6 (in token message handler):
  - modei = INCs and typei = MAIN by line 5.4.
  - toki ≠ ⊥ by line 5.1.
  - ~leavingi by precondition.
  - gSize > 0 by line 5.5.
  - acksi = 0 by precondition.

• Line 6.2 (in subtoken message handler):
  - modei = INCs and typei = SUB by line 6.1.
  - toki = ⊥ by precondition.
  - ~leavingi by precondition.
  - acksi = 0 by InvJ (36).

• Line 12.12 (in ack message handler)
  - modei = INCs and typei = MAIN by line 12.9.
  - toki ≠ ⊥ by line 12.4 (and precondition).
  - ~leavingi by line 12.3.
  - gSize > 0 by line 12.10.
  - acksi = 0 by line 12.2.

Thus, when requestDone event is triggered, precondition of
releaseEvent handler is satisfied.

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

• modei = INCs is maintained because the value of modei
becomes IDLE only in releaseEvent handler (lines 3.2 and
3.11). Since releaseEvent handler is not invoked before
releaseEvent handler is invoked, the value of modei is
maintained.

• typei = MAIN ∨ typei = SUB is maintained by InvE (14).
  - When typei = MAIN, toki ≠ ⊥ holds by InvE (15).
  - When typei = MAIN, gSize > 0 holds InvB (2).
  - When typei = SUB, toki = ⊥ holds by InvE (16).
  - When typei = SUB, homei ≠ ⊥ holds by InvE (17).
  - ~leavingi holds by InvJ (33) and InvJ (37).
  - acksi = ∅ holds by ~leavingi and InvJ (36).

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

Lemma 7: Suppose that releaseDonei event is triggered assum-
ing 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
ture until requestEvent handler is invoked.

Proof: Note that releaseDonei event is triggered at line 3.13
only. Since we have typei = ⊥ and modei = IDLE at lines 3.2 and
3.11 when releaseDonei event is triggered. Thus, precondition of
requestEvent handler is satisfied when releaseDonei event is
triggered.

Next, we show that the precondition is maintained until re-
questEvent handler is invoked.

• By InvC (6), each request message in transit issued by Pj
  has timestamp value less than or equal to tsi. Thus, each
  request message issued by Pj is not enqueued into reqQ
  because tsi = tsReq[i] holds.
Since each process in its critical section eventually exits, the value of gSize is maintained until requestEvent handler is invoked. Because the value of modei is not changed, the value of grpj is maintained. By InvE (14), we have typej = ⊥. Since the value of modei is maintained, the value of typei 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
\[
\text{mode}_i = \text{mode}_j = \text{InCS} \land (\text{grp}_i \neq \text{grp}_j).
\]

By InvB (2), we have \( g\text{Size} > 0 \). This implies \( g\text{Name} \neq ⊥ \) by InvD (9). By InvD (13), we have \( g\text{Name} = \text{grp}_i = \text{grp}_j \); 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 \( \text{grp}_i \) 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**, \( tok_i \neq ⊥ \) (line 2.2): The request is always enqueued in the main-token (line 2.4).

- **Case A2**, \( tok_i = ⊥ \) and \( \text{holder}_i \neq ⊥ \) (line 2.6): \( P_i \) sends a request message directly to \( P_k \), where \( P_k = \text{holder}_i \).

We claim that \( tok_k \neq ⊥ \) holds when the request message arrives at \( P_k \). By InvI (32) and \( \text{holder}_i = P_k \), we have \( \text{Racquired},k,i \neq \text{Rleave},k,i \). Starting from this formula, by InvI (31), we have
\[
\begin{align*}
\text{Racquired},k,i &- \text{Rleave},k,i \\
= & \text{Racquired},k,i - \text{Sack},k,i \\
= & (\text{Sacquired},k,i - \text{Cacquired},k,i) - (\text{Ack},k,i + \text{Rack},k,i) \\
= & \text{Sacquired},k,i - \text{Rack},k,i - \text{Cack},k,i - \text{Cacquired},k,i \\
= & 1.
\end{align*}
\]

Because \( 0 \leq \text{Sacquired},k,i - \text{Rack},i,k \leq 1 \) by InvH (27), we must have \( \text{Sacquired},k,i - \text{Rack},i,k = 1 \) and \( \text{Cacquired},k,i = \text{Cack},i,k = 0 \).

- In case \( \text{Sacquired},k,i - \text{Sleave},k,i = 0 \): Since \( \text{Sleave},k,i - \text{Rack},i,k = 1 \) holds, \( \text{ack},k \neq 0 \) by InvH (29).

- Otherwise, i.e., in case \( \text{Sacquired},k,i - \text{Sleave},k,i = 1: \text{ack},k \neq 0 \) by InvH (28).

Thus, in any case, we have \( tok_k \neq ⊥ \) 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**, \( tok_i = ⊥ \) and \( \text{holder}_i = ⊥ \) (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 tmpQi (line 2.11) if \( P_i \in q_i \).

There are two cases to consider.

- There exists \( P_j \in (q_i \setminus \{P_i\}) \) such that \( \text{holder}_j \neq ⊥ \) holds when \( P_j \) receives the request message from \( P_i \). Then, \( P_j \) forwards the request to \( P_k \), where \( P_k = \text{holder}_j \). As claimed in case A2, \( tok_k \neq ⊥ \) holds when the request message arrives at \( P_k \).

- Otherwise, i.e., \( \text{holder}_j = ⊥ \) holds for any \( P_j \in (q_i \setminus \{P_i\}) \) 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., \( \text{Cacquired} = \text{Cleave} = 0 \), and \( \text{holder}_j = ⊥ \) holds for any \( P_j \). By InvI (32), we have \( \text{Racquired},j,i = \text{Rleave},j,i \) for any \( P_j \).

At the same time, by InvI (31) and InvH (27), eventually we have \( \text{Sacquired},j,i = \text{Sleave},j,i = \text{Rack},j,i \) for any \( P_j \). Consider when \( \text{Sacquired},j,i = \text{Sleave},j,i = \text{Rack},j,i \) becomes true for any \( P_j \). By InvH (27), the formula becomes true by receiving an ack message at \( P_j \). By receiving the message, we have \( \text{ack},j = \text{ack},i = 0 \) by InvH (28 and 29). This implies that

* acquire messages are sent (line 12.8),
* a token message is sent to other process, say \( P_k \), and acquire 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 acquire messages. By intersection property of coterie, there exists \( P_j \in q_i \cap q_k \) such that \( \text{holder}_j \neq ⊥ \) eventually becomes true. Thus, the condition assumed cannot be true forever.
<table>
<thead>
<tr>
<th>Property</th>
<th># states of model</th>
<th>time (sec)</th>
<th>memory (Mbyte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>$1.37 \times 10^6$</td>
<td>14</td>
<td>418</td>
</tr>
<tr>
<td>Liveness</td>
<td>$3.46 \times 10^6$</td>
<td>168</td>
<td>3103</td>
</tr>
</tbody>
</table>

Then, the request of $P_i$ in tmp$Q_j$ 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 $C_{\text{release}}$, 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 $C_{\text{release}}$ 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 $C_{\text{maj}}$.
- 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