Distributed Termination

Robin Sharp
robin@imm.dtu.dk

Systems Security Group
Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark.
DISTRIBUTED TERMINATION

GIVEN:
- A set of disjoint processes (no shared variables!).
- Possibility of communication between processes.

REQUIRED:
- A technique to decide when a task, which the processes cooperate to solve, is finished.

NOTE: This is not trivial in a distributed system!
Global termination when a post-condition $B(\bar{y})$ holds, where $\bar{y}$ is **GLOBAL STATE**.

$\bar{y}$ can be divided into $n$ ($\geq 2$) disjoint states $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n$.

There are $n$ predicates $B_i(\bar{y}_i), i = 1, 2, \ldots, n$ such that:

\[
(\bigwedge_{i=1}^{n} B_i(\bar{y}_i)) \Rightarrow B(\bar{y})
\]

There are $n$ processes $P_1, P_2, \ldots, P_n$ with state vectors $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n$ which can reach states where $B_i(\bar{y}_i), i = 1, 2, \ldots, n$ hold after finite time. The necessary communication to achieve this is called **BASIC COMMUNICATION**.
NOTE: Uses old-fashioned CSP (1978) notation!!

Program $P :: [P_1 \parallel P_2 \parallel \ldots \parallel P_n]$ will be a solution if termination can be forced as soon as each $P_i$ reaches a state where $B_i$ holds. This state is called a FINAL STATE.

Each $P_i$ is repetitive, of form:

$$P_i = *[g_{i1} \rightarrow S_{i1}$$
$$\parallel g_{i2} \rightarrow S_{i2}$$
$$\parallel \ldots$$
$$\parallel g_{ik_i} \rightarrow S_{ik_i}$$

where each guard $g_{ij}$ may involve Boolean conditions and BASIC COMMUNICATION

In stable state $\forall i \cdot B_i(\bar{y}_i)$, all $P_i$ are in their outer level without any ready guard. So we assume:

No processes in FINAL STATES perform BASIC COMM.
TERMINATION CONDITIONS

Processes $P_i$ establish termination by using CONTROL COMMUNICATION in addition to their BASIC COMMUNICATION.

Assume that CONTROL COMMUNICATION does not require extra COMM. CHANNELS!

For given process $P_i$ there are two termination possibilities:

**ENDOTERMINATION:** Reachability of a final state is determined by a predicate over $P_i$’s initial state $B_i(y_{0i})$.

**EXOTERMINATION:** Termination depends on each member of a TERMINATION DEPENDENCY SET:

$$T = \{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}, \quad k > 0$$

having terminated.
TERMINATION CONDITIONS

Processes \( P_i \) establish termination by using CONTROL COMMUNICATION in addition to their BASIC COMMUNICATION.

Assume that CONTROL COMMUNICATION does not require extra COMM. CHANNELS!

For given process \( P_i \) there are two termination possibilities:

**ENDOTERMINATION:** Reachability of a final state is determined by a predicate over \( P_i \)'s initial state \( B_i(y_{0i}) \).

**EXOTERMINATION:** Termination depends on each member of a TERMINATION DEPENDENCY SET:

\[
T = \{ P_{i1}, P_{i2}, \ldots, P_{ik} \}, \quad k > 0
\]

having terminated.

We can similarly define:

**ENDONONTERMINATION:** No final state can be reached.

**EXONONTERMINATION:** At least one process in the TDS does not terminate.
COMMUNICATION GRAPH, $G_P$ defined for system $P :: [P_1 \parallel P_2 \parallel \ldots \parallel P_n]$ by:
- Each process $P_i$ corresponds to a node in $G_P$.
- Each process pair $< P_i, P_j >$ for which communication can take place from $P_i$ to $P_j$, corresponds to an edge in $G_P$.

NOTE: $G_P$ can be determined syntactically from $P$. 
COMMUNICATION GRAPH, $G_P$ defined for system $P :: [P_1 \parallel P_2 \parallel \ldots \parallel P_n]$ by:

- Each process $P_i$ corresponds to a node in $G_P$.
- Each process pair $< P_i, P_j >$ for which communication can take place from $P_i$ to $P_j$, corresponds to an edge in $G_P$.

NOTE: $G_P$ can be determined syntactically from $P$.

TERMINATION GRAPH, $T_P$ defined for system $P$ by:

- Each process $P_i$ corresponds to a node in $G_P$.
- For each edge $(P_i, P_j)$ in $G_P$:
  
  \[(P_i, P_j) \in T_P \iff P_i \in TDS_j\]

  \[(P_j, P_i) \in T_P \iff P_j \in TDS_i\]

NOTE: $T_P$ reflects all termination dependencies in $P$, and may depend on initial state.

- All nodes corresponding to ENDOTERMINATING processes are sources in $T_P$ (i.e. with in-degree=0).
THEOREM: $P$ TERMINATES FOR $(\bar{y}_1, \ldots, \bar{y}_n)$ ONLY IF $T_P$ IS ACYCLIC.

PROOF: If $T_P$ contains a loop, deadlock can take place, as all nodes on loop correspond to processes which are EXONONTERMINATING for $\bar{y}$. 
THEOREM: $P$ TERMINATES FOR $(\bar{y}_1, \ldots, \bar{y}_n)$ ONLY IF $T_P$ IS ACYCLIC.

PROOF: If $T_P$ contains a loop, deadlock can take place, as all nodes on loop correspond to processes which are EXONONTERMINATING for $\bar{y}$.

What should we conclude from all this?
THEOREM: $P$ TERMINATES FOR $(\bar{y}_1, \ldots, \bar{y}_n)$ ONLY IF $T_P$ IS ACYCLIC.

PROOF: If $T_P$ contains a loop, deadlock can take place, as all nodes on loop correspond to processes which are EXONONTERMINATING for $\bar{y}$.

What should we conclude from all this?

If $P$ terminates for $\bar{y}$, a partial ordering is defined such that when all ENDOTERMINATING processes terminate, then all processes in their $TDS$s terminate, etc.

This partial ordering defines a $TERMINATION\ WAV$...
Basic idea for ensuring orderly termination:

1. Choose termination dependencies among $P_1, P_2, \ldots, P_n$ such that $T_P$ is acyclic.

2. Designate arbitrary $P_{i0}$ which collects information about whether all others are in a state where $B_i(\bar{y_i})$ holds.

3. When this happens, $P_{i0}$ must terminate, and this initiates $TERMINATION WAVE$:

   All processes in $P_{i0}$’s $TDS$ terminate, then all processes in their $TDS$s and so on.
STRATEGY (2)

Need to find a **SPANNING TREE** in undirected graph corresponding to $G_P$. Then:

1. Root process in tree initiates **CONTROL WAVE** to all its successors in the tree.
2. Wave propagates past $P_j$ if $B_j(y_j)$ holds. Passage of wave **FREEZES** all **BASIC COMMUNICATION**.
3. Each node in tree reports back to its parent whether its successors have terminated.
4. When root process receives **POSITIVE ANSWER**, it terminates and this initiates **TERMINATION WAVE**.
5. If root process receives **NEGATIVE ANSWER**, an **UNFREEZING WAVE** is propagated to allow **BASIC COMMUNICATION** again.

At least one more **BASIC COMMUNICATION** must then take place before a new **CONTROL WAVE** can be initiated.
WHAT MUST $P$ LOOK LIKE?

For this to work, $P$ must have a particular form.

In general, this will not be true of the system $P$ from the original algorithm whose termination is desired.
WHAT MUST \( P \) LOOK LIKE?

- For this to work, \( P \) must have a particular form.
- In general, this will not be true of the system \( P \) from the original algorithm whose termination is desired.
- To construct the final system, say \( \bar{P} \):
  1. Choose \( T_P^* = \) arbitrary spanning tree in undirected \( G'_P \).
  2. Modify \( P_1, P_2, \ldots, P_n \) to \( \bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n \), where:
     - \( TDS_i = \) \{parent of \( \bar{P}_i \) in \( T_P^* \)\}.
     - Root of \( T_P^* \) is ENDOTERMINATING.
     - Each \( \bar{P}_i \) is derived from \( P_i \) by adding a CONTROL SECTION, \( C_i \), consisting of further alternatives in \( P_i \).

  CONTROL SECTION deals with CONTROL WAVE, TERMINATION WAVE, UNFREEZING WAVE etc. For details, see Francez’ paper.
DISTRIBUTED PARTITION SORT

- Disjoint partitioning of $S = S_1 + S_2 + \ldots + S_n$, where cardinality of $S_i$, $|S_i| = m_i$.

- After sorting:

$$\forall i, j \ (1 \leq i < j \leq n) \cdot \ (\forall p, q \cdot (p \in S_i \land q \in S_j \Rightarrow p < q))$$
$$\land \ \forall i \ (1 \leq i \leq n) \cdot \ (|S_i| = m_i)$$

Or, equivalently:

$$\forall i, j \ (1 \leq i < n) \cdot \ (\max (S_i) < \min (S_{i+1}))$$
$$\land \ \forall i \ (1 \leq i \leq n) \cdot \ (|S_i| = m_i)$$

Or, alternatively $\land_{i=1}^{n} B_i(S_i, lin_i)$

where $\left\{
\begin{array}{l}
B_i(S_i, lin_i) \ \overset{\text{def}}{=} \ \max (S_i) \leq lin_i \land |S_i| = m_i \ (1 \leq i < n) \\
B_n \ \overset{\text{def}}{=} \ \text{true}
\end{array}\right.$

where $lin_i$ is the latest value received from $P_{i+1}$
DPS ALGORITHM

\[ P_i \; :: \; update; \text{lin} := -\infty; \]
\[ \ast [mx > \text{lin}; P_{i+1}!mx \rightarrow S_i := S_i - \{mx\}; P_{i+1}?\text{lin}; \]
\[ S_i := S_i + \{\text{lin}\}; update \]
\[ \| P_{i-1}?l \rightarrow S_i := S_i + \{l\}; update; P_{i-1}!mn; \]
\[ S_i := S_i - \{mn\}; update; P_{i-1}!mn; \]
\[ \| P_{i+1}?\text{lin} \rightarrow \text{skip} \]

\[ P_1 \; :: \; update; \text{lin} := -\infty; \]
\[ \ast [mx > \text{lin}; P_2!mx \rightarrow S_1 := S_1 - \{mx\}; P_2?\text{lin}; \]
\[ S_1 := S_1 + \{\text{lin}\}; update \]
\[ \| P_2?\text{lin} \rightarrow \text{skip} \]

\[ P_n \; :: \; update; \]
\[ \ast [P_{n-1}?l \rightarrow S_n := S_n + \{l\}; update; P_{n-1}!mn; \]
\[ S_n := S_n - \{mn\}; update; P_{n-1}!mn \]

where \( \text{update} \; \text{def} = \; mx := \max(S_i); mn := \min(S_i); \)