Describing Asynchronous Communications in Distributed Algorithms

FRIDA 2017

Philippe Quéinnec
(joint work with Florent Chevrou, Aurélie Hurault, Adam Shimi)

Université de Toulouse – IRIT ACADIE – Pardi ANR-16-CE25-0006

October 16, 2017

Description of Interactions

Description of communication in a distributed algorithm

- Explicit communication actions or shared variables?
- Messages in transit ?
- Set/multiset of messages in transit, input queues?
- Embedded in the chosen formalism?
- ...

$\rightarrow$ Common knowledge or folklore ?

Unexpected properties hidden in the communication models.

Consensus

Consensus

A set of processes each propose a value. All processes must agree on a single one.

Properties

Agreement	The correct processes all decide the same value.
Integrity	A process decides at most once.
Validity	The decision value is one of the proposed values.
Termination	A correct process eventually decides.

An algorithm in an asynchronous world

Each process compute the minimum of the proposed values: it broadcasts its value, waits for all the values and chooses the min.

Questions: broadcast? waits? messages or values?

VARIABLES *proposed*, for each process, its proposed value
hasdecided, the set of processes that have decided on a value
decision for each process, the decided value

$$\mathit{TypeInvariant} \triangleq \begin{array}{l} \land\ \mathit{proposed} \in [\mathit{PROC} \to \mathit{VALUE}] \\ \land\ \mathit{hasdecided} \in \text{SUBSET } \mathit{PROC} \\ \land\ \mathit{decision} \in [\mathit{PROC} \to \mathit{VALUE}] \end{array}$$

The correct processes all decide the same value.

$$\mathit{Agreement} \triangleq \Box(\forall\, i, j \in \mathit{hasdecided} : \mathit{decision}[i] = \mathit{decision}[j])$$

Integrity $\triangleq$ A process decides at most once.

$$\forall\, i \in \mathit{PROC}, k \in \mathit{VALUE} : \Box(i \in \mathit{hasdecided} \land (\mathit{decision}[i] = k) \Rightarrow \Box(\mathit{decision}[i] = k))$$

Validity $\triangleq$ The decision value is one of the proposed values.

$$\Box(\forall\, i \in \mathit{hasdecided} : \exists\, j \in \mathit{PROC} : \mathit{decision}[i] = \mathit{proposed}[j])$$

A correct process eventually decides.

$$\mathit{Termination} \triangleq \forall\, i \in \mathit{PROC} : \Diamond(i \in \mathit{hasdecided})$$

Not a Distributed Algorithm

VARIABLES *proposed*, for each process, its proposed value
hasdecided, the set of processes that have decided on a value
decision for each process, the decided value

$$Init \triangleq \land hasdecided = \{\}$$
$$\land proposed \in [PROC \to VALUE]$$
$$\land decision = proposed$$

$$decide(i) \triangleq$$
$$\land i \notin hasdecided$$
$$\land hasdecided' = hasdecided \cup \{i\}$$
$$\land decision' = [decision \text{ EXCEPT } ![i] = min(\{proposed[j] : j \in PROC\})]$$
$$\land \text{UNCHANGED } \langle proposed \rangle$$

$$Next \triangleq \exists\, i \in PROC : decide(i)$$

What is proved ?

Simple Model

net = bag of values, atomic broadcast, atomic reception

$$broadcast(i) \triangleq$$
$$\land\ i \notin hasbroadcasted$$
$$\land\ hasbroadcasted' = hasbroadcasted \cup \{i\}$$
$$\land\ net' = net \oplus SetToBag(\{proposed[i]\})$$
$$\land\ \text{UNCHANGED}\ \langle proposed, hasdecided, decision\rangle$$

$$decide(i) \triangleq$$
$$\land\ i \notin hasdecided$$
$$\land\ \exists\, M \in SubBag(net) :$$
$$\land\ BagCardinality(M) = N$$
$$\land\ hasdecided' = hasdecided \cup \{i\}$$
$$\land\ decision' = [decision\ \text{EXCEPT}\ ![i] = min(M)]$$
$$\land\ \text{UNCHANGED}\ \langle proposed, hasbroadcasted, net\rangle$$

$$Next \triangleq \exists\, i \in PROC : broadcast(i) \lor decide(i)$$

Messages?

net = set of messages, atomic broadcast, atomic reception

$$
\begin{aligned}
&broadcast(i) \triangleq \\
&\quad \land\ i \notin hasbroadcasted \\
&\quad \land\ hasbroadcasted' = hasbroadcasted \cup \{i\} \\
&\quad \land\ net' = net \cup \{[src \mapsto i,\ dest \mapsto j,\ value \mapsto proposed[i]] : j \in PROC\} \\
&\quad \land\ \text{UNCHANGED}\ \langle proposed,\ hasdecided,\ decision\rangle \\
\\
&decide(i) \triangleq \\
&\quad \land\ i \notin hasdecided \\
&\quad \land\ \exists\, M \in \text{SUBSET}\ net : \\
&\qquad \land\ Cardinality(M) = N \\
&\qquad \land\ \forall\, m \in M : m.dest = i \\
&\qquad \land\ hasdecided' = hasdecided \cup \{i\} \\
&\qquad \land\ decision' = [decision\ \text{EXCEPT}\ ![i] = min(\{m.value : m \in M\})] \\
&\qquad \land\ net' = net \setminus M \\
&\quad \land\ \text{UNCHANGED}\ \langle proposed,\ hasbroadcasted\rangle \\
\\
&Next \triangleq \exists\, i \in PROC : broadcast(i) \lor decide(i)
\end{aligned}
$$

Input Queues?

Input queues, atomic broadcast, atomic reception

$broadcast(i) \triangleq$
$\quad \land\ i \notin hasbroadcasted$
$\quad \land\ hasbroadcasted' = hasbroadcasted \cup \{i\}$
$\quad \land\ net' = [j \in PROC \mapsto Append(net[j], proposed[i])]$
$\quad \land\ \text{UNCHANGED}\ \langle proposed, hasdecided, decision \rangle$

$decide(i) \triangleq$
$\quad \land\ i \notin hasdecided$
$\quad \land\ \text{LET}\ M \triangleq net[i]\ \text{IN}$
$\quad\quad \land\ Len(M) = N$
$\quad\quad \land\ hasdecided' = hasdecided \cup \{i\}$
$\quad\quad \land\ decision' = [decision\ \text{EXCEPT}\ ![i] = min(M)]$
$\quad \land\ net' = [net\ \text{EXCEPT}\ ![i] = \langle\rangle]$
$\quad \land\ \text{UNCHANGED}\ \langle proposed, hasbroadcasted \rangle$

$Next \triangleq \exists\, i \in PROC : broadcast(i) \lor decide(i)$

Non atomic reception

A set of messages, atomic broadcast, non-atomic reception

$$
\begin{array}{l}
broadcast(i) \triangleq \\
\quad \land\ i \notin hasbroadcasted \\
\quad \land\ hasbroadcasted' = hasbroadcasted \cup \{i\} \\
\quad \land\ net' = net \cup \{[src \mapsto i,\ dest \mapsto j,\ value \mapsto proposed[i]] : j \in PROC\} \\
\quad \land\ \text{UNCHANGED}\ \langle proposed, hasdecided, decision, received \rangle
\end{array}
$$

$$
\begin{array}{l}
receive(i) \triangleq \\
\quad \exists\, m \in net : \\
\qquad \land\ m.dest = i \\
\qquad \land\ received' = [received\ \text{EXCEPT}\ ![i] = @ \oplus SetToBag(\{m.value\})] \\
\qquad \land\ net' = net \setminus \{m\} \\
\qquad \land\ \text{UNCHANGED}\ \langle proposed, hasdecided, decision, hasbroadcasted \rangle
\end{array}
$$

$$
\begin{array}{l}
decide(i) \triangleq \\
\quad \land\ i \notin hasdecided \\
\quad \land\ BagCardinality(received[i]) = N \\
\quad \land\ hasdecided' = hasdecided \cup \{i\} \\
\quad \land\ decision' = [decision\ \text{EXCEPT}\ ![i] = min(received[i])] \\
\quad \land\ \text{UNCHANGED}\ \langle proposed, hasbroadcasted, received, net \rangle
\end{array}
$$

Non atomic broadcast and reception

A set of messages, non-atomic broadcast, non-atomic reception

$$
\begin{array}{l}
send(i) \triangleq \\
\quad \exists\, j \in PROC : \\
\qquad \land\ j \notin sent[i] \\
\qquad \land\ sent' = [sent \ \text{EXCEPT}\ ![i] = @ \cup \{j\}] \\
\qquad \land\ net' = net \cup \{[src \mapsto i,\ dest \mapsto j,\ value \mapsto proposed[i]]\} \\
\qquad \land\ \text{UNCHANGED}\ \langle proposed,\ hasdecided,\ decision,\ received\rangle \\
\\
receive(i) \triangleq \\
\quad \exists\, m \in net : \\
\qquad \land\ m.dest = i \\
\qquad \land\ received' = [received \ \text{EXCEPT}\ ![i] = @ \oplus SetToBag(\{m.value\})] \\
\qquad \land\ net' = net \setminus \{m\} \\
\qquad \land\ \text{UNCHANGED}\ \langle proposed,\ hasdecided,\ decision,\ sent\rangle \\
\\
decide(i) \triangleq \ \ldots \\
\\
Next \triangleq \exists\, i \in PROC : send(i) \lor receive(i) \lor decide(i)
\end{array}
$$

Comparison

net	broadcast	reception	number of distinct states	
			3 sites, 2 values	4 / 2
direct access	–	–	64	256
bag of values	atomic	atomic	120	496
set of messages	atomic	atomic	120	496
input queues	atomic	atomic	216	1450
set of messages	atomic	non atomic	7568	1624096
synchronous-like	non atomic	non atomic	23814	8214536
set of messages	non atomic	non atomic	175616	?

Unexpected properties:

- input queues $\Rightarrow$ delivery order stricter than causal delivery (so-called fifo n-1)
- atomic broadcast with input queues $\Rightarrow$ total order broadcast!
- atomic multiple reception = transaction

Messages in transit

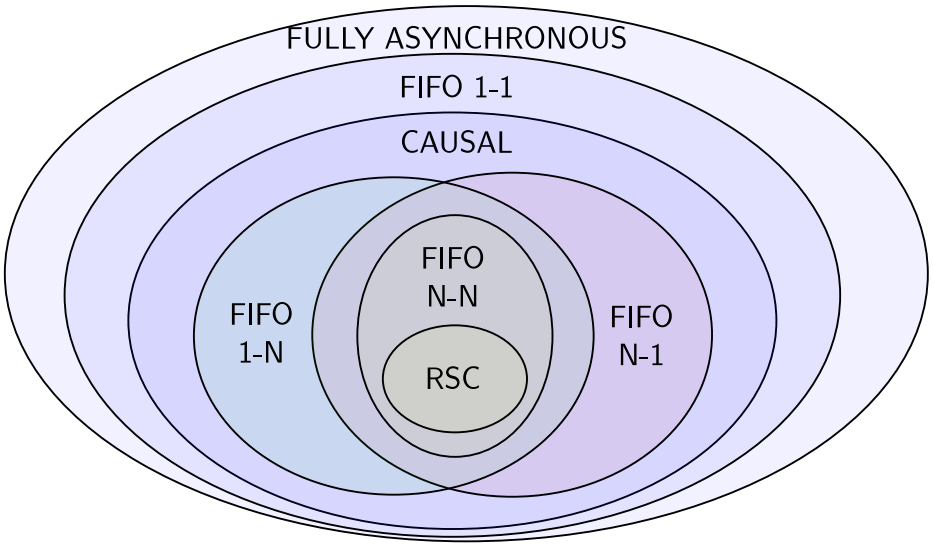
How can we model the messages in transit?

- Shared variables: no explicit message
- A set or multiset of messages (asynchronous communication)
- A sequence (a global queue)
- An array of sets (unordered input ports)
- An array of sequences (ordered input ports)
- A bi-dimensional array of sequences (fifo channels between couple of sites)
- Buffers
- ...

Is it important?

A data structure imposes a delivery order ⇒ different communication models

- Set = asynchronous
- Sequence = Fifo N-N
- Input queues = Fifo N-1
- Channels = Fifo 1-1
- 1-bounded = RSC



- Lots of different delivery orders: fifo11, causal, etc
- Hidden in the data structures
- Almost all are stricter than causal delivery (i.e. they generate less behaviors)

On the Diversity of Asynchronous Communication. F. Chevrou, A. Hurault, and P. Quéinnec. Formal Aspects of Computing, 28(5):2016.

Fairness

- On each message $\Rightarrow$ no message can stay forever in transit.
- Per receptor $\Rightarrow$ no receptor can refuse to receive some message.
  (however, by over-flooding it, one can lose message progress if no delivery order exists)
- Per sender $\Rightarrow$ a site cannot be permanently ignored?
- Per couple (sender,receptor) $\Rightarrow$ communication occurs between all sites.

Atomicity of communication actions

- Receive-and-reply
  $net' = net \setminus \{m1\} \cup \{m2\}$
  Analogous to a test-and-set or compare-and-swap

Assume we have *zero* which tests if there is a message in transit

$P1$	$P2$	$P3$	$\bot$ reachable?
$a! \cdot b!$	$a? \cdot c!$	$b? \cdot (c? + zero \cdot \bot)$	*yes*
$a! \cdot b!$	$[a?c!]$	$b? \cdot (c? + zero \cdot \bot)$	*no*

- Multiple receptions
  $net' = net \setminus \{m_1, m_2, \ldots\}$
  Looks like a transaction

$a! \cdot b! \parallel \{a, b\}? \parallel \{a, c\}?$ is different from $a! \cdot b! \parallel a? \cdot b? \parallel a? \cdot c?$
(in the second case, the third process can get a and deadlock)

Atomicity of communication actions

- Multiple emissions: $net' = net \cup \{m \mid ...\}$

Chandy-Misra algorithm for mutual exclusion

A requesting site must send a request message to all the sites to which it has previously given its permission.
Pitfall: a site must not handle a request message while it is itself sending request messages (or it must be careful).

- Broadcast = multiple emissions?

Atomicity of multiple emissions + queues (be it per site or per channel) = total order broadcast : if two broadcasts concurrently occur, the messages to the common destinations are in the same order everywhere.

Latency

- Is a message immediately available, in the transition which follows the emission? Or do we need buffers to delay? It should be irrelevant (interleaving, non-determinism).
- Application priority on channels:

$c! \cdot a! \cdot b! \parallel b? \cdot (a? + c?)$ with $a > c$.
The a branch is mandatory without explicit latency.

- Assume Test-and-Action: $zero\_a!$ sends a message on a if there is no pending message for the peer.

	both can die?
$(\tau \cdot kill_2! + kill_1?) \parallel (\tau \cdot kill_1! + kill_2?)$	yes
$(zero\_kill_2! + kill_1?) \parallel (zero\_kill_1! + kill_2?)$ without buffers	no
$(zero\_kill_2! + kill_1?) \parallel (zero\_kill_1! + kill_2?)$ with buffers	yes

Emptiness Test / Capacity / Boundness

Emptiness test

- Is it allowed to test $net = \emptyset$?
- Is it allowed to test if there is a pending message for a process?

$\rightarrow$ Carefully (see previous examples in atomicity and latency)

Global predicates

- Example: maximal capacity (maximal number of messages in transit)
- Are global predicates meaningful? Enforceable?

1. Several Models of Interaction
2. Modelisation of the Communication
3. A Framework for the Verification of Asynchronous Communicating Systems
   - Framework Architecture
   - Communication Models
   - Correctness & Completeness
   - Conclusion

Verification of Asynchronous Communicating Systems

Modeling

- *Asynchronously* communicating peers
- Multi-senders multi-receivers channels
- Group of channels associated to different communication models

Method

- Peers and communication specified using transition systems
- Compatibility checking of LTL properties
- Modular (new models)
- Fully automatic: model checking
- TLA$^+$

Automated Verification of Asynchronous Communicating Systems with TLA$^+$. F. Chevrou, A. Hurault, and P. Quéinnec. Electronic Communications of the EASST (special issue: AVOCS'15), 2015.

Explicit hypotheses

- Several communication models with specified delivery order
- Weak fairness on communication actions
- One communication action per transition (no multiple emissions, no receive-and-reply)
- . . . but point-to-point and broadcast communication
- No latency, but no test-and-action

Communication models

A communication model is built by assembling μ-models

Two point-to-point channels, with fifo11 ordering, and at most two messages in transit on channel `Request`.

```
MODELS == {
    [name |-> "p2p",
        params |-> [chan |-> {"Request", "Token"} ]],
    [name |-> "fifo11",
        params |-> [chan |-> {"Request", "Token"} ]],
    [name |-> "message_cap",
        params |-> [chan |-> {"Request"}, bound |-> 2]]
}
```

A communication action (send/receive) on a channel is enabled if it is enabled in all concerned models.

Sending & Receiving

```
Request(i) ==
   ∧ ~ requesting[i]
   ∧ requesting' = [ requesting EXCEPT ![i] = TRUE ]
   ∧ COM!send(i, {father[i]}, "Request", i)
        (* sender, destination, channel, content *)
   ∧ ...


ReceiveToken(i) ==
     ∧ ∃ j ∈ Sites : COM!receive(j, i, "Token", 0)
                  (* sender, receiver, channel, content *)
     ∧ token' = [ token EXCEPT ![i] = TRUE ]
     ∧ ...
```

Correctness & Completeness

Valid Execution

An execution respects e.g. the causal delivery order iff

$$\forall m_1, m_2 : peer(d(m_1)) = peer(d(m_2)) \wedge e(m_1) \prec e(m_2) \Rightarrow d(m_1) \prec d(m_2)$$

where $e(m)$ = emission event of m

$r(m)$ = delivery event of m

$\prec$ = causal order of events

Correctness

Any execution generated by the framework is valid.

Mechanized proofs, by refinement.

Completeness

All the valid executions are generated by the framework.

(with restrictions on the form of the system)

Manual proofs.

Advantages/Limitations

Advantages

- No re-re-re-coding of the communication actions.
- Easy reconfiguration, easy tests.
- No hidden properties.
- Experimenting plateform available online (`http://vacs.enseeiht.fr`).

Limitations

- Generic action vs ad-hoc implementation:
  $\exists i, j \in \mathit{Site} : \exists d \in \mathit{Data} : \mathit{COM}!\mathit{receive}(i, j, "\mathit{chan}", d) \wedge \ldots$
  $\text{LET } m \triangleq \mathit{Head}(\mathit{chan}[i][j]) \text{ IN} \ldots$
  $\rightarrow$ same number of states/transitions but slower
- Broadcast is still experimental
- Failures are missing (should they be in a communication model?)

Conclusion

In a communication model, beware of

- Hidden properties
- Incomplete model
- Model too close to an implementation...
- ... model too abstract / unclear w.r.t. reality

Need for

- Generic models (of computation, of communication),
- with modularity and parameterization,
- with known properties (and no hidden ones),
- with mechanization.