Robustness against stronger semantics

1 Message-passing systems

Let us fix a set $\mathbb{I}$ of process ids and a set $\mathbb{P}$ of message payloads . A *message* is a triple $m = (i, j, p)$ where $i \in \mathbb{I}$ denotes the source, $j \in \mathbb{I}$ the destination, and $p \in \mathbb{P}$ the payload. The set of messages is denoted by $\mathbb{M}$.

We also fix a set $\mathbb{S}$ of *send* actions, a set $\mathbb{R}$ of *receive* actions, and a set $\mathbb{A}$ of *internal* actions, where

$$\mathbb{S} = \{send_i(m) : i \in \mathbb{I},\ m = (i, \_, \_) \in \mathbb{M}\} \text{ and } \mathbb{R} = \{rec_j(m) : j \in \mathbb{I},\ m = (\_, j, \_) \in \mathbb{M}\}.$$

Above, i and j denote the process executing the action. We assume that internal actions are also indexed by the process executing them. For an action $a \in \mathbb{S} \cup \mathbb{R} \cup \mathbb{A}$, $proc(a)$ is the process executing the action a.

A *message-passing system* $\mathcal{S}$ is described abstractly as a set of sequences of actions $e \in (\mathbb{S} \cup \mathbb{R} \cup \mathbb{A})^*$, called *executions*. We assume that for every execution $e \in \mathcal{S}$, every receive action $rec_j(m)$ with $m = (i, j, \_)$ is preceded by a send action $send_i(m)$. Also, we assume that $\mathcal{S}$ is prefix-closed.

For instance, the set of executions $\mathcal{S}$ can be generated by a parallel composition of state machines equipped with (fifo) buffers.

2 Robustness against the rendez-vous semantics

We define robustness against rendez-vous which ensures that even if the system uses buffers to store messages it provides the illusion that messages are received instantaneously, as in the rendez-vous semantics. This is analogous to the atomicity criterion for concurrent shared-memory systems, where even though transactions can interleave, every execution is "equivalent" to an execution where transactions happen atomically without interference.

2.1 Defining robustness against rendez-vous

We define a conflict relation $\prec$ on actions in $\mathbb{S} \cup \mathbb{R} \cup \mathbb{A}$ that relates every two actions of the same process and every send with the corresponding receive. Formally,

$$\begin{aligned} a \prec a' \text{ iff } & proc(a) = proc(a') \\ & \text{or } a = send_i(m) \text{ with } m = (i, j, \_), \text{ and } a' = rec_j(m) \end{aligned}$$

A permutation e' of an execution e is *conflict-preserving* when every pair a and a' of actions of e appear in the same order in e' whenever $a \prec a'$. Intuitively, by the definition of $\prec$, if a system admits an execution e, then it admits also any conflict-preserving permutation of e.

Definition 1. *An execution e is called* rendez-vous equivalent *iff there exists a conflict-preserving permutation e' of e where every send action is immediately followed by the corresponding receive.*

For instance, the following executions are rendez-vous equivalent:

$$send_{i_1}(i_1, j_1, \_)\ send_{i_2}(i_2, j_1, \_)\ rec_{j_1}(i_1, j_1, \_)\ rec_{j_1}(i_2, j_1, \_)$$
$$send_{i_1}(i_1, j_1, \_)\ send_{i_2}(i_2, j_2, \_)\ rec_{j_2}(i_2, j_2, \_)\ rec_{j_1}(i_1, j_1, \_)$$

Definition 2. *A message-passing system $\mathcal{S}$ is called* robust against rendez-vous *iff every execution $e \in \mathcal{S}$ is rendez-vous equivalent.*

2.2 Checking robustness against rendez-vous

Checking that a given execution e is rendez-vous equivalent can be done by tracking a "conflict-graph" where transactions are pairs of sends and corresponding receives. If the conflict-graph is cyclic, then the execution is not rendez-vous equivalent.

The approach above requires executing the original system with all its complexities, e.g., unbounded buffers. Deciding whether a system is robust against rendez-vous can be reduced to a reachability problem in a system that *executes under the rendez-vous semantics*, and thus doesn't use message buffers. The idea is the following:

- consider the class of *minimal* violations to robustness, i.e., executions which are not rendez-vous equivalent and every strict prefix is rendez-vous equivalent.
- starting from the original system $\mathcal{S}$, define a new system $\mathcal{S}'$ executing under the rendez-vous semantics, which simulates minimal robustness violations of $\mathcal{S}$, if any. $\mathcal{S}'$ goes to an error state whenever such a violation exists.

This last approach allows to prove that checking robustness is decidable and has a "low" complexity.

2.3 Extended rendez-vous – Joint actions

An execution where multiple processes send messages at the same time is not rendez-vous equivalent, for instance:

$$send_i(i, j, \_)\ send_j(j, i, \_)\ rec_j(i, j, \_)\ rec_i(j, i, \_).$$

This could be handled in two ways:

- as a joint action of i and j where the two processes exchange at the same time two messages, one from i to j and one from j to i, or
- extending the rendez-vous semantics to allow *bounded-size* buffers, and prove robustness against such a semantics instead of the plain rendez-vous.

Investigate the verification problems ??

3 Robustness against the FIFO (non-deferring) semantics

This notion of robustness is related to the specifics of the P language, where processes can *defer* messages. From our discussions, I remember that this is needed to implement some notion of *transaction* that concerns multiple processes executing some session of a communication protocol. Making again the analogy with atomicity, we would like to prove that the deferred semantics ensures that indeed the transactions are atomic. This can be formalized as robustness against the FIFO semantics.

3.1 Defining robustness against the FIFO semantics

Definition 3. *An execution e is called* FIFO equivalent *iff there exists a conflict-preserving permutation e' of e where for every two send actions that deliver messages to the same process, the corresponding receives are in the same order, i.e., for every two actions $send_i(i, j, p)$ and $send_{i'}(i', j, p')$ occurring in this order in e' the corresponding receive actions occur in the same order, i.e., $rec_j(i, j, p)$ before $rec_j(i', j, p')$.*

For instance, the following execution is FIFO equivalent (sends from different processes are not in conflict and they can be reordered):

$$send_i(i, j, \_)\ send_{i'}(i', j, \_)\ rec_j(i', j, \_)\ rec_j(i, j, \_)$$

3.2 Checking robustness against the FIFO semantics

Checking that a given execution e is FIFO equivalent can be done as follows:

- define a graph G where nodes correspond to actions in e and edges to conflicts between these actions w.r.t. $\prec$
- e is *not* FIFO equivalent iff there exist two messages (i, j, p) and (i', j, p') such that G contains both a path from $send_i(i, j, p)$ to $send_{i'}(i', j, p')$ and a path from $rec_j(i', j, p')$ to $rec_j(i, j, p)$.

This approach doesn't imply that the problem is decidable. Is the problem decidable in general ? Can we define reductions to reachability problems in simpler models as for robustness against rendez-vous ?