Decidability results of Communicating Finite State Machines

G. Namratha Reddy
Chennai Mathematical Institute, India
`http://www.cmi.ac.in/~namratha`

Abstract

Models of concurrent finite state processes that communicate over queues allow different reachability results based on the underlying topology. The reachability is known to be undecidable for a general topology, but when we restrict to acyclic topologies it becomes decidable. The paper provides a proof of how it can be decided by reducing the problem to checking emptiness of a certain regular language.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed Systems, Reachability, Finite state machines with FIFO queues, Network of concurrent processes

1 Introduction

The concurrent nature of distributed systems makes their verification challenging. Formal methods employ mathematical models to reason about them. Communicating finite state machines (CFSM) over queues has been a standard model for modelling concurrent non-recursive programs.

Defining correctness of a concurrent program is also non-trivial. One idea is to define it via a safety property, which states that a bad state should never be reached in the program. This translates to asking whether it is possible to reach this bad state from the initial configuration in this model. The answer to this question depends on the model we consider. It turns out that for CFSMs with queues, it is undecidable to know if a state is reachable, but when restricted to certain class of acyclic topologies it becomes decidable.

2 Model

2.1 Topology

We describe our topology using the tuple

$$\mathcal{T} = (\mathcal{P}, \mathcal{C}, r, w),$$

where

$\mathcal{P}$ is the set of processes;
$\mathcal{C}$ is the set of channels;
$r : \mathcal{C} \to \mathcal{P}$, reader function that assigns to each channel a process that can read from it;
$w : \mathcal{C} \to \mathcal{P}$, writer function that assigns to each channel a process that can write to it.

Channels

Each channel $c \in \mathcal{C}$ has a message alphabet Γ_c. We assume channel alphabets are disjoint. If $w \in \Gamma_c^*$, we mean the first letter of w is at the head of the channel c.

Operations on the channel:

- $c!m$ is a send operation. It appends the message m to the end of channel c.
- $c?m$ is a read operation. It receives the message m from the head of channel c (enabled only if m is at the head of channel c).

Processes

Each process $p \in \mathcal{P}$ is modelled as a finite state transtion system

$$TS_p = (Q_p, \Delta_p, s_p),$$

where

Q_p is a finite set of states;

$\Delta_p \subseteq Q_p \times O_p \times Q_p$ is a finite set of transitions, where,

$$O_p = \{c?m \mid r(c) = p, m \in \Gamma_c\} \cup \{c!m \mid w(c) = p, m \in \Gamma_c\};$$

s_p is the initial state.

2.2 Configuration Graph

A configuration (α, β) of the system is a pair describing the states of all processes (control state) and the contents of all channels (channel state).

$$\alpha \in \prod_{i \in \mathcal{P}} Q_i,\ \beta \in \prod_{c \in \mathcal{C}} \Gamma_c^*$$

For any $p \in \mathcal{P}$, $\alpha(p)$ gives the state of process p, and for any $c \in \mathcal{C}$, $\beta(c)$ gives the content of channel c. We define $\alpha[p \leftarrow q]$ to be the control state α' such that $\alpha'(p) = q$, and $\alpha'(p') = \alpha(p')$ if $p' \neq p$. Similarly, we define $\beta[c \leftarrow w] = \beta'$ such that $\beta'(c) = w$, and $\beta'(c') = \beta(c')$ if $c' \neq c$

Vertices in the configuration graph $(G = (V, E))$ are configurations, and the edges are labelled with channel operations. The edges need to satisfy the following conditions:
Let $(\alpha, \beta), (\alpha', \beta') \in V, c \in \mathcal{C}$, then

- $(\alpha, \beta) \xrightarrow{c!m} (\alpha', \beta')$ if
  - $\alpha' = \alpha[w(c) \leftarrow \alpha'(w(c))]$
  - $(\alpha(w(c)), c!m, \alpha'(w(c))) \in \Delta_{w(c)}$
  - $\beta' = \beta[c \leftarrow \beta(c).m]$
- $(\alpha, \beta) \xrightarrow{c?m} (\alpha', \beta')$ if
  - $\alpha' = \alpha[r(c) \leftarrow \alpha'(r(c))]$
  - $(\alpha(r(c)), c?m, \alpha'(r(c))) \in \Delta_{r(c)}$
  - $\beta = \beta'[c \leftarrow m.\beta'(c)]$

A run ρ is a path in this graph. $c_1 \xrightarrow[G]{*} c_2$ represents a run starting in c_1 and ending in c_2. $\rho \downarrow_p$ is the projection of ρ to process p, i.e. only transitions of process p are considered. Similarly, we can define projection to a set of processes.

3 The Reachability Problem

A target control state α_t is said to be reachable if there is a run $c_0 \xrightarrow{*} c_t$, where, $c_0 = (\alpha_0, \beta_0)$, $\alpha_0(p) = s_p\ \forall p \in \mathcal{P}$, and $\beta_0(c) = \epsilon\ \forall c \in \mathcal{C}$; $c_t = (\alpha_t, \beta)$ for some β

The reachability problem asks whether a given target control state is reachable. We know that the problem is undecidable in general, because if there is a loop in the topology then we can simulate a queue machine and state reachability is undecidable for queue machines.

So we ask the question of whether it is decidable for acyclic topologies. What we mean by acyclic topologies is consider the network topology and ignore the direction of the edges, we get an undirected graph and this graph should have no cycles.

Tree topology

Given an acyclic topology the reachability problem can be reduced to one in which the target state is reached only if the queue is empty [1]. Using this reduction our reachabilty problem reduces to reaching a channel state β_t with every channel being empty i.e. we take every channel content to be ϵ in the target state.

We further reduce our topology to another isomorphic topology that looks like a tree, where every process has one incoming edge (except the root) i.e every process can read from one channel but write to multiple channels [1]. Henceforth we work with a tree topology. Although the root node doesn't read from any channel, we add a dummy channel which is initialized to ϵ for consistency with other nodes. This way every node reads from some channel and we define a function $\iota(p)$, which gives the channel that process p reads from. We define $G_p = (V_p, E_p)$ to be the induced subgraph of the configuration graph G where V_p are the nodes reachable from p in the tree.

Languages

Given the target control state α_t to be reached, we define for all $p \in \mathcal{P}$ the language $L_p = L(A_p)$. We get the finite automation A_p by taking the transition system TS_p and replacing the opertion in the transition with just the message and having final state as $\alpha_t(p)$. $A_p = (Q_p, \Sigma_p, \delta_p, s_p, \alpha_t(p))$, where, $\Sigma_p = \bigcup_{c \in \mathcal{C}} \{\Gamma_c : r(c) = p \text{ or } w(c) = p\}$, $\delta_p = \{(q, m, q') : (q, c?m, q') \in \Delta_p \text{ or } (q, c!m, q') \in \Delta_p\}$. $\pi_c(w)$ is the word we get by deleting all letters not in Γ_c. This can be extended to languages as well i.e. $\pi_c(L) = \{\pi_c(w) : w \in L\}$.

We define for all $p \in \mathcal{P}$, L^e_p as follows :

- if p is a leaf node, $L^e_p = L_p \cap \Gamma^*_{\iota(p)}$,
- if p is a non-leaf node, and children of p is given by the set $k(p) = \{k_1, k_2, .., k_l\}$, $L^e_p = L_p \cap \text{shuffle}(\{\pi_{\iota(i)}(L^e_i) : i \in k(p)\} \cup \{\Gamma^*_{\iota(p)}\})$

If there is run ρ in G then by projecting it to p we get $\rho\downarrow_p$, we can take this run and erase the channel name to get a corresponding run in A_p. Conversely, if we have a run in A_p labelled with w and start in a configuration where channel $\iota(p)$ has the word $\pi_{\iota(p)}(w)$ then we can get the corresponding run in G.

▶ **Lemma 1.** *Let p be the root of the tree, then*

$$w \in L^e_p \implies (\alpha_0, \beta_0[\iota(p) \leftarrow \pi_{\iota(p)}(w)]) \xrightarrow[G_p]{*} (\alpha'_t, \beta_t),$$

where $\alpha'_t(p) = \alpha_t(p)$ for $p \in V_p$ and $\alpha'_t(p) = \alpha_0(p)$ for $p \notin V_p$

Proof. By induction on the no. of nodes in the tree graph G_p.

Base case: No. of nodes is 1. Let this node be p. $w \in L^e_p \implies w \in L_p$, so there is a run $s_p \xrightarrow[A_p]{w} \alpha_t(p)$. Since p is a leaf node it doesn't write to any channel, $\pi_{\iota(p)}(w) = w$. Therefore, in G_p we get the run $(\langle s_p \rangle, \langle \pi_{\iota(p)}(w) \rangle) \xrightarrow[G_p]{*} (\langle \alpha_t(p) \rangle, \langle \epsilon \rangle)$

Induction: Let p be a non-leaf node and children of p be $k(p) = \{k_1, k_2, .., k_l\}$. We have $w \in L_p \cap \text{shuffle}(\{\pi_{\iota(i)}(L^e_i) : i \in k(p)\} \cup \{\Gamma^*_{\iota(p)}\}) \implies w \in shuffle(w_1, .., w_l, \pi_{\iota(p)}(w))$ where $w_i \in \pi_{\iota(i)}(L^e_i)$. By induction hypothesis we have a run ρ_i : $(\alpha_0, \beta_0[\iota(i) \leftarrow w_i]) \xrightarrow[G_i]{*} (\alpha'_t, \beta_t)$ for each $i \in k(p)$. Since $w \in L_p$ we have the run ρ: $s_p \xrightarrow[A_p]{*} \alpha_t(p)$. Using ρ and the runs for each i, ρ_i, we can construct a run in G_p as follows: $(\alpha_0, \beta_0[p \leftarrow \pi_{\iota(p)}(w)]) \xrightarrow[\text{p moves}]{*} (\alpha_0[p \leftarrow \alpha_t(p)], \beta_0[\{i \leftarrow w_i : i \in k(p)\}]) \xrightarrow[k_1 \text{ moves}]{*} \cdots \xrightarrow[k_l \text{ moves}]{*} (\alpha'_t, \beta_t)$ ◀

▶ **Lemma 2.** *Let p be the root of the tree, then*

$$(\alpha_0, \beta_0[\iota(p) \leftarrow \omega]) \xrightarrow[G_p]{*} (\alpha'_t, \beta_t) \implies \exists w \in L^e_p \text{ such that } \pi_{\iota(p)}(w) = \omega,$$

where $\alpha'_t(p) = \alpha_t(p)$ for $p \in V_p$ and $\alpha'_t(p) = \alpha_0(p)$ for $p \notin V_p$

Proof. Induction on the no. of nodes in the tree.

Base case: No. of nodes is 1. Let this node be p. We have $(\langle s_p \rangle, \langle \omega \rangle) \xrightarrow[G_p]{*} (\langle \alpha_t(p) \rangle, \langle \epsilon \rangle)$ implies we have the run $s_p \xrightarrow[A_p]{\omega} \alpha_t(p) \implies \omega \in L^e_p$ and we can take w as ω itself since $\pi_{\iota(p)}(\omega) = \omega$.

Induction: Let p be a non-leaf node and children of p is given by the set $k(p) = \{k_1, k_2, .., k_l\}$. We have the run $\rho : (\alpha_0, \beta_0[\iota(p) \leftarrow \omega]) \xrightarrow[G_p]{*} (\alpha'_t, \beta_t)$. We can construct another run ρ' in which all the p transitions are taken first and no other process moves until p finishes. Once p finishes let channel state be β' where $\beta'(\iota(k_i)) = \omega_i$ and the rest of the channels have ϵ.

Take the corresponding run in A_p and let that be $s_p \xrightarrow[A_p]{w'} \alpha_t(p)$

We can extract the following run from ρ' by taking only transitions of V_{k_i}

$(\alpha, \beta_0[\iota(k_i) \leftarrow \omega_i]) \xrightarrow[\text{only nodes in } V_{k_i} \text{ move}]{} (\alpha'_t, \beta_t)$. By induction hypothesis we have some word $w_i \in L^e_{k_i}$ and $\omega_i = \pi_{\iota(k_i)}(w_i) \implies \omega_i \in \pi_{\iota(k_i)} L^e_{\iota(k_i)}$

We have $w' \in shuffle(\omega_1, ..., \omega_l, \omega) \implies w' \in L^e_p$ and $\pi_{\iota(p)}(w') = \omega$

◀

▶ **Theorem 3.** *The Reachability problem is decidable for CFSMs with queues over acyclic toplogies.*

Proof. We use the reductions stated above to get a tree topology. Let p be the root of the tree. From Lemma 1 we know that if $w \in L^e_p$ then there is a run $(\alpha_0, \beta_0[\iota(p) \leftarrow \pi_{\iota(p)}(w)]) \xrightarrow[G_p]{*} (\alpha'_t, \beta_t)$. Since the root node doesn't read from a channel we can keep it empty and take the same transitions to get $\rho : (\alpha_0, \beta_0) \xrightarrow[G_p]{*} (\alpha_t, \beta_t)$. Conversely, if we have such a run ρ, using Lemma 2 and substituting ϵ for ω we get a $w \in L^e_p$. Therefore, we can reduce reachability to emptiness of the language L^e_p which is regular and hence decidable.

◀

References

1 La Torre, S., Madhusudan, P., Parlato, G. (2008). Context-Bounded Analysis of Concurrent Queue Systems. In: Ramakrishnan, C.R., Rehof, J. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008. Lecture Notes in Computer Science, vol 4963. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/978-3-540-78800-3_21`