Contents

6 Simulating Synchronous Systems 1

6.1 Overview 2

6.2 Synchronous Message Passing 2

6.2.1 Example: Simultaneous Restart 5

Bibliography 15

6 Simulating Synchronous Systems

Chapter Contents

6.1	Overview	2
6.2	Synchronous Message Passing	2

Learning Goals

The main goal of this chapter is to introduce a simple yet highly idealized model of distributed computing: the synchronous model. We design and analyze algorithms for fundamental problems in the synchronous model, and discuss the model's power and limitations. We then show how one can simulate the synchronous model in an asynchronous system.

6.1 Overview

Perhaps the simplest theoretical distributed models are synchronous models. Here, we describe a *synchronous message passing* model, SMP, that abstracts away many computational details that must be considered in practice: local computational costs, bandwidth restrictions, asynchrony, and faults. Nonetheless, SMP is instructive for designing and reasoning about distributed protocols. In SMP, all nodes progress in lock step, and all messages sent in each round are received before the next round begins. Thus, all nodes progress in a coordinated way. The only computational restriction in SMP is *locality*: each node maintains a private state, and can only communicate with neighboring nodes to learn about their states.

Since the synchronous model assumes so few restrictions on its computational power, lower bounds for this model transfer *a fortiori* to weaker, more realistic computational models. Yet synchronous protocols can often be adapted to more realistic models as well—the challenge is to understand how.

We next introduce a *fully asychronous message passing* model, AMP, that makes no assumptions whatsoever about synchrony between nodes. In AMP, all computations occur in response to *events* without any synchrony between the timing of events; local computations and message delivery may take arbitrarily long to complete.

Despite the seemingly pessimistic assumption of total asynchrony in the AMP model, we present a general technique, called an *α-synchronizer*, that allows one to simulate any algorithm for the SMP model in the seemingly much weaker AMP model. Specifically, we will prove the following theorem.

Theorem ??. *Given any algorithm $\mathcal{A}$ in SMP with running time T, there is an algorithm $\mathcal{B}$ that simulates $\mathcal{A}$ in AMP with a running time of T.*

In general, it is often fruitful to first understand how to *simulate* a more powerful model before attempting to solve a problem in the more restrictive model directly. The α-synchronizer technique we describe gives a general procedure for simulating executions of SMP in AMP.

6.2 Synchronous Message Passing

We describe a distributed system by a simple, connected graph $G = (V, E)$ (see Appendix **??**), where V is the set of $n := |V|$ nodes (our computational entities, e.g., clock domains on a chip, processors in a multi-core machine, or computers in a network). Nodes v and w can directly communicate if and only if there is an edge $\{v, w\} \in E$. We model each node in the network as a finite state

machine, where the state of a node includes the contents of messages received from other nodes .

Definition 6.1. *[The SMP model] The network consists of as a simple graph* $G = (V, E)$ *of n nodes. Each node* $v \in V$ *has a unique* port number *assigned to each neighbor from the range* $1, 2, \ldots, \deg(v)$ *(where* $\deg(v)$ *denotes the* degree*—i.e., number of neighbors—of* v*). Nodes are modeled by finite state machines. An algorithm is executed in* synchronous rounds*, where in each round, each node performs the following steps:*

1. *update its local state according to its transition function applied to its current state at the beginning of the round and messages received in previous rounds;*
2. *send messages to its neighbors in the graph* G*;*
3. *receive messages (that were sent by neighbors in step 2 of the current round).*

In addition, nodes may determine a (local) output and terminate at the end of a round.

In SMP, we think of the state of a node v at the beginning of round as consisting of both the state of v's "internal" finite state machine, as well as the state (i.e., value) of messages received in the previous round. Thus, for a node v with two neighbors, the state of v is a triple of values $(\textsc{state}_v, \textsc{rec}_1, \textsc{rec}_2)$. Here $\textsc{rec}_1$ and $\textsc{rec}_2$ are the messages recieved from v's neighbors with port numbers 1 and 2 (respectively) in the previous round. The transition function for v determines both how v updates $\textsc{state}_v$ and the messages, $\textsc{snd}_1$ and $\textsc{snd}_2$, v sends to its neighbors in Step 2 of the current round.

Example 6.2 (Distributed Program Counter)**.** *In a* distributed program counter*, each node v maintains a variable* $\textsc{count}_v \in \mathbb{N}$ *satisfying the following conditions:*

1. monotonicity*: for all times* t, t' *with* $t \leq t'$*, we have* $\textsc{count}_v(t) \leq \textsc{count}_v(t')$*;*
2. increment*: if* $\textsc{count}_v(t) > 0$ *at time t, then there exists some time* $t'(< t)$ *for which* $\textsc{count}_v(t') = \textsc{count}_v(t) - 1$*;*
3. waiting for neighbors*: for all times t and all neighbors w of v* $\textsc{count}_v(t) \leq \textsc{count}_w(t) + 1$.

Conditions 1 and 2 simply require that $\textsc{count}_v$ *attains values* $0, 1, 2, \ldots$ *sequentially (though the count is allowed to stop at any time). Condition 3 is important for distributed coordination: it stipulates that v cannot increase* $\textsc{count}_v$ *until all of its neighbors' counters have also reached the value* $\textsc{count}_v$.

MITPress NewMath.cls LaTeX Book Style Size: 6x9 December 18, 2020 6:02pm

Implementing a distributed program counting in SMP is extremely simple, as each node can set COUNT_v *to the current round:*

1: $\text{COUNT} := 0$;
2: ***for all*** *rounds r* ***do***
3: $\quad \text{COUNT} := \text{COUNT} + 1$
4: ***end for***

The pseudo-code above clearly satisfies the monotonicity and increment conditions. The wait-up condition is satisfied due to the specification of the SMP model: all nodes are assumed to transition from one round to the next at precisely the same time. Thus, in the SMP model, the pseudo-code above gives the stronger guarantee that for all times t *and all nodes* $v, w \in W$*, we have* $\text{COUNT}_v(t) = \text{COUNT}_w(t)$*. Moreover, this perfect synchrony is achieved without any communication! Achieving such perfect synchronization is, of course, impossible in practice. In the following section, we will show how to implement a distributed program counter in a fully asynchronous model. Understanding how to do so will give the key insight into simulating* any *synchronous algorithm in the asynchronous model.*

E6.1 Consider a relaxation of a distributed program counter where item 2 is replaced with:

2' *weak increment*: if $\text{COUNT}_v(t) > \text{COUNT}_v(0)$ at time t, then there exists some time $t'(< t)$ for which $\text{COUNT}_v(t') = \text{COUNT}_v(t) - 1$.

That is, COUNT_v may be initialized to an arbitrary count, but the other properties of a distributed program counter must be maintained. Write pseudo-code for an algorithm that solves this relaxed program counter problem, and prove its correctness (i.e., that it satisfies the three conditions of a distributed program counter with the modified condition 2).

In SMP, an important measure of the quality of an algorithm is the number of rounds the algorithm requires in order to perform some task. (Other important considerations might be the message size, number of states in the finite state machine, or complexity of local computations performed. For now, we focus on the number of rounds.) The notion of *time complexity* formalizes a measure of complexity for performing tasks in SMP.

Definition 6.3. *Let* $\mathcal{A}$ *be an algorithm in the SMP model, and let* $\mathcal{T}$ *denote a task performed by* $\mathcal{A}$*. The* time complexity *of* $\mathcal{A}$ *completing* $\mathcal{T}$ *is defined to be*

$$\sup\left\{r_1 - r_0 \,\middle|\, \mathcal{T} \text{ initiated in round } r_0,\ \text{completed in round } r_1\right\}, \tag{6.1}$$

where, for a fixed network G*, the supremum is taken over all possible states of the individual nodes in round* r_0*. That is, the time complexity of* $\mathcal{A}$ *completing*

$\mathcal{T}$ on G is the maximum number of rounds that elapse between when $\mathcal{T}$ is initiated until it is completed.

Remark 6.4. *We defined time complexity as the worst-case completion time for a fixed network G. However, it is often valuable to understand the performance of algorithms on families of graphs, and how the time complexity depends on different graph parameters. For example, one would often like to know how the performance of an algorithm scales with the size—i.e., number of nodes—of the network. One can then interpret the* sup *in (6.1) to be taken over all inputs and all networks with n nodes, so that the time complexity becomes a function n. Thus the statement "algorithm $\mathcal{A}$ has time complexity $O(n)$" can be interprated as saying that the worst case time complexity of $\mathcal{A}$ over all graphs with n nodes scales (at most) linearly in the size of the network. Similarly, one can analyze how the time complexity of $\mathcal{A}$ depends on, say, the network diameter, D, or maximum degree of a node, Δ (or even a combination of parameters) by interpretating the* sup *in (6.1) as taken over all networks with fixed values of these parameters, and treating the complexity as a function of the parameters of interest.*

Remark 6.5. *Definition 6.3 allows for the possibility that a task is completed in the same round that it is initiated. Thus, the time complexity for a task can be* 0*.*

6.2.1 Example: Simultaneous Restart

Here, we introduce a fundamental problem in hardware design that we call *simultaneous restart*.[1] In the simultaneous restart problem, a node—which we call the *leader*—enters a state READY (due to some external signal, say) at the beginning of some arbitrary round r_0. The goal is for all nodes to simultaneously enter a state START at some later round $r_1 > r_0$.

For the present discussion, the network consists of a path of n nodes, which we will refer to as $v_1, v_2, \ldots, v_n$, ordered from left to right. For each v_i with $2 \leq i \leq n-1$, v_{i-1} is v_i's *left neighbor*, with associated port number 1, and v_{i+1} is v_i's *right neighbor* with port number 2. We assume that the left-most node v_1 is the leader.

6.2.1.1 A Solution with Counters We first present a conceptually simple counter-based solution to the simultaneous restart problem, assuming that the

[1] Historically, this problem was called the "firing squad" problem, but we opt for the less morbid and more descriptive "simultaneous restart." See the chapter notes for a discussion of the classical literature on the subject.

network size, n, is known to the leader, and that messages may be as large as $\log n$ bits. While the solution we present is conceptually straightforward, the resources required to implement the counter-based solution in hardware may be unsuitably large. We present a solution where the state space and message sizes are constant (indpendent of n) in the sequel.

In our counter-based solution, the restart is initiated when the leader, v_1, receives a signal "RESTART." At that point, the leader starts a countdown of $n-1$ rounds until it enters the START state. It also decrements its counter to $n-2$, and sends this value to its right neighbor. Upon receiving a counter value, each v_i decrements the received value, sets its own counter to the decremented value, and sends this value to its right neighbor. Each round thereafter, v_i decrements its counter until it reaches 0. Once v_i's counter reaches 0, v_i transitions to START. We give pseudo-code in Algorithm 1.

Algorithm 1 RestartCounter.

```
1:  COUNT := ∞
2:  for all rounds r do
3:      if LEADER and receive RESTART then
4:          COUNT := n − 1                          ▷ set local counter
5:          SND_2 := n − 1         ▷ send counter value to right neighbor
6:      end if
7:      if REC_1 ≠ ∅ then            ▷ receive a count from left neighbor
8:          COUNT := REC_1 − 1
9:          SND_2 := COUNT      ▷ send decremented count to right neighbor
10:     else if COUNT < ∞ then
11:         COUNT := COUNT − 1
12:     end if
13:     if COUNT = 0 then
14:         STATE := START
15:     end if
16: end for
```

Proposition 6.6. *The algorithm* RestartCounter *(Algorithm 1) solves the simultaneous restart problem in $n-1$ rounds.*

Proof. Let r_0 denote the round in which the leader receives the RESTART message. For $i = 0, 1, 2, \ldots, n-1$, we claim that the following properties hold at the end of round $r_0 + i$:

1. v_i sets COUNT$_i$ to $n - i - 1$ and sends this value to v_{i+1},

2. all v_j with $j \leq i$ satisfy $\textsc{count}_j = n - i - 1$.

We argue by induction on i. The base case $i = 0$ is immediate from Lines 4 and 5. For the inductive step, assume items 1 and 2 above hold for round $r_0 + i$ with $i > 0$. By the inductive hypothesis, node v_{i+1} receives the message "$n - i - 1$" from v_i in round $r_0 + i + 1$. In Lines 8 and 9, v_{i+1} sets COUNT to $n - i - 2 = n - (i + 1) - 1$ and sends this value to v_{i+2}. Thus v_i satisfies items 1 and 2 of the claim in round $r_0 + i + 1$. For each $j < i + 1$ the inductive hypothesis implies that $\textsc{count}_j = n - i - 1$ at the beginning of round $i + 1$. This value is decremented in Line 11, so that $\textsc{count}_j = n - (i + 1) - 1$ at the end of round $r_0 + i + 1$. Therefore v_j satisfies item 2 for all $j < i + 1$ at the end of round $i + 1$ as well. This concludes the inductive argument.

By item 2 above, all nodes transition to state START in round $r_1 = r_0 + i - 1$ (Line 14), and no node transitions to START in any round $r_0 + i$ with $0 \leq i < n - 1$. Therefore, RestartCounter solves the simultaneous restart problem, as desired. □

E6.2 Suppose that you can customize each state machine, i.e., the state machine of node v_i may depend on i. Modify Algorithm 1 to work with "empty" messages, i.e., the only communicated bit of information is whether a message was sent or not. Prove that the modified algorithm is correct!

E6.3 Consider the case where the network $G = (V, E)$ is an arbitrary connected graph (i.e., not a path as described above). Suppose a fixed node v_1 is the leader, and that the radius R (i.e., the maximum distance from v_1 to any other node in the network) is known to v_1. Modify Algorithm 1 to solve the simultaneous restart problem on G, and prove the correctness of your algorithm. What is the time complexity of the modified algorithm?

6.2.1.2 A Finite State Solution Here we describe a solution to the simultaneous restart problem that requires no previous knowledge of the size of the network, and uses only a constant number of node states, independently of the network size. The basic idea of the solution is recursive: each "phase" divides the network in half, and then recursively solves the firing squad problem on the two (equal sized) halves of the network. More specifically, the first phase of the algorithm finds the midpoint(s) of the network, which serve as leader(s) for the second phase. In each phase, the size of each "active" component of the network gets cut in half, as does the amount of time to complete the phase. At the end of the last phase, all nodes are in the READY state, at which point they simultaneously transition to START. Before the final phase, each node will have at least one neighbor who is not in the READY state, so no one transitions to START before the final phase.

MITPress NewMath.cls LaTeX Book Style Size: 6x9 December 18, 2020 6:02pm

Finding the center node of an "active" component—i.e., a set of nodes between consecutive READY nodes—is at the heart of the algorithm. The idea is simple: initially the leader sends two messages to its right neighbor: a "fast" message and a "slow" message. The fast message progresses one hop per round, until it reaches the right end of the network. At this point, the fast message bounces back from right to left, still moving at one hop per time step. Meanwhile, the slow message moves to the right at one hop every 3 rounds. Thus, after $\lfloor 3n/2 \rfloor$ rounds, the two messages meet at the middle node or adjacent middle nodes (depending on the parity of n). When the two messages meet, the node(s) where they meet enter(s) the state READY, and initiate(s) fast/slow messages both to the right and left. This concludes the first phase. The subsequent phases progress analogously with the newly READY nodes serving as leaders for the next phase.

In the k-th phase, the size of each active component is at most $n/2^k$, and all active components have the same size. In the final phase, all nodes are in the READY state, and they simultaneously transition to START in the following round. As for the run-time of the procedure, the duration of a phase is $3/2$ times the size of active components during that phase. Thus, the total run-time is

$$\frac{3}{2}n + \frac{3}{2} \cdot \frac{n}{2} + \frac{3}{2} \cdot \frac{n}{4} + \cdots < 3n \sum_{k=1}^{\infty} \frac{1}{2^k} = 3n.$$

In order to describe and analyze the algorithm sketched above formally, we describe explicit states and transition functions. The state space consists of 16 states, with the following symantics:

R the ready state (a.k.a., READY),

S the start state (a.k.a. START),

Q the quiescent state (i.e., the nominal state before the procedure begins),

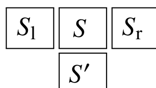
Q' the quiescent state of the right-most node,

$\overrightarrow{F}_i, \overleftarrow{F}_i$ fast message states moving right and left, respectively, for $i = 1, 2$

$\overrightarrow{S}_i, \overleftarrow{S}_i$ slow message states moving right and left, respectively, for $i = 1, 2, 3,$

$\overrightarrow{W}, \overleftarrow{W}$ waiting states indicating the direction of the previous "message" state.

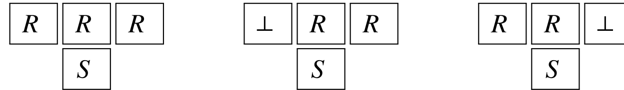
We assume that each node sends its internal state to its neighbors in each round. Thus, the transition function is specified by determining the next local state of each node from the local states of itself and its neighbors at the end of the previous round. We use the notation



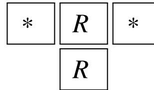
to indicate that a node in state S whose left and right neighbors are in states S_l and S_r, respectively, transititions to state S'.

We assume that for all rounds $r < r_0$, every node is in state Q except the right-most nodes, which is in state Q'. In round r_0, the leader v_0 transitions to the ready state R. We now describe the transition function, along with the semantics. We use the "wildcard" $*$ to indicate an arbitrary state. We use $\perp$ to denote that a neighbor is nonexistent (i.e., for the left neighbor of v_1 and the right neighbor of v_n. In order to simplify the presentation, the following rules are written in order of precedence. That is, if two rules apply (due to wildcards), then the first rule appearing on the list below is applied.

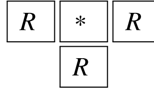
1. Start if everyone is ready



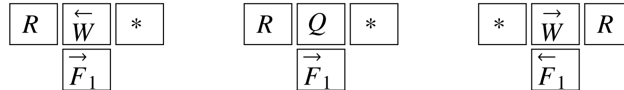
2. Ready stays ready



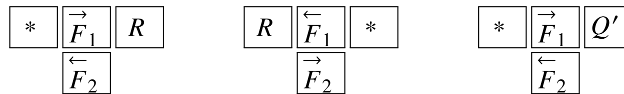
3. Ready if both neighbors are



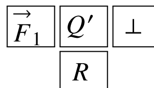
4. Ready states initiate a fast message in the correct direction



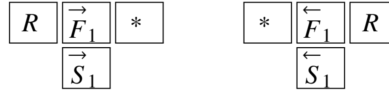
5. Fast messages bounce



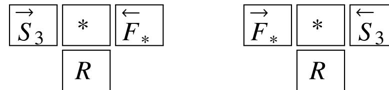
6. Rightmost node ready when first message arrives



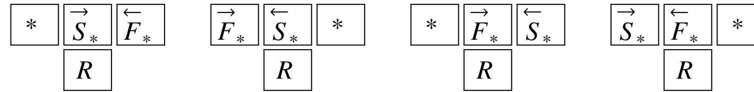
7. Start a slow message after fast message leaves



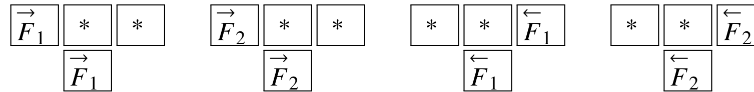
8. Pre-collision messages create ready nodes



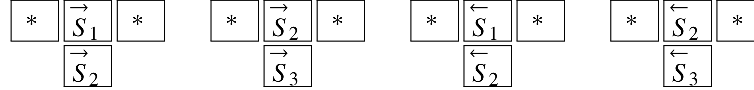
9. Colliding messages create ready nodes



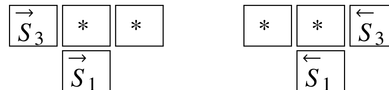
10. Fast messages move forward



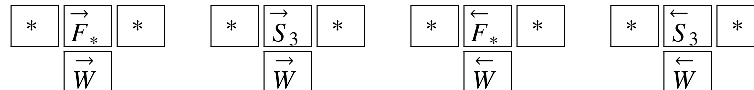
11. Slow messages wait



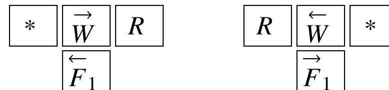
12. Slow messages move forward



13. Wait after sending a message



14. Newly ready nodes send fast messages



We give an illustration of an execution in Figure 6.1.

In order to prove the correctness of our construction, we formally introduce some terminology. We assume without loss of generality that $r_0 = 0$. We call

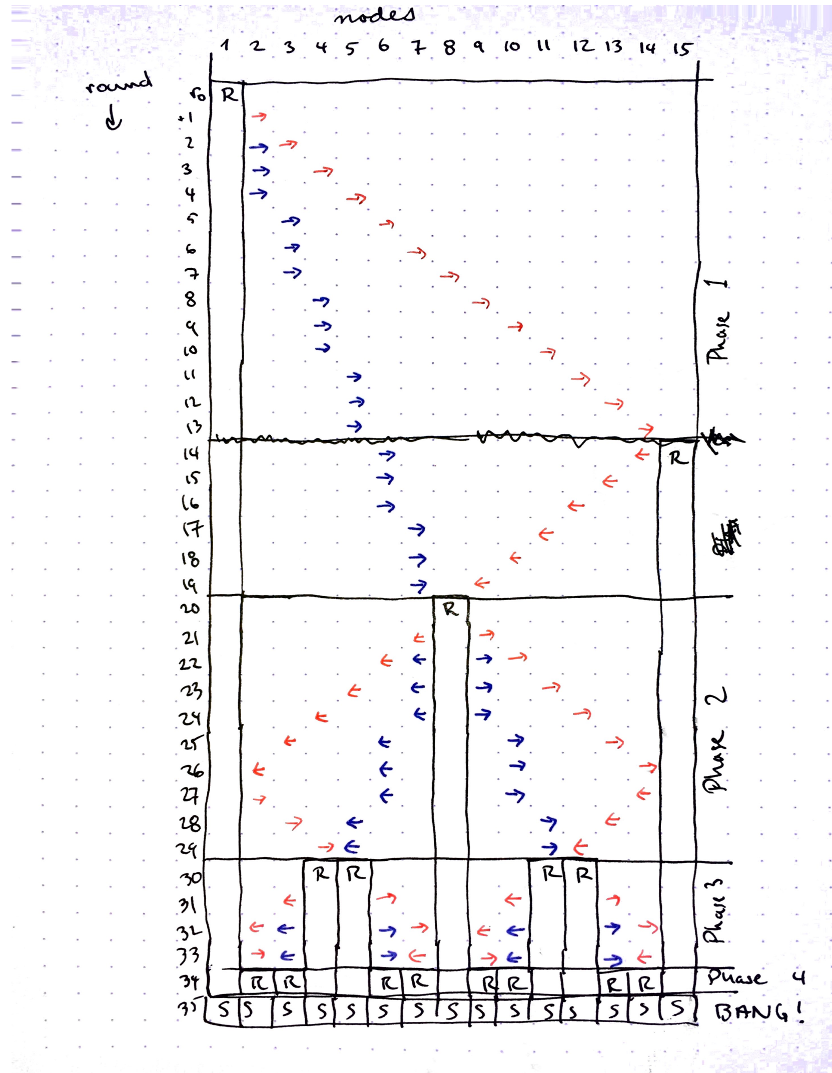


Figure 6.1
An illustration of the firing squad algorithm execution with $n = 15$ nodes.

an interval $I = [a, b]$ *active* in round r if v_{a-1} and v_{b+1} are state R or Q', but no v_i with $i \in I$ is in state R. We say that I *becomes active* in round r if I is an active interval in round r, but not in round $r - 1$.

Observation 6.7. *Suppose I becomes active in round $r > 0$. Then by Rule 2, in round $r - 1$ there was some active interval I' such that $I \subseteq I'$, and some node v_i with $i \in I'$ transitioned to state R in round r.*

We say that a *phase* begins in round r if some interval I becomes active in round r. By convention, the zeroth phase begins in round 0 when v_1 transitions to R. The *duration* of the phase is the number of rounds until another phase begins. In what follows, we will bound the duration of each phase and show that during each phase, all active intervals have the same size.

Lemma 6.8. *Suppose a phase begins in round $r > 0$, and the interval $I = [a, b]$ becomes active in round r. Let $c = \lfloor (a + b)/2 \rfloor$, $d = \lceil (a + b)/2 \rceil$, and $s = \lfloor 3\,|I|\,/2 \rfloor + 1$.*

Then the following hold.

1. *In round r, all nodes v_i with $i \in I$ are in the same state, either $\overleftarrow{W}$, $\overrightarrow{W}$, and nodes v_{a-1} and v_{b+1} are in state R.*
2. *I is active for all rounds $r, r + 1, \ldots, s - 1$.*
3. *If $|S| > 2$, then a new phase begins in round $r + s$ with intervals $I_\ell = [a, c - 1]$ and $I_r = [d + 1, b]$ becoming active.*
4. *If $|S| = 2$ then all nodes v_i with $i \in I$ transition to state R in round $r + s$.*

(Note: a proof sketch will be added later. You are not expected to verify this on your own.)

Proposition 6.9. *The 16 state machine described above solves the simultaneous restart problem on a network of size n in at most $3n$ rounds.*

Proof. For simplicity, we assume $n \geq 4$ and leave the cases $n = 2, 3$ as an exercise. We first prove that the state machine solves the simultaneous restart problem. To this end, observe that by Lemma 6.8, there is a sequence of rounds $r_0 < r_1 < \cdots$ such that a phase begins in each round r_i, and for each i, no phase begins in any round r satisfying $r_i < r < r_{i+1}$.

Claim. For every i and round $r \in [r_i, r_{i+1} - 1]$, all active intervals I have the same size, and all such intervals are deactivated in round r_{i+1}.

Proof of claim. We argue by induction on i. For the base case $i = 1$, there is a single active interval $I = [2, n - 1]$ in round r_0. By Lemma 6.8, I is deactivated in round $r_1 = r_0 + s$.

For the inductive step, suppose the claim holds for the phase beginning in round r_{i-1}, and let $I_1, I_2, \ldots, I_m$ be the intervals active during this phase. Applying Lemma 6.8 and the inductive hypothesis, all I_j are the same size, and each I_j becomes deactivated in round $r_i = r_{i+1} + s$, where s is computed as in Lemma 6.8. Now fix some $I = I_j$, and let I_ℓ and I_r be the intervals computed in Lemma 6.8. Then by the same lemma $|I_\ell| = |I_r| = \lfloor (|I| - 1)/2 \rfloor$, and either both intervals are empty, or both become active in round r_i. Thus, the claim holds for r_i as well.

By the claim, all active intervals during the ith phase have the same length, which we denote ℓ_i. Moreover, by Lemma 6.8, we have

$$\ell_{i+1} = \lfloor (\ell_i - 1)/2 \rfloor. \tag{6.2}$$

Therefore, there exists some minimal k for which $\ell_k \leq 2$. At the end of the kth phase, all active nodes transition to state R by item 4 or Lemma 6.8. Finally, by Rule 1 of the transition function, all nodes simultaneous transition to S in the subsequent round. This concludes the proof of correctness.

To prove the run-time of the algorithm, by Lemma 6.8, duration of the ith phase is $\lfloor 3\ell_i/2 \rfloor \leq \frac{3}{2}\ell_i$, where $\ell_0 = n - 2 < n$. Applying Equation (6.2) inductively, we have $\ell_i < \ell_0/2^i$. Therefore, the total duration is at most

$$\begin{aligned}\sum_{i=0}^{k} \frac{3}{2}\ell_i &< \sum_{i=0}^{k} \frac{3}{2}\frac{\ell_0}{2^i} \\ &< \frac{3}{2}n \sum_{i=0}^{\infty} 2^{-i} \\ &= 3n.\end{aligned}$$

This gives the desired bound on the round complexity. □

E6.4 Prove Proposition 6.9 in the cases $n = 2, 3$.

E6.5 In the preceding discussion, we assumed that only the “leader” node v_1 could spontaneously initiate the execution of the algorithm. Modify the finite state restart algorithm so that any single node can initiate a restart in a round. (Assume that only a single node initiates the restart during the execution.) What is the round complexity of the new algorithm?

Bibliography

MITPress NewMath.cls LaTeX Book Style Size: 6x9 December 18, 2020 6:02pm