CHAPTER 5

Information Theory Arguments

Information theory arguments rely on the fact that collecting information from many different processes takes a long time. They usually give bounds of the form $\Omega(\log n)$, where n is the number of processes in the system.

An information theory lower bound begins by carefully defining a measure of information, for example, the number of input values that influence the state of a process or the value of an object at a given point in time. Then a recurrence is used to describe how much the information can increase as a result of a single step.

Information theory arguments are mostly used to obtain lower bounds on the number of rounds in synchronous systems without process failures. These automatically imply lower bounds when processes can fail. In a synchronous system, each process takes one step per round until it completes or fails, so information theory arguments also give lower bounds for the number of steps taken by a process in an asynchronous system. But they are not generally useful for showing a problem is harder in an asynchronous system than in a synchronous system.

Throughout this chapter, our lower bounds do not assume any upper bounds on the sizes of the objects that are being used: the values that they contain can be arbitrarily large. This only makes the lower bounds stronger.

We begin this chapter with lower bounds for the simpler situation in which processes communicate using only single-writer registers. After developing an important technical lemma in Section 5.1, we apply it to obtain lower bounds for two problems: computing OR (Section 5.2) and approximate agreement (Section 5.3). Then we consider lower bounds when more powerful objects, including multi-writer registers, are available. In Section 5.4, we present another technical lemma, followed by two applications, to collect (Section 5.5) and to atomic snapshots (Section 5.6).

5.1 Lower Bounds using Single-Writer Registers

In Sections 5.1, 5.2, and 5.3, we consider the synchronous shared memory model where processes communicate using single-writer registers and no processes fail. To minimize the flow of information in each round, an adversary will schedule the processes performing reads before the processes performing writes. The order in which processes

read does not matter. Since no two processes write to the same register, the order in which processes write does not matter either.

A register is allowed to hold an arbitrarily large amount of information. Therefore, any number of single-writer registers that can be written to by the same process can be combined into one single-writer register with multiple components. Hence, there is no loss of generality in assuming that each process p_i has exactly one single-writer register r_i into which it may write. We may also restrict attention to full information algorithms, in which each process writes its entire history whenever it writes to its single-writer register. Then $C \overset{p_i}{\sim} C'$ implies that r_i has the same value in both C and C'.

Consider any synchronous execution α. The set of processes whose inputs process p_i knows about at the end of round t can be described by the recurrence:

$$I(p_i, t) = \begin{cases} \{p_i\} & \text{if } t = 0, \\ I(p_i, t-1) \cup I(p_j, t-1) & \text{if the step by process } p_i \text{ in round } t \\ & \text{of } \alpha \text{ is a read from register } r_j, \\ I(p_i, t-1) & \text{otherwise.} \end{cases}$$

More generally, the cardinality of the set $I(p_i, t)$ is a measure of the amount of information process p_i has at the end of round t of execution α. Let $I(t) = \max_{p_i}\{|I(p_i, t)|\}$. Then

$$\begin{aligned} I(0) &= 1, \text{ and} \\ I(t) &\leq 2I(t-1), \text{ for } t > 0. \end{aligned}$$

It is easy to prove by induction on t that $I(t) \leq 2^t$.

Let α_t denote the longest prefix of α in which no process has taken more than t steps. In other words, α_t consists of the first t rounds of α. In particular, α_0 is the empty execution starting from the initial configuration of α.

During α_t, process p_i only learns information about the input values of processes in $I(p_i, t)$. It does not learn any information about other input values. Formally, this means that p_i cannot distinguish α_t from the first t rounds of any other synchronous execution that has the same input values for all processes in $I(p_i, t)$.

Lemma 5.1. *Let α be a synchronous execution starting from an initial configuration C. Let β be a synchronous execution starting from another configuration C'. For any process p_i and any nonnegative integer t, if $C \overset{p}{\sim} C'$ for all $p \in I(p_i, t)$, then $\alpha_t \overset{p_i}{\sim} \beta_t$.*

Proof. The proof is by induction on t.

If $C \overset{p}{\sim} C'$ for all $p \in I(p_i, 0) = \{p_i\}$, then process p_i has the same state in C and C'. Since α_0 and β_0 are both empty executions, process p_i has the same local history in both. Thus $\alpha_0 \overset{p_i}{\sim} \beta_0$ and the claim is true for $t = 0$.

Let $t \geq 1$ and assume the claim is true for $t - 1$. Suppose that $C \overset{p}{\sim} C'$ for all $p \in I(p_i, t)$. Since $I(p_i, t-1) \subseteq I(p_i, t)$, it follows that $C \overset{p}{\sim} C'$ for all $p \in I(p_i, t-1)$. Hence, by the induction hypothesis, $\alpha_{t-1} \overset{p_i}{\sim} \beta_{t-1}$ and the local history of process p_i is the same in executions α_{t-1} and β_{t-1} and p_i is in the same state at the end of both these executions. Thus, the step performed by process p_i in round t will be the same in executions α_t and β_t. For the local history of process p_i to be the same in executions

α_t and β_t, it suffices to prove that this step, if it exists, has the same response in both executions.

Since writes always have the same response, it suffices to consider the case when the step by process p_i in round t is a read from the register, r_j, of some other process p_j. In this case, $I(p_j, t-1) \subseteq I(p_i, t)$. Hence, $C \overset{p}{\sim} C'$ for all $p \in I(p_j, t-1)$ and, by the induction hypothesis, $\alpha_{t-1} \overset{p_j}{\sim} \beta_{t-1}$. Therefore, the value of r_j is the same at the end of α_{t-1} and β_{t-1}. The reads in round t precede the writes in round t. It follows that process p_i reads the same value from r_j during round t in α_t and β_t. Therefore, the claim is true for t and, hence, by induction, for all $t \geq 0$. □

5.2 A Round Lower Bound for OR

We apply Lemma 5.1 to get a lower bound on the worst-case number of rounds to compute OR. In this problem, each process p_i has an input value $x_i \in \{0, 1\}$ and must output $x_0 \vee \cdots \vee x_{n-1}$. For a process to output 1, it only has to know that some input value is 1. However, to output 0, it has to know that all input values are 0. We focus on this case. Let C_0 denote the initial configuration in which all input values are 0.

Theorem 5.2. *Any synchronous algorithm for computing OR among n processes that only communicate using single-writer registers has an execution in which each process takes at least $\log_2 n$ steps.*

Proof. The proof is by contradiction. Let α denote any execution starting from C_0. Suppose there is a process p_i that outputs the OR in execution α within $t < \log_2 n$ steps. Since all the input values are 0, process p_i must output value 0 in α.

Since $|I(p_i, t)| \leq 2^t < n$, there is some process $p_j \notin I(p_i, t)$. Let C' denote the initial configuration in which $x_j = 1$, but all other input values are 0. Then $C_0 \overset{p}{\sim} C'$ for all $p \in I(p_i, t)$. Let β be an execution starting from C'. Lemma 5.1 implies that $\alpha_t \overset{p_i}{\sim} \beta_t$. Since process p_i outputs value 0 in α_t, it also outputs value 0 in β_t. But this is incorrect, since the OR of the input values in this execution is 1. □

If no processes fail, the OR can be computed in $O(\log n)$ rounds, using a binary tree to collect information.

Can we get a better (i.e., bigger) lower bound if all processes do not necessarily start at the same time? If the input values are private and all 0, then no process can output (the value 0) until after all of the processes have started, because, until then, it is possible that one of the input values is 1. This can be an arbitrarily long time.

So, suppose that the input value of each process p_i is initially in its single-writer register r_i. In this case, a process executing by itself from the initial configuration in which all of the input values are 0 would have to read the single-writer register of every other process. Thus, it must perform at least $n - 1$ steps.

If processes communicate using multi-writer registers, OR is easy to compute in a synchronous system in which processes don't fail: Assume that register r initially contains the value 0. In round 1, each process p_i with $x_i = 1$ writes 1 to r. In round 2, each process reads r and returns the value it contains. Thus, the $\Omega(\log n)$ lower bound on the number of rounds to compute OR using single-writer registers doesn't extend to multi-writer registers. Moreover, this implies that any synchronous simulation of

multi-writer registers using only single-writer registers will have an execution in which some operation takes $\Omega(\log n)$ rounds to be performed.

5.3 A Round Lower Bound for Approximate Agreement

Next, we consider the approximate agreement problem, which was defined in Section 2.3. If all processes are guaranteed to start in the same round, approximate agreement can be solved in two rounds, even for $\epsilon = 0$. In round 1, process p_0 writes its input x_0 to its register r_0 and outputs x_0. In round 2, all other processes read x_0 from r_0 and output x_0. So, the model considered in this section allows processes to start at different times (although, once a process has begun, it takes one step per round until it finishes). However, we will still prove a lower bound on the worst case number of steps taken by processes in executions in which all processes start at the same time. The lower bound holds even for a very restricted set of possible input values.

Theorem 5.3. *For any $\epsilon < 1$, any algorithm for approximate agreement among n processes that only communicate using single-writer registers has a synchronous execution in which no process outputs a value before round $\lceil \log_2 n \rceil$, even if all of the inputs $x_0, \ldots, x_{n-1}$ are restricted to be in $\{0, 1\}$.*

Proof. The proof is by contradiction. Let C_0 denote the initial configuration in which all input values are 0. Let α denote any synchronous execution starting from C_0. Let p_i be the first process that outputs a value in execution α. Suppose this happens during round $t < \lceil \log_2 n \rceil$. By validity, process p_i must output value 0 in α, since $\min\{x_1, \ldots, x_n\} = \max\{x_1, \ldots, x_n\} = 0$ in configuration C_0.

Since $|I(p_i, t)| \leq I(t) \leq 2^t < n$, there is some process $p_j \notin I(p_i, t)$. Let C_1 denote the initial configuration in which all input values are 1. Let γ denote the solo execution by process p_j starting from C_1 that continues until p_j halts. By validity, process p_j outputs value 1 during γ.

Let C denote the initial configuration in which $x_j = 1$ and all other input values are 0 and let γ' be the solo execution by process p_j starting from C. Since process p_j has the same state in C_1 and C and each register has the same value in C_1 and C, it follows that $\gamma \overset{p_j}{\sim} \gamma'$. Thus, process p_j outputs value 1 during γ'.

Let C' be the configuration at the end of γ'. Then $C_0 \overset{p}{\sim} C'$ for all $p \in I(p_i, t)$, since these processes take no steps in γ'. Let β be an execution starting from C'. Note that process p_j takes no steps in β, since it has halted in configuration C'. Lemma 5.1 implies that $\beta_t \overset{p_i}{\sim} \alpha_t$. Since process p_i outputs value 0 during α_t and takes no steps in γ', it also outputs value 0 during β_t and $\gamma'\beta_t$. But p_j outputs value 1 during γ' and, hence, during $\gamma'\beta$. This contradicts ϵ-agreement, since $\epsilon < 1$. □

Since an asynchronous system is more general than a synchronous system in which processes can start at different times, $\log_2 n$ is also a lower bound on the worst case number of rounds taken by a process to solve approximate agreement in an asynchronous system.

The lower bound presented in this section is from Hagit Attiya, Nancy Lynch, and Nir Shavit, *Are Wait-free Algorithms Fast?*, JACM, volume 41, number 4, 1994, pages 725–763. In the same paper, they also show this bound is tight (to within a constant

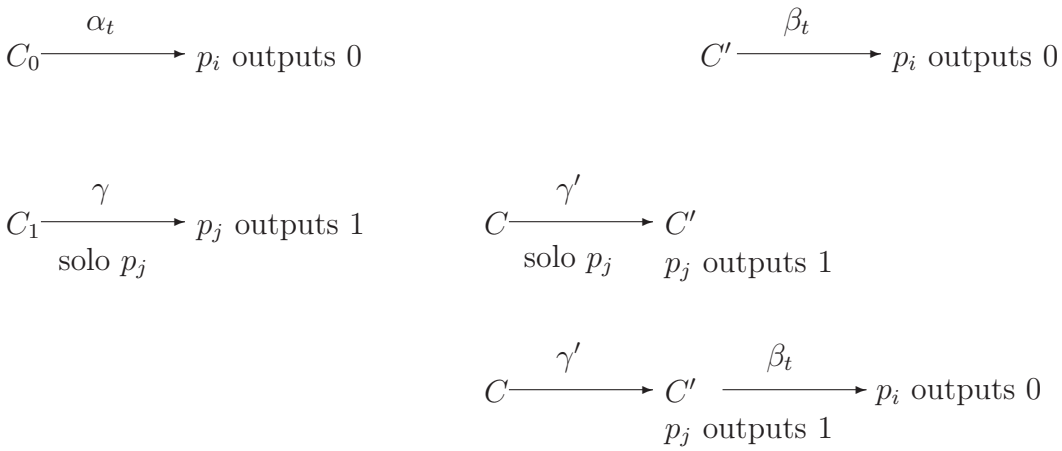

Figure 5.1. The executions considered in the proof of Theorem 5.3.

factor), even in an asynchronous system. Moreover, their algorithm tolerates process crashes, allows arbitrary inputs, and works for any value of ϵ.

For synchronous systems, the worst case step complexity of an algorithm is the same as its round complexity. However, for asynchronous systems, the worst case step complexity can be much larger, as the $\Omega(n)$ lower bound on step complexity for approximate agreement in Section 2.3 shows.

5.4 Lower Bounds using More Powerful Objects

Proving information theory lower bounds is more difficult when multi-writer registers are available or when other primitives, such as compare&swap, load-linked and store-conditional can be applied to objects in shared memory.

The compare&swap primitive (which we abbreviate as cas) applied to an object r atomically reads the value of r, compares it to the value of the parameter *old*, and, if they are the same, writes the value of the parameter *new* to r. It returns the value r had immediately before the cas was applied. We say that a cas is *successful* if its comparison was successful, i.e. if it returns the value *old*. Formally, cas can be specified by the following piece of code, where r is the object to which it is being applied.

```
cas(r, old, new)
temp ← r
if temp = old
then r ← new
return temp
```

If $old = new$, we say that the cas is *degenerate*.

The load-linked (LL) and store-conditional (SC) primitives work as follows. Like read, LL returns the value of the object to which it is applied. When a process performs SC(v) on an object, it either changes the value of the object to v and returns *true*, in

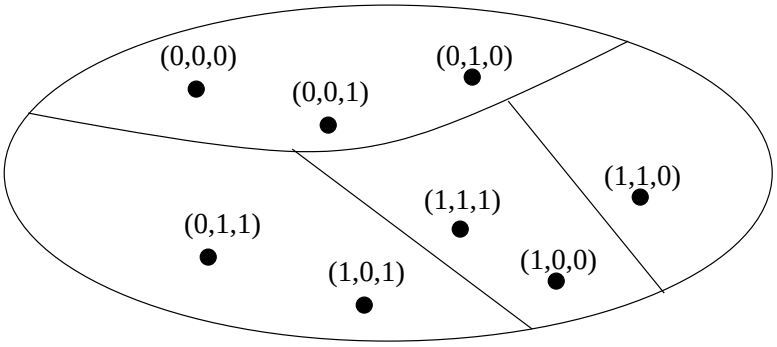

Figure 5.2. A partition of the set $\{0,1\}^3$.

which case we say it is *successful*, or it leaves the value of the object unchanged and returns $false$, in which case we say it is *unsuccessful*. The SC performed by a process on an object is successful if and only if no write, successful SC, or successful cas has been performed on the object since the process last performed LL on the object.

For the rest of this chapter, we consider the synchronous shared memory model in which no processes fail and processes communicate using the primitives read, write, cas, LL, and SC applied to any object. Once a process starts a computation, it performs exactly one of these primitives every round until it finishes. In the lower bound proofs, we restrict attention to certain synchronous executions. Specifically, in every round, all instances of read and LL are performed before any instances of cas, all instances of cas are performed before any writes, and all writes are performed before any instances of SC. Amongst the cas operations, the degenerate ones are scheduled first, followed by those whose old value is different from the value, v, in the object at the end of the previous round, followed by those whose old value is v. Note that, in each round, at most one successful non-degenerate cas or successful SC is applied to each object.

Consider any synchronous algorithm and let α_x denote a synchronous execution of this algorithm on the vector of inputs $x \in \{0,1\}^n$. Let $\alpha_{x,t}$ denote the prefix of α_x containing its first t rounds. When only single-writer registers are available, Lemma 5.1 implies that, if $x_j = x'_j$ for all $p_j \in I(p_i, t)$, then $\alpha_{x,t} \overset{p_i}{\sim} \alpha_{x',t}$.

In the case of more general objects, we also want to understand what different inputs lead to executions that are indistinguishable to a process. For every process p_i and every round t, we partition the set $\{0,1\}^n$ into equivalence classes such that $x, x' \in \{0,1\}^n$ are in the same class if and only if the executions $\alpha_{x,t}$ and $\alpha_{x',t}$ are indistinguishable to process p_i, i.e., $\alpha_{x,t} \overset{p_i}{\sim} \alpha_{x',t}$. Let $P(p_i, t)$ denote this partition and let

$$P(t) = \max_{p_i}\{ \text{ number of classes in } P(p_i, t) \}.$$

Note that, for every process p_i, the partition $P(p_i, 0)$ has 2 classes, since $\alpha_{x,0} \overset{p_i}{\sim} \alpha_{x',0}$ if and only if $x_i = x'_i$. Hence $P(0) = 2$.

Similarly, for every object r and every round t, partition the set $\{0,1\}^n$ into equivalence classes such that $x, x' \in \{0,1\}^n$ are in the same class if and only if the value of r is the same after executions $\alpha_{x,t}$ and $\alpha_{x',t}$. Let $V(r, t)$ denote this partition and let

$$V(t) = \max_{r}\{ \text{ number of classes in } V(r, t) \}.$$

Since every object has its initial value at the end of $\alpha_{x,0}$ for each $x \in \{0,1\}^n$, it follows that $V(0) = 1$.

Lemma 5.4. *$P(t), V(t) \leq 2^{2^t} n^{2^t - 1}$ for all nonnegative integers t.*

Proof. Let $t > 0$. The value of object r at the end of round t depends on which of the n processes last performs a write, a successful cas, or a successful SC on r during round t, if any. Each process p_i can change object r to at most $P(t-1)$ different values during round t, since it applies the same primitive with the same input parameters in round t of α_x and $\alpha_{x'}$, if x and x' are in the same class of the partition $P(p_i, t-1)$. If no process performs a write, a successful cas, or a successful SC on r during round t, the value of r at the end of round t depends on its value at the end of round $t-1$. Hence $V(t) \leq n \cdot P(t-1) + V(t-1)$.

If x and x' are in the same class of $P(p_i, t-1)$, then process p_i has the same state at the end of $\alpha_{x,t-1}$ and $\alpha_{x',t-1}$ and, hence, will take the same step (i.e. apply the same primitive, with the same parameter values, on the same object) in round t of both α_x and $\alpha_{x'}$. If this step is a write, then p_i will have the same state at the end of $\alpha_{x,t}$ and $\alpha_{x',t}$, so $\alpha_{x,t} \overset{p_i}{\sim} \alpha_{x',t}$. In this case, x and x' remain in the same class of $P(p_i, t)$. If this step applies SC, then p_i will get one of two possible responses, *true* or $false$ and this class of $P(p_i, t-1)$ is split into at most two different classes in $P(p_i, t)$. If this step is a read or LL, then x and x' remain in the same class of $P(p_i, t)$ only if the object they access has the same value at the end of round $t-1$ in α_x and $\alpha_{x'}$. Thus, in this case, a class in $P(p_i, t-1)$ is split into at most $V(t-1)$ different classes. Otherwise, this step applies cas to some object. Then the value returned is either one of the at most $V(t-1)$ values in the object at the beginning of the round or one of the at most $P(t-1)$ different values each of the other $n-1$ processes could have changed it to. In this case, a class in $P(p_i, t-1)$ is split into at most $V(t-1) + (n-1)P(t-1)$ different classes. Hence, in all cases, $P(t) \leq P(t-1) \cdot [V(t-1) + (n-1)P(t-1)]$.

It follows by induction that $P(t), V(t) \leq 2^{2^t} n^{2^t - 1}$ for all $t \geq 0$. □

The lemma can be extended to objects that also support k-cas. This primitive atomically compares the current values of k objects to *old* parameters and, if they are the same, assigns the values of *new* parameters to the objects. It returns a Boolean value, indicating whether it was successful. If, instead, the k-cas returns the values of the k objects accessed, then the lower bound becomes $\Omega(\log_k n)$. See Attiya and Hendler, *Time and Space Lower Bounds for Implementations Using k-CAS*, IEEE Transactions on Parallel and Distributed Systems, volume 21, 2010, pages 162–173.

5.5 A Round Lower Bound for Collect

In the *collect* problem, each process p_i has an input value $x_i \in \{0,1\}$ and must output the vector $x = (x_0, \ldots, x_{n-1})$ of all the input values. All processes are assumed to start performing COLLECT at the same time.

Theorem 5.5. *Any synchronous algorithm that COLLECTS n values requires $\Omega(\log n)$ rounds in the worst case.*

Proof. If the algorithm performs t rounds in the worst case, then $P(t) = 2^n$, since, on each vector of inputs, the processes must produce a different output and, hence, each vector of inputs must be in a separate class of the partition $P(p, t)$, for every process p. Therefore, by Lemma 5.4, $(2n)^{2^t} \geq P(t) \geq 2^n$, so $t \geq \log_2 n - \log_2 \log_2(2n)$. This implies that $t \in \Omega(\log n)$. □

This lower bound, for the case of multi-writer registers, is due to Paul Beame, *Limits on the Power of Concurrent-Write Parallel Machines*, Proceedings of STOC, 1986, pages 169–176. A matching upper bound can be obtained by collecting the information using a binary tree. Prasad Jayanti, *A Time Complexity Lower Bound for Randomized Implementations of Some Shared Objects*, PODC, 1998, pages 201-210, proves an equivalent lower bound for the LL, SC, validate, swap, and move primitives.

5.6 A Step Complexity Tradeoff for Synchronous Snapshots

In this section, we use Lemma 5.4 to get a tradeoff between the step complexities of two different operations of an implemented object. As in Section 5.3, processes may start performing their operations at different rounds.

A *(multi-writer) snapshot* object consists of $m \geq 2$ components and supports two operations: UPDATE(i, v), which sets the value of component i to v, and SCAN, which returns the current value of all m components. Initially, all components are 0. A *single-writer* snapshot is an n-component snapshot in which only process p_i can perform UPDATE on component i. Thus, in single-writer snapshots, no UPDATE operations to the same component overlap one another. For our lower bound, it suffices to consider restricted executions of a single-writer snapshot object in which processes $p_1, \ldots, p_{n-1}$ may each perform a single UPDATE and then process p_0 performs a single SCAN. In particular, the UPDATES are concurrent, but the SCAN does not overlap any UPDATE. Thus, the value of component i returned by the SCAN is either the value of the UPDATE performed by p_i, or the initial value of component i, if p_i did not perform an UPDATE. In these restricted executions, the SCAN performed by p_0 is like COLLECT, except that the other processes can coalesce information for p_0 before it starts.

Theorem 5.6. *Consider any implementation of a single-writer snapshot shared by n processes. If U is the worst case step complexity of UPDATE and S is the worst case step complexity of SCAN, then $U + \log_2 S \in \Omega(\log n)$.*

Proof. For every vector $x \in \{0, 1\}^{n-1}$, let C_x be the initial configuration where, for each $i \in \{1, \ldots, n-1\}$, process p_i has input x_i. Let α_x denote the U round execution starting from C_x in which process p_i performs UPDATE$(i,1)$ starting at round 1 for all $i \in \{1, \ldots, n-1\}$ with $x_i = 1$. Let σ_x denote the solo SCAN performed by p_0 starting in round $U + 1$ at configuration $C_x\alpha_x$. Note that, by assumption, all UPDATES are finished by the end of round U.

For each $t' \leq S$, let $\sigma_{x,t'}$ denote the prefix of σ_x containing its first t' rounds. Also let $P'(p_0, t')$ denote the partition of $\{0, 1\}^{n-1}$ into equivalence classes, where $x, x' \in \{0, 1\}^{n-1}$ are in the same class if and only if the executions $\alpha_x\sigma_{x,t'}$ and $\alpha_{x'}\sigma_{x',t'}$

are indistinguishable to process p_0. Let $P'(t')$ denote the number of classes in $P'(p_0, t')$. Since p_0 has no input and takes no steps in the first U rounds, $P'(0) = 1$.

Now let $0 < t' \leq S$ and suppose that x and x' are in the same class of the partition $P'(p_0, t'-1)$. Then process p_0 has the same state at the end of $\alpha_x\sigma_{x,t'-1}$ and $\alpha_{x'}\sigma_{x',t'-1}$ and, hence, will perform the same step in round $U + t'$ of both $\alpha_x\sigma_x$ and $\alpha_{x'}\sigma_{x'}$.

If this step is a write or SC to an object or a read, LL, or cas of an object to which p_0 has previously performed read, LL, write, or cas, then p_0 will have the same state at the end of $\alpha_x\sigma_{x,t'}$ and $\alpha_{x'}\sigma_{x',t'}$. In other words, $\alpha_x\sigma_{x,t'} \overset{p_0}{\sim} \alpha_{x'}\sigma_{x',t'}$. In this case, x and x' are also in the same class of the partition $P'(p_0, t')$.

Otherwise, this step is a read, cas, or LL of an object r that p_0 has not already accessed. Since r has not changed since round U, it can have at most $V(U)$ different values. If r had the same value at the end of α_x and $\alpha_{x'}$, then the vectors x and x', are in the same class of $P'(p_0, t')$. Thus, each class of $P'(p_0, t'-1)$ contains at at most $V(U)$ different classes of $P'(p_0, t')$. Hence, the number of classes in $P'(p_0, t')$ is bounded above by $V(U)$ times the number of classes in $P'(p_0, t'-1)$. In other words,

$$P'(t') \leq P'(t'-1) \cdot V(U).$$

It follows by induction that $P'(t') \leq (V(U))^{t'}$.

The SCAN by process p_0 completes by the end of round $U+S$. Thus $\sigma_x = \sigma_{x,S}$ for all $x \in \{0,1\}^{n-1}$. If $x' \neq x$, then the result returned by p_0's SCAN must be different in the executions $\alpha_{x'}\sigma_{x'}$ and $\alpha_x\sigma_x$. Thus x' and x are in different classes of $P'(p_0, S)$. Hence $P'(S) = 2^{n-1}$.

By Lemma 5.4, $V(U) \leq (2n)^{2^U}$. It follows that $2^{n-1} = P'(S) \leq (V(U))^S \leq (2n)^{S2^U}$. This implies that $U + \log_2 S \geq \log_2(n-1) - \log_2\log_2(2n) \in \Omega(\log n)$. □

When S denotes the number of objects used by the implementation, instead of the worst case step complexity of SCAN, it is still the case that $U + \log_2 S \in \Omega(\log n)$. This is because a step by process p_0 only increases the number of classes in the partition if it is a read, LL, or cas of an object that p_0 has not already accessed. Since there are only S objects, this can happen at most S times. Each time, the number of classes increases by at most a factor of $V(U)$, so $2^{n-1} \leq (V(U))^S$.

The lower bound in Theorem 5.6, for the case of multi-writer registers, is from Alex Brodsky and Faith Ellen Fich, *Efficient Synchronous Snapshots*, Proccedings of PODC, 2004, pages 70–79. This paper also constructs a family of linearizable (multi-writer) snapshot implementations from multi-writer registers whose step complexities of SCAN and UPDATE match the lower bound to within a constant factor. Hence the complexities of implementing a restricted single-writer snapshot (in which only a single fixed process is allowed to perform SCAN and only SCAN operations that do not overlap UPDATE operations are permitted) and a (multi-writer) snapshot are the same in this model. Moreover, allowing cas, LL, and SC primitives in addition to read and write does not decrease the complexity of implementing a synchronous snapshot object.