Tolerating Failures of Continuous-Valued Sensors

NASA -CR

Keith Marzullo

TR 90-1156
September 1990

P- 31

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported by the Defense Advanced Research Projects Agency (DoD) under NASA Ames Grant Number NAG2-593, Contract N00140-87-C-8904. The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

Tolerating Failures of Continuous-Valued Sensors

Keith Marzullo[*]
Cornell University
Department of Computer Science

September 14, 1990

Abstract

One aspect of fault-tolerance in process control programs is the ability to tolerate sensor failure. This paper presents a methodology for transforming a process control program that cannot tolerate sensor failures into one that can. Issues addressed include modifying specifications in order to accommodate uncertainty in sensor values and averaging sensor values in a fault-tolerant manner. In addition, a hierarchy of sensor failure models is identified, and both the attainable accuracy and the run-time complexity of sensor averaging with respect to this hierarchy is discussed.

Keywords: fault-tolerance, process control systems, real-time distributed systems.

1 Introduction

A *process control program* communicates and synchronizes with a physical process. Typically, the program reads values from the physical process through sensors and writes values through actuators, as shown schematically in Figure 1. This paper is concerned with tolerating failures of continuous-valued sensors.

The approach developed in this paper is outlined as follows:

[*] This work was supported by the Defense Advanced Research Projects Agency (DoD) under NASA Ames grant number NAG 2-593, Contract N00140-87-C-8904. The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

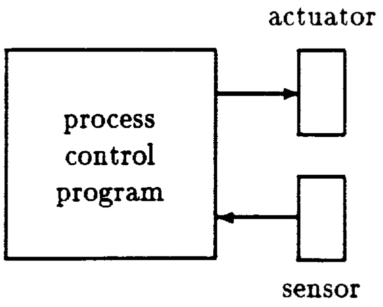

Figure 1: A process-control program

1. A specification of the control program is written in terms of the state variables of the physical system. For example, the specification of a program controlling a chemical reaction vessel would refer to a variable T whose value is assumed to be the temperature of the vessel.

2. Each physical state variable referenced by the specification is replaced with a reference to an *abstract sensor*. An abstract sensor is a set of values that contains the physical variable of interest. Uncertainty in sensor values now becomes an issue, and the specification must be re-examined and possibly changed to accommodate it.

3. The control program is written based on the specification produced by Step 2. This program reads abstract sensors that are assumed to always contain the correct value of the corresponding physical variables.

4. For each abstract sensor referenced by the program written in Step 3, a set of abstract sensors that fail independently are constructed. Each abstract sensor is implemented using a *concrete sensor*, which is a physical device that "reads" a physical variable[1], such as a thermometer. This step will require some knowledge of the physical process being controlled as well as the specification of the concrete sensor.

5. A *fault-tolerant averaging algorithm* is used with these replicated abstract sensor values in order to calculate another abstract sensor that

[1] The concrete sensor need not sense the exact physical state variable of interest. For example, an abstract temperature sensor could be constructed from a pressure gauge by using Boyle's law: $PV = nRT$.

is correct even if some of the original sensors are incorrect. The averaging algorithm assumes that no more than f out of the n abstract sensors are incorrect, where f is a parameter. The relation between n and f depends on the way sensors can fail.

The resulting system will have a structure like that shown in Figure 2. The rest of the paper is organized as follows. In Section 2, we define a method of representing sensors that makes them amenable to replication and discuss the effect of uncertainty on process control program specifications. In Section 3, we discuss sensor failure models and present a sensor averaging algorithm. Section 4 contains a demonstration of our methodology.

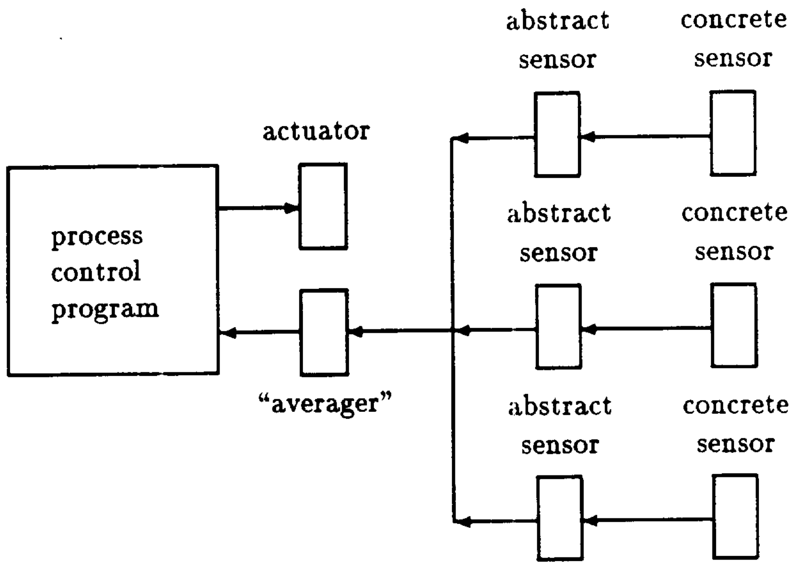

Figure 2: Replicated sensors

2 Physical State Variables and Concrete Sensors

A variable in a computer is quite different from a state variable in a physical process. A computer variable takes on values from a finite domain, and can assume only a bounded number of values in any finite time period. A physical state variable, however, may take on any real value at arbitrary times. A convenient way to represent a physical state variable in a computer program is as a function. The domain of such a function is typically *time*, but it can be some other physical variable, depending on the safety properties of interest.

A *concrete sensor* is a device that can be used to sample a physical state variable. For example, a computer controlling a reaction vessel might have a thermometer as a concrete sensor. A concrete sensor may interact with the computer in a variety of way: the computer may poll the sensor, the sensor may asynchronously alert the computer when a certain value is sensed, or the sensor may send a stream of values to the computer where each value indicates that the physical variable has changed by a certain amount. We will assume that a concrete sensor σ has a specification Φ_σ, and will call this sensor *faulty* if it exhibits a behavior not consistent with its specification.

For example, consider a thermometer whose value is read by polling. Suppose this concrete sensor returns a value $\hat{T}$ with an accuracy of ϵ degrees and the computer obtains the sensor's value within δ seconds of the thermometer being sampled. If the time the computer program receives $\hat{T}$ is $\hat{t}$, then the specification of this thermometer is:

$$\Phi(\hat{T}, \hat{t}) = \exists t_0 : \hat{t} - \delta \leq t_0 < \hat{t} : \hat{T} - \epsilon/2 \leq T(t_0) \leq \hat{T} + \epsilon/2$$

A concrete sensor is not very convenient mechanism. For example, with the thermometer:

- The sensor has a limited accuracy. Network delay and processor scheduling further limit the accuracy of the sensor.

- The control program may be interested in a temperature at a time the thermometer was not sampled. A value must then be interpolated; doing so requires knowledge of the physical process being monitored.

- Some properties of the concrete sensor, while important to the implementation, should be irrelevant to the specification used by the process control program. For example, another thermometer might generate an interrupt if the temperature rises above 100 degrees. This is an important property of the sensor–it allows for an accurate determination of when 100 degrees is reached. There may be other ways to make the same kind of precise measurement, however, for a sensor that is polled. It would be convenient if the control program could be the same for any method of measurement, as long as the measurement is accurate enough.

We will address these difficulties in two ways. The first problem cannot be eliminated, so in Section 2.2 the effect of inaccuracy in specifications is addressed. The other two problems, interpolation and data abstraction, are addressed here by *abstract sensors*.

2.1 Abstract Sensors

An *abstract sensor* is a piecewise continuous function from a physical state variable to a dense interval of real numbers. We will denote an abstract sensor with an overbar over the variable, such as $\overline{T}(t)$. When possible, we will simply write $\overline{T}$ if we are interested in the "current" value; that is, the sensor value for the current value of t. Intuitively, interval $\overline{T}$ represents the *possible values* of T, given the imprecision of the concrete sensor used to compute $\overline{T}$ and any uncertainty in the physical process.

An abstract sensor $\overline{T}(t)$ can be represented as a pair of functions $\overline{T}_{min}(t)$ and $\overline{T}_{max}(t)$, allowing $\overline{T}(t)$ to be the interval $[\overline{T}_{min}(t) \,..\, \overline{T}_{max}(t)]$. The *accuracy* of an abstract sensor is the width of the interval, or $|\overline{T}(t)|$. With this representation, $\mathbf{min}\,\overline{T}(t) = \overline{T}_{min}(t)$, $\mathbf{max}\,\overline{T}(t) = \overline{T}_{max}(t)$, and $|\overline{T}(t)| = \overline{T}_{max}(t) - \overline{T}_{min}(t)$.

An abstract sensor $\overline{T}$ is *correct* if it is not too inaccurate and always includes the value of the actual physical variable. More precisely, for some upper bound $acc_{\overline{T}}$ on the accuracy of $\overline{T}$,

$$\begin{aligned}&\overline{T} \text{ correct over } D \overset{\text{def}}{=} \\ &\qquad \forall t \in D : \mathbf{min}\,\overline{T}(t) \leq T(t) \leq \mathbf{max}\,\overline{T}(t) \ \wedge \ |\overline{T}(t)| \leq acc_{\overline{T}}\end{aligned}$$

We assume that a *failure* of an abstract sensor can arise when the underlying concrete sensor fails. As will be discussed in Section 3, a hierarchy of failure classes can be defined:

- fail-stop failures (following [17]), in which a failed abstract sensor can be detected[2];
- arbitrary failures with bounded inaccuracy[3], in which either $|\overline{T}(t)| \leq$

[2] The value of a failed fail-stop sensor can be defined to be the *empty interval* whose value is $[e \,..\, e-1]$ for some value of e. The empty interval has the convenient properties that it contains no points and intersects no interval, including itself.

[3] We use the term *bounded inaccuracy* to refer to bounding from above the accuracy of an abstract sensor. Similarly, an abstract sensor is *too inaccurate* if the numeric value of its accuracy is too large.

$acc_{\overline{T}}$ is always true or $acc_{\overline{T}}$ is known, and thus abstract sensors that are too inaccurate can be detected;

- arbitrary failures, in which an abstract sensor can fail arbitrarily.

Given a concrete sensor, it may not be easy to implement an abstract sensor. In general, it may require considerable knowledge about the physical process being monitored. For example, consider the specification $\Phi(\hat{T}, \hat{t})$ for the polled thermometer. The specification, alone, is not sufficient information to define an abstract sensor $\overline{T}$, since we don't know how to interpolate values between successive sensor readings. Suppose, however, we know from the physical process being monitored that $|\frac{dT}{dt}| \leq \Delta_T$. This bound on the change of T allows us to interpolate intermediate values with a known accuracy. The abstract sensor $\overline{T}(t)$ can be defined as

$$\hat{T} - \epsilon/2 - \Delta_T(t - \hat{t} + \delta) \;\leq\; \overline{T}(t) \;\leq\; \hat{T} + \epsilon/2 + \Delta_T(t - \hat{t} + \delta) \quad \text{for } t \geq \hat{t}$$

One can use this example as a recipe for writing abstract sensors, but the resulting sensor may be too inaccurate for any practical use. For example, if $|\frac{dT}{dt}|$ can be bound more tightly at certain known times, a more accurate sensor can be constructed. In Section 4, the development of an abstract sensor is shown in some detail.

2.2 Abstract Sensors in Specifications

The specification of a system typically includes a set of *safety conditions*: predicates on the state of the system that the implementation must ensure are always true. A safety condition on a process control program will reference physical state variables. For example, consider a reaction vessel with a pressure relief valve. One safety condition might be that whenever the pressure p is greater than some ceiling p_{max}, the valve must be open. We could write this safety condition as $p > p_{max} \equiv open$, where *open* is a state function that is true when the valve is open.

The specification of a process control program will have to be changed when expressed in terms of abstract sensors. It is not possible to take a control program written in terms of physical state variables and, for each reference to such a variable, substitute a reference to a corresponding abstract sensor. Consider $p > p_{max} \equiv open$. The condition that results from replacing the physical state variable with an abstract sensor is $\overline{p} > p_{max} \equiv open$; one must decide what the term $p > p_{max}$ means.

Let S be a predicate on the system state and V be the set of physical variables mentioned in S that will be accessed through abstract sensors. We need another condition S' that contains no references to any $v_i \in V$ but may instead contain references to $\overline{v}_i$. The only constraint on S' is that it reduces to S when the abstract sensors have perfect accuracy [4]:

$$(S' \wedge |\overline{v}_i| = 0) \Rightarrow S^{v_i}_{\overline{v}_i}$$

There are several ways such an S' can be constructed. We could replace all references to v_i in S with references to the midpoint of $\overline{v}_i$. However, if all values in $\overline{v}_i$ have the same likelihood of being valid, then there are only two reasonable alternatives. We can either require that *all* points in $\overline{v}_i$ satisfy S or that ***there exists at least one*** point in $\overline{v}_i$ that satisfies S. More precisely, for each physical variable v_i the condition S can be generalized as

$$S' \overset{\text{def}}{=} \forall v_i \in \overline{v}_i : S \quad \text{or} \quad S' \overset{\text{def}}{=} \exists v_i \in \overline{v}_i : S$$

The generalization of S cannot be done automatically, since it is really a refinement of the problem specification. Ideally, one would like to strengthen S so that states excluded by the safety condition are still excluded. For example, we might want to assert that a catalyst is injected (denoted by the state function C) only when the pressure is above a minimum value: $C \Rightarrow (p > p_{min})$. In this case, the state we are trying to avoid is one where the catalyst is injected at too low a pressure, and we can strengthen $C \Rightarrow (p > p_{min})$ to $C \Rightarrow (\forall p \in \overline{p} : p > p_{min})$.

We may find, however, that a specification cannot be strengthened in a meaningful way. The property $p > p_{max} \equiv open$ is an example. Changing the property to $(\forall p \in \overline{p} : p > p_{max}) \equiv open$ will allow states with $p > p_{max}$ and $\neg open$, and changing the specification to $(\exists p \in \overline{p} : p > p_{max}) \equiv open$ will allow states with $p \leq p_{max}$ and $open$. Unless we can guarantee that $|\overline{p}| = 0$, the program's specification must be changed. Here, we are probably more interested in avoiding an explosion of the vessel. If so, the condition we want is $(\exists p \in \overline{p} : p > p_{max}) \equiv open$, and we would accept the fact that the pressure valve may be unnecessarily open.

It shouldn't be surprising that, in some cases, a property of a specification must be changed (as compared to being strengthened) when references

[4] The expression $S^{v_i}_{\overline{v}_i}$ is S with all occurrences of physical state variable v_i changed to abstract sensor $\overline{v}_i$.

to physical state variables are replaced with references to abstract sensors. Using abstract sensors exposes uncertainty in the physical process' state and a specification may have been written implicitly assuming no such uncertainty. Of course, specifications are sometimes written with such uncertainty explicitly mentioned. For example, an informal expression of the pressure relief valve property might be "if the pressure rises to within 0.1 millibars of p_{max} then the relief valve must open". In our notation, this property would be expressed as $((\exists p \in \overline{p} : p > p_{max}) \equiv open) \wedge (|\overline{p}| \leq 0.1)$.

3 Fault-Tolerant Abstract Sensors

Given n independent abstract sensors and some assumptions about failures, we would like to construct an abstract sensor that is tolerant of failures. We will first present an algorithm that constructs a sensor containing the correct value given that no more than f of the original sensors are not correct. We will then consider how this algorithm performs with different failure models.

3.1 Fault-Tolerant Sensor Averaging

Let $\overline{T}_i$ and $\overline{T}_j$ $(i \neq j)$ be two abstract sensors for the same physical value T. If $\overline{T}_i$ and $\overline{T}_j$ both contain the correct value, then the intervals $\overline{T}_i$ and $\overline{T}_j$ must intersect, and their intersection must contain the (unknown) value T.

If f or less sensors do not contain the correct value, then any $(n-f)$-*clique*, or set of $n-f$ mutually intersecting sensors may contain the correct value, since they each share a common value. Conversely, any point not contained in at least $n-f$ intervals cannot be the correct value; if it were, then there would be more than f sensors that do not contain the correct value. So, the cover of all $(n-f)$-cliques must contain the correct value. This gives us an abstract sensor averaging algorithm.

Algorithm 1 *Fault-tolerant Sensor Averaging*

> Let S be a set of values taken from n abstract sensors, and suppose the abstract sensors are of the same physical state variable where their values were read at the same point in their domain (*e.g.* at the same time). Assuming that at most f of these sensors are incorrect, calculate $\cap_{f,n}(S)$ which is the smallest interval that is guaranteed to contain the correct physical value.

Implementation: Let l be the smallest value contained in at least $n - f$ of the intervals in $\mathcal{S}$ and h be the largest value contained in at least $n - f$ of the intervals in $\mathcal{S}$ (by assumption, these values must exist). Let $\cap_{f,n}(\mathcal{S})$ be the interval l .. h].

Algorithm 1 is inexpensive–it can be implemented in $O(n \log n)$ time. Appendix B gives an implementation that has this running time.

The accuracy of $\cap_{f,n}(\mathcal{S})$ depends on the value of f, as illustrated in Figure 3. In this example, the value of $\cap_{0,n}(\mathcal{S})$ is the empty interval because it is impossible for both intervals a and b to contain the correct value; at least one of them must be incorrect. In general and when defined, $\cap_{0,n}(\mathcal{S})$ is the intersection of the intervals in $\mathcal{S}$, $\cap_{n-1,n}(\mathcal{S})$ is the cover of the intervals in $\mathcal{S}$, and $| \cap_{f,n} (\mathcal{S})| \leq | \cap_{f',n} (\mathcal{S})|$ if $f \leq f'$.

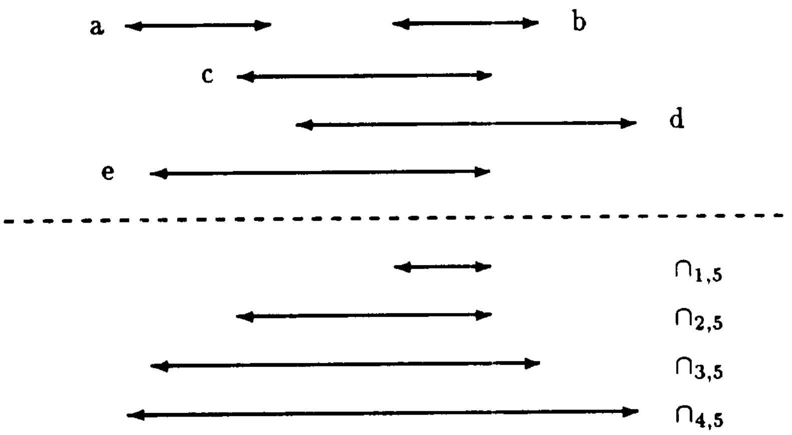

Figure 3: Intersection with $f = 1, 2, 3$ and 4

One consequence of the definition of $\cap_{f,n}(\mathcal{S})$ is that for $f > 0$, $\cap_{f,n}(\mathcal{S})$ can contain values that cannot be the correct value. For example, Figure 4 shows the intersection of three intervals a, b and c. If $f = 1$ then the correct value must be within $I1$ or $I2$. Algorithm 1, however, would calculate the interval I. The points between $I1$ and $I2$ are added to preserve the "shape" of the abstract sensor as seen by the control program.

It is instructive to compare $\cap_{f,n}(\mathcal{S})$ with *n-modular redundancy* [20] (NMR). In NMR, n independently produced values of a variable are presented to a voter that selects the majority value as its output. By doing so, the voter can mask up to f incorrect inputs where $n \geq 2f + 1$. The function $\cap_{f,n}(\mathcal{S})$ resembles an NMR voter, except that it accepts intervals rather than points as inputs and it produces the most accurate value possible as output for *any* value of $f : (0 \leq f < n)$. If the inputs to $\cap_{f,n}(\mathcal{S})$ are point intervals (that is, have a width of zero), then the NMR voter and $\cap_{f,n}(\mathcal{S})$ produce the same output when $n \geq 2f + 1$.

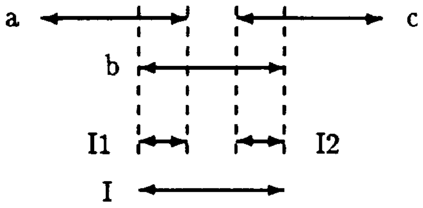

Figure 4: Intersection with $n = 3$ and $f = 1$

The relation of f to n (and hence the accuracy of $\cap_{f,n}(\mathcal{S})$) depends on the failure model that is assumed. We will first assume arbitrary failures (both with and without bounded inaccuracy) and then consider a fail-stop failure model. We assume that no more than f of the n sensors can be faulty and that once failed, a sensor remains failed.

3.2 Arbitrary Failures

The width of an interval that is an abstract sensor value determines the sensor's accuracy. If the ratio f/n of the number of faulty to non-faulty abstract sensors is too large, then one cannot bound the inaccuracy of the resulting abstract sensor. The following theorem bounds f/n. Define the functions $\mathbf{min}_i$ and $\mathbf{max}_i$ to be the i^{th} smallest and largest values of a set of n values respectively. Note that $\mathbf{min}_i$ is the same as $\mathbf{max}_{n-i+1}$. For example, if $\mathcal{S} = \{13, 14, 15\}$ then $\mathbf{min}_3(\mathcal{S}) = \mathbf{max}_1(\mathcal{S}) = 15$.

Theorem 1 *If* $f < \left\lfloor \frac{n+1}{2} \right\rfloor$ *then* $|\cap_{f,n}(\mathcal{S})| \leq \mathbf{min}_{2f+1}\{|s| : s \in \mathcal{S}\}$.

The proof of this theorem is in Appendix A.

If $f \geq \lfloor (n+1)/2 \rfloor$ then the derived interval can be more inaccurate than any sensor in the system. Theorem 4 in Appendix A formally states this property. An example is shown in Figure 5. Suppose the three sensors a, b

and c are "maliciously" faulty. They can make $\cap_{f,n}(S)$ as inaccurate as desired by choosing appropriately distant values from intervals d and e.

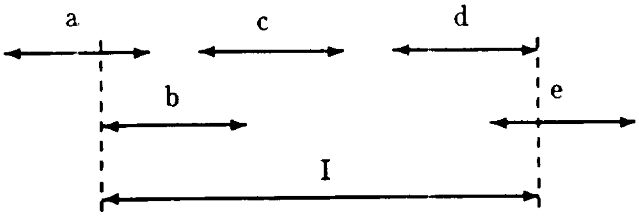

Figure 5: Intersection with $n = 5$ and $f = 3$

One property of $\cap_{f,n}(S)$ is that, depending on the values of S, $\cap_{f,n}(S)$ can be more accurate than any sensor in S. Figure 6 illustrates this property. Such a value of S can result from different delays, errors, or other sources of uncertainty that arise in computing the value of the abstract sensors comprising S. This property makes replication of abstract sensors attractive not only for tolerating failures, but also for increasing the expected accuracy of a sensor's value.

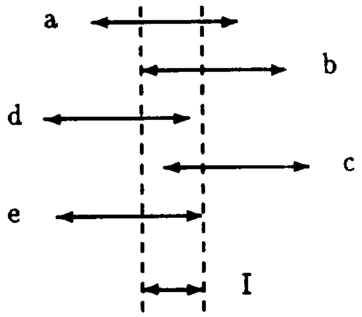

Figure 6: Intersection with $n = 5$ and $f = 1$

If $n = 2f + 1$, however, then the accuracy of $\cap_{f,n}(S)$ is limited, in that it cannot be more accurate than the most accurate sensor in S. This is illustrated in Figure 7 where $f = 1$ and $n = 3$; here, the only way we could change sensor c so that it contains values outside of $\cap_{f,n}(S)$ would be to make c more inaccurate than either a or b or to make c detectably faulty (as discussed in Section 3.3). It is, therefore, advantageous to have $n > 2f+1$ for a system with arbitrary failures. Theorems 5 and 6 in Appendix A formally state this property.

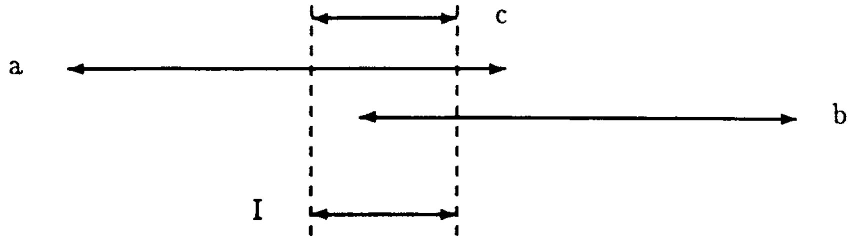

Figure 7: Intersection with $n = 3$ and $f = 1$

Theorem 1 bounds the accuracy of a derived abstract sensor in terms of the accuracy of one of the abstract sensors $\overline{s}_i$ used in its construction. Such a bound is useful only if $\overline{s}_i$ is not faulty—in particular, if $|\overline{s}_i| \leq acc_{\overline{s}}$. Hence, Theorem 1 only applies for arbitrary failures with bounded inaccuracy. However, if this bounding sensor could have an erroneously large inaccuracy, then the bound is not meaningful. Consider the sensors shown in Figure 8. If sensor c is erroneously inaccurate, then the value of $\cap_{1,3}(\mathcal{S})$ is as inaccurate as c. Thus, the ratio f/n of the number of faulty to non-faulty abstract sensors must be smaller than that stated in Theorem 1 when sensors can have unbounded inaccuracy. Theorem 2 gives this bound on f/n.

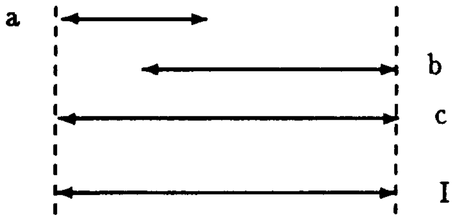

Figure 8: Intersection with $n = 3$ and $f = 1$

Theorem 2 *Let C be the (unknown) subset of S that are correct. If $f < \lfloor \frac{n}{3} \rfloor$ then* $| \cap_{f,n} (\mathcal{S})| \leq \mathbf{min}_{f+1}\{|s| : s \in \mathcal{C}\}$.

The proof of this theorem is simple: from Theorem 1,

$$|\cap_{f,n}(\mathcal{S})| \leq \mathbf{max}_{n-2f}\{\forall s \in \mathcal{S}\}$$

For $|\cap_{f,n}(\mathcal{S})|$ to be bounded by a correct sensor, $n - 2f > f$ and so $n > 3f$. The worst case is when f faulty sensors are the most inaccurate, so

$$|\cap_{f,n}(\mathcal{S})| \leq \mathbf{min}_{f+1}\{|s| : s \in \mathcal{C}\}$$

□

Under the hypothesis of Theorem 2, a minimum of four sensors are necessary to tolerate a single faulty sensor. Figure 9 illustrates this case—even if sensor d has an erroneously large inaccuracy, $| \cap_{1,4} (\mathcal{S})|$ is bounded by a nonfaulty sensor.

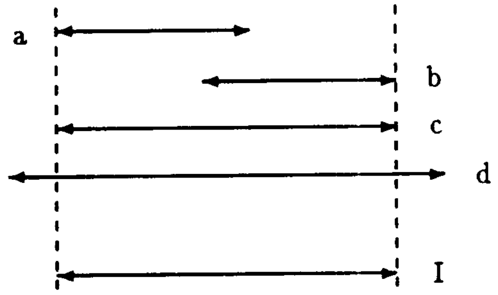

Figure 9: Intersection with $n = 4$ and $f = 1$

3.3 Other Issues on Failure

If $f' \leq f$ sensors can be detected as failed then they can be removed from $\mathcal{S}$, and n and f can be reduced by f' before computing $\cap_{f,n}(\mathcal{S})$. By doing so, the ratio f/n will be decreased, thereby improving the bound on the inaccuracy of $\cap_{f,n}(\mathcal{S})$. In a fail-stop failure model, *all* sensor failures are detectable, meaning that up to $n - 1$ failures can be tolerated and $\cap_{f,n}(\mathcal{S})$ will be as accurate as the most accurate nonfaulty sensor. Additionally, the running time of Algorithm 1 with fail-stop sensors is $O(n)$.

We can use Algorithm 1 to detect some failed abstract sensors assuming an arbitrary failure model. This algorithm is very simple: any sensor in $\mathcal{S}$ that does not intersect $\cap_{f,n}(\mathcal{S})$ cannot contain the correct value, and is therefore incorrect.

Algorithm 2 *Detecting failed sensors.*

> Given n sensors $\mathcal{S}$ and a maximum number of faulty sensors f, find a subset of the sensors $\mathcal{D} \subseteq \mathcal{S}$ that are incorrect.
>
> *Implementation:* Compute $\cap_{f,n}(\mathcal{S})$ using Algorithm 1. Then, $\mathcal{D} = \{s : s \in \mathcal{S} \wedge s \cap (\cap_{f,n}(\mathcal{S})) \ = \emptyset\}$.

It is likely that Algorithm 2 will fail to detect some of the incorrect sensors. For example, using Algorithm 2 with the sensors in Figure 4 yields $\mathcal{D} = \emptyset$; even though we know that only one of the two sensors a, c must be incorrect, we cannot tell which of the two is incorrect.

So far, we have assumed that once a sensor fails it remains failed. This assumption may not be realistic for sensors, since an abstract sensor maintains no state. It seems natural to assume a sensor may occasionally fail in an apparently malicious way and then "heal" itself and subsequently yield correct values. So, a natural extension to the arbitrary failure model is to denote the faulty sensors at time t as a function $\mathcal{F}(t)$ such that $\forall t : |\mathcal{F}(t)| \leq f$. Unfortunately, we cannot construct a correct abstract sensor under these conditions; the averager might be unlucky and each time read a (temporarily) incorrect abstract sensor. We must also guarantee that there exists a period Π such that the number of failures in all time intervals of length Π is bounded:

$$\exists \Pi > 0 : \forall t, t' : t \leq t' \leq t + \Pi : | \cup \mathcal{F}(t')| \leq f.$$

If Algorithm 1 obtains values from each concrete sensor within Π time units then it constructs a correct abstract sensor. In the limit of large Π, this model reduces to the earlier arbitrary failure model.

4 Example

The methodology presented in this paper requires some thought to use. An original specification may have to be changed to accommodate abstract sensors, and it may be difficult to construct a set of independent abstract sensors. In this section, we show an example of how a specification can be converted from one that uses physical state variables to one using abstract sensors. We also show how an abstract sensor can be implemented from a concrete sensor.

As part of the *Cornell Real-Time Reliable Distributed Systems (RR)* project, we are deriving correct process control programs from specifications. One of the problems we have chosen is that of a train traversing a sequence of n adjacent track segments of possibly unequal lengths. Assume that segment i spans track locations c_i through c_{i+1} where $(\forall i : 0 \leq i < n : c_i < c_{i+1})$. A train has position $x(t)$ and velocity $v(t)$, has zero length[5], starts at position $c_0 = 0$ and moves in the direction of increasing x (towards c_1). Each track segment has an associated minimum and maximum speed min_i and max_i; if the train exceeds these limits, it may derail. Additionally, there is a random *communications delay* associated with all messages in the system that is bounded by δ seconds.

A *track circuit* $\sigma_{(q,r)}$ is a concrete sensor associated with a span of track $q \leq x \leq r$. A nonfaulty track circuit returns *true* iff the train occupies any part of the circuit's span at the time the circuit is polled. We will assume that there are M track circuits.

The safety condition for correct operation of the train is that it not derail, or

$$S \overset{\text{def}}{=} \forall t, i : 1 \leq i \leq n : c_i \leq x(t) \leq c_{i+1} \Rightarrow min_i \leq v(t) \leq max_i$$

S is expressed in terms of physical variables, so it must be changed to be expressed in terms of abstract sensors. The obvious condition is

$$\begin{aligned} S' \overset{\text{def}}{=} & \forall t, i : 1 \leq i \leq n : \\ & (\exists x \in \overline{x(t)} : c_i \leq x \leq c_{i+1}) \Rightarrow (\forall v \in \overline{v(t)} : min_i \leq v \leq max_i) \end{aligned}$$

since this also excludes all unsafe states (at a penalty of running the train conservatively).

Since the condition S' refers to the abstract sensors $\overline{x}$ and $\overline{v}$, the control program will need to refer to these sensors. We will show how an abstract position sensor $\overline{x}_i$ can be constructed from the track circuits $\sigma_{(q,r)}$. The simplest way to do this is to assume a bound on the velocity of the train $v \leq v_{max}$. Define the global array of M elements:

[5] In Appendix C we show that controlling a train of length $L > 0$ is equivalent to controlling a train of zero length.

```
var train[i]: {before, in, after} := before, ..., before;
```

Define a *polling process* for each track circuit $\sigma_{(q,r)}$. Note the delay is represented by a **delay** statement; the implementation must ensure that no more than Δ seconds elapse between successive polls of a sensor where Δ is small enough so that the polling process does not "miss" the train traversing the track segment it is monitoring: $\Delta \leq (r - q - \delta v_{max})/v_{max}$. Assertion I is a loop invariant, and t is the current time.

```
process Poll[i] =
        begin
        {$I$ : train[i] = before $\Rightarrow 0 \leq x(t) < q + \delta v_{max} \wedge$
              train[i] = in $\Rightarrow q \leq x(t) \leq r + \delta v_{max} \wedge$
              train[i] = after $\Rightarrow r < x(t) \leq c_n$}
        do true $\rightarrow$
              delay $\Delta$;
              if $\sigma_{(q,r)} \wedge$ (train[i] = before) $\rightarrow$ train[i] := in
              [] $\neg\sigma_{(q,r)} \wedge$ (train[i] = in) $\rightarrow$ train[i] := after
              [] $\neg\sigma_{(q,r)} \wedge$ (train[i] = before) $\rightarrow$ skip
              [] $\sigma_{(q,r)} \wedge$ (train[i] = in) $\rightarrow$ skip
              [] (train[i] = after) $\rightarrow$ skip
              fi
        od;
        end
```

The definition of the abstract sensor comes from the loop invariant I and the distance the train could have moved since the last time $\sigma_{(q,r)}$ was read:

```
$\overline{x}_i$ = if train[i] = before $\rightarrow$ [0 .. $q + (\delta + \Delta)v_{max}$]
      [] train[i] = in $\rightarrow$ [$q$ .. $r + (\delta + \Delta)v_{max}$]
      [] train[i] = after $\rightarrow$ [$r$ .. $c_n$]
      fi
```

Fault-tolerance is achieved by constructing an abstract position sensor from each track circuit and then using Algorithm 1. Additional fault-tolerance could be achieved by replicating the track circuit for each track circuit.

The abstract sensor developed here is too simplistic to be of any real use. Correct track circuits far away from the train give very inaccurate bounds on the train's location, and by Theorem 2 the accuracy of the fault-tolerant abstract sensor will be poor for any reasonable f. In the actual system, we make use of an abstract sensor $\overline{x}_0(t)$ whose value is derived from the initial condition $x(0) = 0$ and from the commands sent to the train. We call this abstract sensor a *model sensor* since if it is incorrect, then either the control program is faulty or the specification of the environment was incorrect. The model sensor is initially very accurate, and can be used to detect some of the failures of the abstract sensors $\overline{x}_i$. Having a model sensor also simplifies the computation of the other abstract sensors. The train has the property that if an abstract sensor $\overline{x}_i$ is computed from a fixed set of track circuit polls and the commands sent to the train, then the interval $[\overline{x}_i(t).min - \overline{x}_0(t).min \mathrel{..} \overline{x}_i(t).max - \overline{x}_0(t).max]$ is a constant. So, the implementation of $\overline{x}_i$ computes an accurate value of $[\overline{x}_i(t).min - \overline{x}_0(t).min \mathrel{..} \overline{x}_i(t).max - \overline{x}_0(t).max]$ at the time t it notes the track circuit first coming on, and computes $\overline{x}(t')_i$ for $t' > t$ as $[\overline{x}(t').min_0 + \overline{x}_i(t).min - \overline{x}_0(t).min \mathrel{..} \overline{x}(t').max_0 + \overline{x}_i(t).max - \overline{x}_0(t).max]$. The implementation of $\overline{x}_i$ can do the same computation when the track circuit subsequently goes off, and if the two resulting values of the abstract sensor do not intersect then the abstract sensor is faulty.

For our program, it is necessary to ensure that that $|\overline{x}(t)| \leq acc_x$ where acc_x is length of the shortest track segment. Given a value of Δ, one can estimate the accuracy of abstract sensors near the train, as these will be the most accurate. The abstract sensors $\overline{x}_i$ have a known bound on their accuracy, so Theorem 1 can be used to find the maximum value of f that will guarantee $|\overline{x}(t)| \leq acc_x$.

5 Discussion

This paper presents a five-step process, through which a program written in terms of physical state variables can be transformed into one that reads the physical state variable through a set of concrete sensors, some of which may be faulty. The degree of sensor replication depends on the failure model being assumed. Figure 10 summarizes the maximum number of faulty sensors that can be tolerated for the three failure models considered in this paper, assuming that an unboundedly accurate sensor is desired.

The work presented here is part of the general problem of *input reification* [9]. The results in this paper are a generalization of the work done by the author and presented in [15,14]. This earlier work looked at the problem of *clock synchronization* in a distributed system. A clock is a special kind of

Failure Model	f_{max}	min n: $f = 1$	min n: $f = 2$
arbitrary failures, unbounded inaccuracy	$\lfloor(n-1)/3\rfloor$	4	9
arbitrary failures, bounded inaccuracy	$\lfloor(n-2)/2\rfloor$	4	6
fail-stop failures	$n-1$	2	3

Figure 10: Maximum failures for different error models

sensor, in that the physical process it senses can be expressed simply.

The approach presented in Section 2.2 concerning transforming specifications is novel. Much work has been done on expressing and determining the validity of properties that refer to real time (for example, [7,19]), but usually these specifications are typically written in terms of physical state variables where, for each variable, an *a priori* upper bound on its accuracy is known.

The methodology presented in this paper is related to the *state machine approach* [18,10]. A set of sensors of the same physical value can be thought of as a set of identical processors that return intervals rather than scalar values. In both cases, failures are masked by replication and voting.

Studies on hierarchies of failure models (for example, [2,16]) originally arose in the context of the agreement problem [5]; a problem not addressed here. If the control program were to be replicated, then the processes of this program would need to use an agreement protocol to disseminate the sensor's values [3,4,8]. There has been work on agreement on the value of sensors. For example, the *inexact agreement* problem discussed in [13] relates the accuracy of the agreement value with respect to the number of rounds the protocol executes. A different approach to agreement among sensors is taken in [12], in which sensor failure is not considered.

The methodology presented here is incomplete. For example, there are other kinds of sensors than those considered here; for example, discrete sensors like one denoting whether or not a door is open, or multivalued sensors like one that returns the altitude and azimuth of an airplane. We are extending the material in this paper to accommodate these more general sensors.

Acknowledgements This work profited from several discussions the author had with Jacob Aizikowitz, Fred Schneider and Sam Toueg. In addition, Özalp Babaoğlu, Ken Birman, Amitabh Shah, and Mark Wood read and commented on earlier versions of this paper. The criticisms and suggestions by the reviewers contributed significantly to the coherence of the paper.

A Proofs

The four theorems in this appendix give upper and lower bounds of $|\cap_{f,n}(\mathcal{S})|$. We need the following two definitions:

Definition 1 *If $\mathcal{S}$ is a set of intervals, a c-clique of $\mathcal{S}$ is a subset $\mathcal{S}'$ of $\mathcal{S}$ where $|\mathcal{S}'| = c$ and all the intervals in $\mathcal{S}'$ mutually intersect.*

Definition 2 *A set of intervals is c-reduced if each interval in S is a member of a c-clique.*

Note that a graph is $(n-f)$-reduced if and only if Algorithm 2 computes the empty set.

The upper and lower bounds of $|\cap_{f,n}(\mathcal{S})|$ are as follows. Theorem 1 is the same as Theorem 3; it is repeated here for clarity:

Theorem 3 *Let $\mathcal{S}$ be a set consisting of n intervals. If $0 \leq f < \lfloor (n+1)/2 \rfloor$ and $\cap_{f,n}(\mathcal{S}) \neq \emptyset$, then $|\cap_{f,n}(\mathcal{S})| \leq \mathbf{min}_{2f+1}\{|\bar{s}| : \bar{s} \in \mathcal{S}\}$.*

Theorem 4 *Given a set $\{\ell_1, \ell_2, ..., \ell_n\}$ of n lengths and $n > f \geq \lfloor (n+1)/2 \rfloor$, then for any length $\Lambda \geq \mathbf{min}\{\ell_1, \ell_2, ..., \ell_n\}$, there exists a set of n intervals $\mathcal{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_n\}$ where $\forall i : i \leq i \leq n : |\bar{s}_i| = \ell_i$ and $|\cap_{f,n}(\mathcal{S})| = \Lambda$.*

Theorem 5 *Let $\mathcal{S}$ be a $(n-f)$-reduced set of n intervals. If $n > f \geq \lfloor n/2 \rfloor$ and $\cap_{f,n}(\mathcal{S}) \neq \emptyset$, then $|\cap_{f,n}(\mathcal{S})| \geq \mathbf{max}_{2(n-f)-1}\{|\bar{s}| : \bar{s} \in \mathcal{S}\}$.*

Theorem 6 *Given a set $\{\ell_1, \ell_2, ..., \ell_n\}$ of n lengths, an arbitrarily small length ϵ, and $0 \leq f < \lfloor n/2 \rfloor$, there exists a $(n-f)$-reduced set of n intervals $\mathcal{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_n\}$ where $\forall i : i \leq i \leq n : |\bar{s}_i| = \ell_i$ and $|\cap_{f,n}(\mathcal{S})| = \epsilon$.*

Theorem 4 can be shown by construction. Let $\mathcal{S}$ consist of the following two cliques:

- C_1 containing $n - f$ intervals, where each interval in this clique has a minimum value of u, and by definition of Λ, a maximum value no larger than $u + \Lambda$;

- C_2 containing f intervals, where each interval in this clique has a maximum value of $u + \Lambda$, and by definition of Λ, a minimum value no smaller than u.

By hypothesis, $\lceil n/2 \rceil = \lfloor (n+1)/2 \rfloor \leq f < n$, or $2f \geq 2\lceil n/2 \rceil \geq n$ and so $f \geq n - f$ meaning both cliques are contained in $\cap_{f,n}(\mathcal{S})$. So, $\cap_{f,n}(\mathcal{S}) \ = [u \mathinner{..} u + \Lambda]$ and the theorem follows. □

Theorem 6 can also be shown by construction. Let $\mathcal{S}$ consist of two cliques:

- C_1 containing $\lfloor n/2 \rfloor$ intervals such that $\lfloor (n-f)/2 \rfloor$ intervals have a maximum value of $u+\epsilon$ and the remaining $\lfloor n/2 \rfloor - \lfloor (n-f)/2 \rfloor$ intervals have a maximum value less than u;

- C_2 containing $\lceil n/2 \rceil$ intervals such that $\lceil (n-f)/2 \rceil$ intervals have a minimum value of u and the remaining $\lceil n/2 \rceil - \lceil (n-f)/2 \rceil$ intervals have a minimum value greater than $u + \epsilon$.

By hypothesis, $0 \leq f < \lfloor n/2 \rfloor$ or $\lfloor n/2 \rfloor \leq \lceil n/2 \rceil < n - f \leq n$, and so neither C_1 nor C_2 are entirely in $\cap_{f,n}(\mathcal{S})$. However, $n - f$ intervals intersect over the interval $[u \mathinner{..} u + \epsilon]$, and the theorem follows. □

To prove theorems 3 and 5, we will need a few lemmas.

Lemma 1 *Let $\mathcal{S}$ be a set of n intervals where $\mathcal{S}$ contains at least one c-clique and all c-cliques in $\mathcal{S}$ have exactly i intervals in common with each other. Then, $n \geq c \geq i$* and *$n \geq 2c - i$.*

Proof: since $\mathcal{S}$ contains at least one c–clique, we know $n \geq c$. Furthermore, since all c–cliques in $\mathcal{S}$ have exactly i intervals in common, each c–clique must have at least i intervals, or $c \geq i$.

If $c = i$, then the smallest graph satisfying our assumptions is a single i–clique, or $n = i = 2c - i$. If $c > i$, then $\mathcal{S}$ must contain more than one c–clique, for otherwise the single c–clique has $c > i$ intervals in common with itself. The smallest such set of intervals consists of two c–cliques sharing i intervals. Each clique has $c - i$ intervals not in common with each other, or $n = i + 2(c - i) = 2c - i$. □

Lemma 2 *Let S be a set of n intervals where S contains at least one c–clique. If $n < 2c$, then all c–cliques in S have at least $2c - n$ intervals in common with each other.*

Proof: by contradiction. Suppose that all the c–cliques in S have exactly i' intervals in common with each other, where $i' < 2c - n$. By lemma 1, S contains at least $2c - i'$ intervals, or $n \geq 2c - i'$. Rearranging the last inequality, we get $i' \geq 2c - n$, which contradicts our hypothesis. □

Lemma 3 *Let $\overline{s} \in S$ be any member of all maximal cliques of S. The cover of the intersection of the maximal cliques is no larger than $|\overline{s}|$.*

Proof: The intersection of any maximal clique cannot contain any point outside of $\overline{s}$, since by definition that point is not in an intersection containing $\overline{s}$ and $\overline{s}$ is a member of each clique. The cover only adds points between the intersections. Since S is a set of intervals over the reals, $\overline{s}$ must contain all points between the maximal cliques, so the cover does not add any points in $\overline{s}$. Since all the points in the cover are also in $\overline{s}$, the cover cannot be larger than $|\overline{s}|$. □

Theorem 3 can now be shown. From the definition of $\cap_{f,n}(S)$, the maximal clique in S must contain at least $n - f$ intervals, for otherwise $\cap_{f,n}(S) = \emptyset$. By assumption, $f \leq \lfloor (n+1)/2 \rfloor$ or $n < 2(n - f)$. By lemma 2, at least $n - 2f$ intervals intersect all cliques. By lemma 3 the cover of the intersection cannot be larger than any of these $n - 2f$ intervals. The cover, however, may be larger than any of the remaining $2f$ intervals. In the worst case, these remaining intervals are the smallest ones in S, and the theorem follows. □

Lemma 4 *Let S be a c-reduced set of n intervals, and let the intervals $\overline{s}_i$ in S be ordered such that* $\min \overline{s}_i \leq \min \overline{s}_j$ *if $i < j$. Then, the intervals $\overline{s}_1, \overline{s}_2, \ldots \overline{s}_c$ form a c–clique.*

Proof: by induction. The lemma is trivially true for $c = 1$ since any interval is by itself a 1–clique. So, we assume the lemma holds for $c = k$ and show that it holds for $c = k + 1$. Let S be a $(k + 1)$–reduced set of intervals. If a set is $(k + 1)$–reduced then it is k–reduced, so by the induction hypothesis the intervals $\overline{s}_1, \overline{s}_2, \ldots \overline{s}_k$ form a k–clique. If $\overline{s}_{k+1}$ does not intersect some interval $\overline{s_i} : 1 \leq i \leq k$, then all intervals $\overline{s}_j : j > k + 1$ also do not intersect $\overline{s}_i$, and so $\overline{s}_i$ is not a member of a $(k + 1)$–clique. This contradicts our assumption that $\mathcal{S}$ is $(k+1)$-reduced, and so $\bar{s}_{k+1}$ must intersect each interval $\bar{s}_1, \bar{s}_2, \ldots \bar{s}_k$, and the lemma holds. □

The same argument can be used to prove the following lemma:

Lemma 5 *Let $\mathcal{S}$ be a c-reduced set of n intervals, and let the intervals $\bar{s}_i$ in $\mathcal{S}$ be ordered such that* $\mathbf{max}\bar{s}_i \geq \mathbf{max}\ \bar{s}_j$ *if $i < j$. Then, the intervals $\bar{s}_1, \bar{s}_2, \ldots \bar{s}_c$ form a c-clique.*

Lemma 6 *If $\mathcal{S}$ is a $(n-f)$-reduced set of n intervals, then*

$$\cap_{f,n}(\mathcal{S}) \ = [\mathbf{min}_{n-f+1}\{\mathbf{min}\ \bar{s} : s \in \mathcal{S}\} \ .. \ \mathbf{max}_{n-f+1}\{\mathbf{max}\ \bar{s} : s \in \mathcal{S}\}]$$

Proof: this lemma follows directly from Lemma 4, Lemma 5 and the definition of $\cap_{f,n}(\mathcal{S})$. □

Theorem 5 can now be shown. From Lemma 6, all intervals intersect $\cap_{f,n}(\mathcal{S})$ and there are exactly $2(n-f-1)$ (not necessarily distinct) intervals that extend outside of $\cap_{f,n}(\mathcal{S})$. This means that there are at least $n-2(n-f-1) = 2(f+1)-n$ intervals that are completely contained by $\cap_{f,n}(\mathcal{S})$. So, $|\cap_{f,n}(\mathcal{S})| \geq \mathbf{min}_{2(f+1)-n}\{\bar{s} : \bar{s} \in \mathcal{S}\}$ or $|\cap_{f,n}(\mathcal{S})| \geq \mathbf{max}_{2(n-f)-1}\{\bar{s} : \bar{s} \in \mathcal{S}\}$. □

B Algorithms for Computing $\cap_{f,n}(\mathcal{S})$

This section contains some algorithms for computing $\cap_{f,n}(\mathcal{S})$. A set of abstract sensors are isomorphic to a class of graphs called *interval graphs*, which in turn are members of the class of *triangulated graphs*. Such graphs are interesting in that many problems, such as coloring, clique, stable set and clique cover can be solved for triangulated graphs in polynomial time. A good reference on triangulated graphs is [6], which includes efficient algorithms that solve the above problems.

The value of $\cap_{f,n}(\mathcal{S})$ is $[l \ .. \ h]$ where l is the smallest point contained in $n-f$ intervals and h is the largest point contained in $n-f$ intervals, and where a point x is contained in an interval $\bar{s}$ if and only if $\mathbf{min}\ \bar{s} \leq x \leq \mathbf{max}\ \bar{s}$. Suppose that there are a intervals $\bar{s}$ in $\mathcal{S}$ such that $\mathbf{min}\ \bar{s} \leq x$ and that there are b intervals $\bar{s}'$ in $\mathcal{S}$ such that $\mathbf{max}\ \bar{s}' < x$. Any interval not counted in a cannot contain x, and the intervals counted in b are those that were counted in a but cannot contain x, so x is contained in exactly $a-b$ intervals.

Let v be an array of $2n$ pairs where for each $\overline{s}_i \in \mathcal{S}$, $v_{2i} = (\mathbf{min}\ \overline{s}_i, 1)$ and $v_{2i+1} = (\mathbf{max}\ \overline{s}_i, -1)$. Given a point x,

$$a = \sum_{\substack{\forall i: v_i[1] \leq x \\ \wedge v_i[2]=1}} v_i[2]$$

and

$$b = -\sum_{\substack{\forall i: v_i[1] < x \\ \wedge v_i[2]=-1}} v_i[2]$$

or

$$\text{number of } \overline{s} \in \mathcal{S} \text{ containing } x = a - b = \sum_{\forall i: v_i[1] < x} v_i[2] \quad + \sum_{\substack{\forall i: v_i[1] = x \\ \wedge v_i[2]=1}} v_i[2]$$

Computing the number of intervals in $\mathcal{S}$ that contain x can be made linear if v is sorted. Define $v_i < v_j = (v_i[1] < v_j[1]) \vee (v_i[1] = v_j[1] \wedge v_i[2] > v_j[2])$, and let v' be v sorted with respect to $<$. Then,

$$\text{number of } \overline{s} \in \mathcal{S} \text{ containing } x = \sum_{i=0}^{\substack{\mathbf{max}\ j: v_j[1] \leq x \wedge \\ (v_j[1]=x) \Rightarrow (v_j[2]=1)}} v_i[2] \qquad (1)$$

Recall that l is the smallest point contained in $n - f$ intervals. Thus, l is the smallest x that makes Equation 1 equal to $n - f$, which is $v'_{low}[1]$ where

$$low = \mathbf{min}\ j : \sum_{i=0}^{j} v'_i[2] = (n - f)$$

Similarly, h is the largest point contained in $n - f$ intervals, which is the largest x that makes Equation 1 equal to $n - f$. This point is also the maximum value of some interval such that all points greater than x are contained in no more than $n - f - 1$ intervals, or h is $v'_{high}[1]$ where

$$high = \mathbf{max}\ j : \sum_{i=0}^{j} v'_i[2] = (n - f - 1)$$

Both *low* and *high* can be computed from v' in $O(n)$ time, and v' can be computed from v in $O(n \log n)$ time, so the overall running time is $O(n \log n)$.

There are two cases for which $\cap_{f,n}(\mathcal{S})$ can be calculated faster than $O(n \log n)$:

1. $\cap_{n-1,n}(\mathcal{S})$ is the cover of $\mathcal{S}$, or l is the smallest minimum value of the intervals and h is the largest maximum value of the intervals. For our purposes, however, this case is not very interesting.

2. If all of the intervals in $\mathcal{S}$ mutually intersect, then all of the minimum values of these intervals are less than or equal to the smallest maximum value of these intervals (this can be tested for in $O(n)$ time). Under this condition, the array v' consists of all of the minimum values (having $v_i[2] = 1$) followed by the maximum values (having $v_i[2] = -1$). Thus, l is the $f + 1^{st}$ largest minimum and h is the $f + 1^{st}$ smallest maximum, both which can be calculated in $O(n)$ time [1]. If $f = 0$ or a fail-stop failure model is assumed, then we are interested in the value of $\cap_{0,n}(\mathcal{S})$, which requires that all intervals mutually intersect and can be calculated trivially in $O(n)$ time.

C Train Length

In the example of Section 4, we assumed the train had zero length. This is not an unreasonable assumption, since we can show that for every train of length L on a track K, there exists a track K' such that a zero-length train is constrained in exactly the same way as the original train on K. In this section, we show how to determine the track K' from L and K. The method is an example of ***transforming to configuration space*** [11].

A track K is defined by three sets $(\forall i : 1 \leq i \leq n : \{c_i\}, \{min_i\}, \{max_i\})$ where c_i is the location of the end of track segment i, min_i is the minimum allowable speed on segment i and max_i is the maximum allowable speed on segment i. If the train has length L and the tail of the train is at x, the safety condition is that ***all*** parts of the train satisfy the speed constraints, or

$$S_L(x, v) \overset{\text{def}}{=}$$
$$\forall x', i : x \leq x' \leq x + L, 1 \leq i < n : c_i \leq x' \leq c_{i+1} \Rightarrow min_i \leq v \leq max_i$$

Suppose we could find a track $K' : (\forall i : 1 \leq i \leq n' : \{c'_i\}, \{min'_i\}, \{max'_i\})$ such that

$$S_L(x, v) \;\equiv\; (S_0(x, v) \overset{\text{def}}{=} \forall i : 1 \leq i < n' : c'_i \leq x \leq c'_{i+1} \Rightarrow min'_i \leq v \leq max'_i)$$

S_0 is the safety condition for a zero-length train on track K' which is constrained in exactly the same way an L-length train on the original track is constrained. If we can find K' then we can write a program that controls a zero-length train on K', and this program will also control the L-length train on K.

Define the two functions

$$\mathrm{Min}(L, x) \overset{\text{def}}{=} \forall j : x \leq c_j \leq x + L : \mathbf{max}\ min_j$$
$$\mathrm{Max}(L, x) \overset{\text{def}}{=} \forall j : x \leq c_j \leq x + L : \mathbf{min}\ max_j$$

These functions determine the actual speed bounds the train must follow when at X. With them, S_L can be rewritten as $S_L : \mathrm{Min}(L, x) \leq v \leq \mathrm{Max}(L, x)$.

We can now find the values of K' that allow $S_L(x, v)$ to be rewritten as $S_0(x, v)$. Both $\mathrm{Min}(L, x)$ and $\mathrm{Max}(L, x)$ are piecewise constant functions, so we can define the track segments of K' to be the spans where both $\mathrm{Min}(L, x)$ and $\mathrm{Max}(L, x)$ are constant. Let c'_i be the union of the points of inflection of $\mathrm{Min}(L, x)$ and $\mathrm{Max}(L, x)$, and let

$$min'_i = \lim_{\delta \to +0} \mathrm{Min}(L, c_i + \delta)$$
$$max'_i = \lim_{\delta \to +0} \mathrm{Max}(L, c_i + \delta)$$

Figure 11 shows an example of K' given K and L. Each track segment is drawn with the maximum speed above the segment and the minimum speed below the segment. Note K' is shorter than K by L, since the end of the train cannot traverse the whole length of K without the train leaving K. Here, K and K' have the same number of segments; in general, K' can have up to twice as many segments as K.

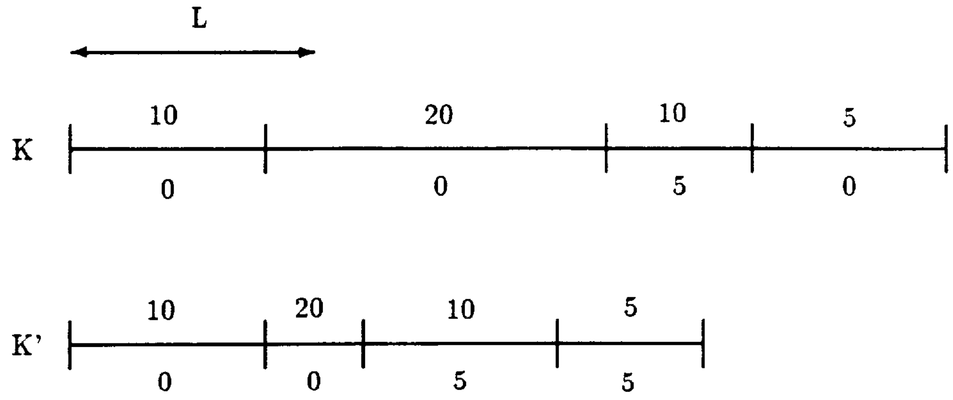

Figure 11: Configuration Space

References

[1] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1972.

[2] Flaviu Cristian, Houtan Aghili, and Ray Strong. Atomic broadcast: From simple message diffusion to byzantine agreement. Technical Report RJ 5244 (54244), IBM Almaden Research Laboratory, July 1986.

[3] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, 1986.

[4] Alan D. Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4:9–29, 1990.

[5] M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). Technical Report DCS/RR-273, Yale University, June 1983.

[6] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[7] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE–12(9), September 1986.

[8] J. Kearns, S. Park, and Sjogren J. Data editing: Faster convergence for synchronous approximate agreement. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, pages 393–4401. IEEE Computer Society, June 1988.

[9] R. Koymans, R. Kuiper, and E. Zijlstra. Paradigms for real-time systems. In G. Goos and J. Hartmanis, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes on Computer Science*, pages 159–174. Springer-Verlag, 1988.

[10] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[11] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, (C–32):108–120, 1983.

[12] I. M. MacLeod. Data consistency in sensor–based distributed computer control systems. In *Proceedings of the Fourth International Conference on Distributed Computing Systems*, pages 440–446. IEEE Computer Society, May 1984.

[13] Steve Maheney and Fred Schneider. Inexact agreement: Accuracy, precision, and graceful degredation. In *Proceedings of the Fourth Symposium on Principles of Distributed Computing*, pages 237–249. ACM SIGACT/SIGOPS, August 1985.

[14] Keith Marzullo. *Maintaining the Time in a Distributed System.* PhD thesis, Stanford University, Department of Electrical Engineering, June 1984.

[15] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *Proceedings of the Second Symposium on Principles of Distributed Computing*, pages 295–305. ACM SIGPLAN/SIGOPS, 1983.

[16] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed systems. In *Proceedings of the Eighth Symposium*

on Principles of Distributed Computing, pages 248–262. ACM SIG-PLAN/SIGOPS, August 1988.

[17] Fred B. Schneider. Byzntine generals in action: Implementing fail-stop processors. *ACM Transactions on Computer Systems*, 2(2):145–154, May 1984.

[18] Fred B. Schneider. The state machine approach: A tutorial. *Computing Surveys*, 22(3), September 1990.

[19] A. Udaya Shankar and Simon S. Lam. Time-dependent distributed systems: Proving safety, liveness and real-time properties. *Distributed Computing*, pages 61–79, 1987.

[20] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. Mc-Marthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.