SRI International

CSL Technical Report • January 2002

Formal Verification of Marzullo's Sensor Fusion Interval

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

This research was supported by NASA Langley Research Center under contract NAS1-20334 and Cooperative Agreement NCC-1-377 with Honeywell Tucson, and by the DARPA MoBIES program under contract F33615-00-C-1700 with US Air Force Research Laboratory.

Abstract

We examine the problem of selecting a best value from a collection of sensor readings, and diagnosing faulty readings in such a collection. We focus on sensor interfaces that return a *range* of values and describe the "fusion functions" $\bigcap_{f,n}(S)$ of Marzullo and $\mathcal{F}_n^f(S)$ of Schmid and Schossmaier. We use PVS formally to prove the soundness of $\bigcap_{f,n}(S)$ (i.e., it always contains the correct value), from which soundness of $\mathcal{F}_n^f(S)$ also follows. $\mathcal{F}_n^f(S)$ is generally to be preferred to $\bigcap_{f,n}(S)$ because it satisfies a "Lipschitz Condition" (small changes in sensor readings produce small changes in its output), and is optimal among all such functions.

Contents

List of Figures

Chapter 1

Introduction

We are interested in application-independent fault tolerance for embedded systems and, in particular, in methods for sensor validation and fusion. We can consider the problem of sensor *validation* to be concerned with deciding whether the reading returned by a particular sensor is trustworthy or not, and the problem of sensor *fusion* to be that of inferring a "best" value for the sampled variable from readings returned by multiple sensors. Validation can be performed on each sensor reading in isolation (e.g., using diagnostic information returned by the sensor itself), or by comparing the reading of each sensor against those of others, or both these methods can be used in combination. Modern "smart" sensors have their own processors and do perform validation locally. However, we also want the local validation process to return information that is useful for validation by comparison, and for the fusion process. One particularly useful and application-independent way in which this can be accomplished is for sensors to return a *pair* of values (representing lower and upper bounds) rather than a single value: sensors that locally diagnose difficulties will return a large range, while those that are confident of their performance will return a tight one. A refinement, consistent with the *temporal firewall* proposal of Kopetz [KN97], is to return a pair of parameters representing the upper and lower bounds as *functions* of time rather than constants: the bounds will then move further apart as the delay increases between reading and use of the sensor value. The sensor fusion task now becomes one of combining the ranges returned by individual sensors to achieve the best "consensus" value. Even smart sensors can fail, however, so the fusion task must be resilient to some number of completely erroneous readings.

The abstract sensor fusion problem is portrayed in Figure 1.1: readings from four sensors are shown as lines representing the intervals between their upper and lower bounds, and the problem is to extract a new interval that represents a good estimate of the true value of the sampled variable, that is resistant to faulty readings, and that has other desirable properties.

If we assume that a property of nonfaulty sensors is that the true value of the sampled variable must lie within the interval that they return, then the intervals returned by nonfaulty

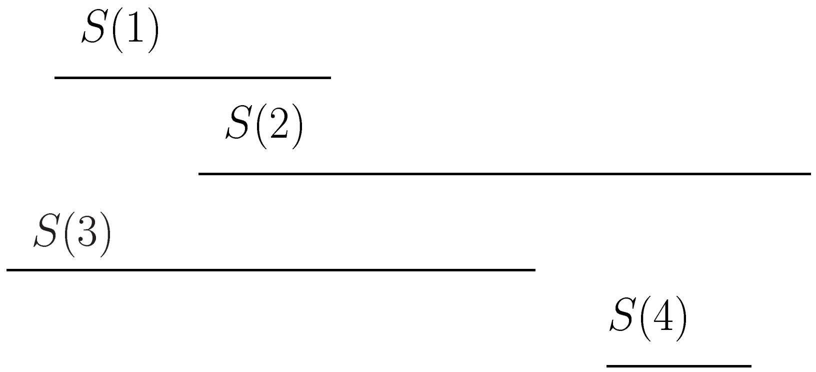

Figure 1.1: Sensor Fusion Problem

sensors must overlap. In Figure 1.1, it is therefore clear that one or both of $S(3)$ and $S(4)$ must be faulty, and similarly for $S(1)$ and $S(4)$. If we further assume that only one fault may arise at a time, then we may conclude that $S(4)$ must be faulty and that the best estimate of the sampled variable is the interval of mutual overlap between $S(1)$, $S(2)$, and $S(3)$ indicated by the vertical dashed lines shown in Figure 1.2.

But now suppose that the interval for $S(4)$ shifted a little to the left as shown in Figure 1.3. It is still clear that (under a single-fault hypothesis) one of $S(1)$ and $S(4)$ must be faulty (because their intervals do not overlap), but the information available provides no basis for preferring one over the other. If $S(4)$ is faulty, then the appropriate "fusion" interval is the one between the two dashed lines on the left, whereas if $S(1)$ is faulty, the fusion interval should be the one between the two dashed lines on the right. But since we have no basis for choosing one over the other, the only safe choice is for the fusion interval to embrace both, and to extend from the leftmost dashed line to the rightmost as shown in Figure 1.4.

This example illustrates that precise diagnosis of sensor faults is not always possible when performing fusion, and that it may be necessary to sacrifice precision (i.e., to choose a wider interval) in order to maintain soundness (the property that the actual value is contained in the selected interval). The example also illustrates a second issue: if we imagine the interval for $S(4)$ gradually shifting left from its position in Figure 1.2 to its position in Figure 1.4, we see that the fusion interval will change abruptly from that indicated by the dashed lines in the first of these figures to that shown in the last. This abrupt change could cause a 'thump" to be passed through the control laws to the actuators, with consequent adverse effects on the behavior, or even the safety, of the controlled plant.

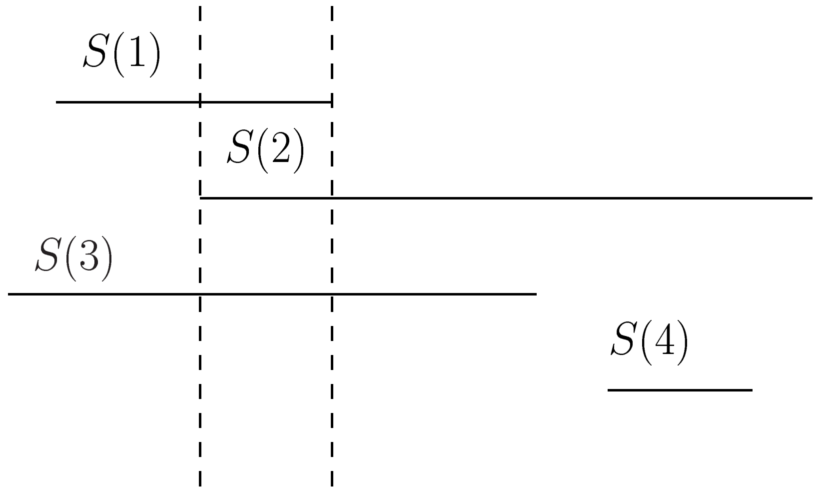

Figure 1.2: Resulting Fusion Interval

In the next chapter, we will consider two fusion functions that guarantee soundness of their fusion intervals, and will then examine the extent to which they satisfy a "Lipschitz Condition" and avoid the abrupt change illustrated in the example above.

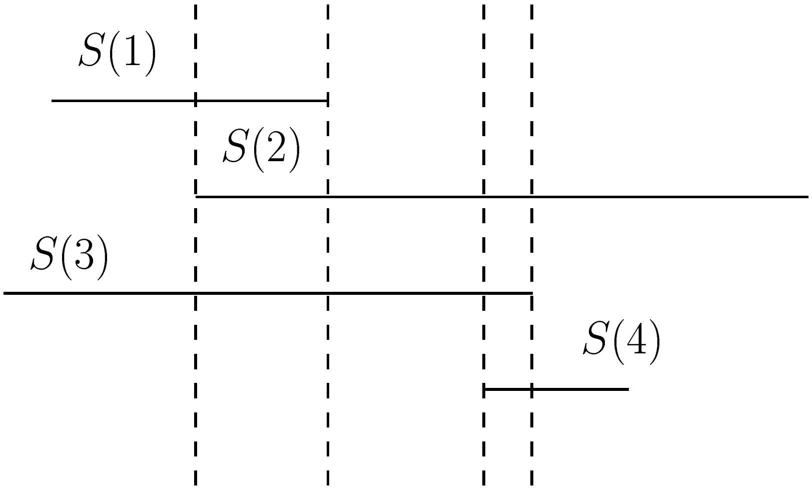

Figure 1.3: Sensor Fusion Problem with Unclear Diagnosis

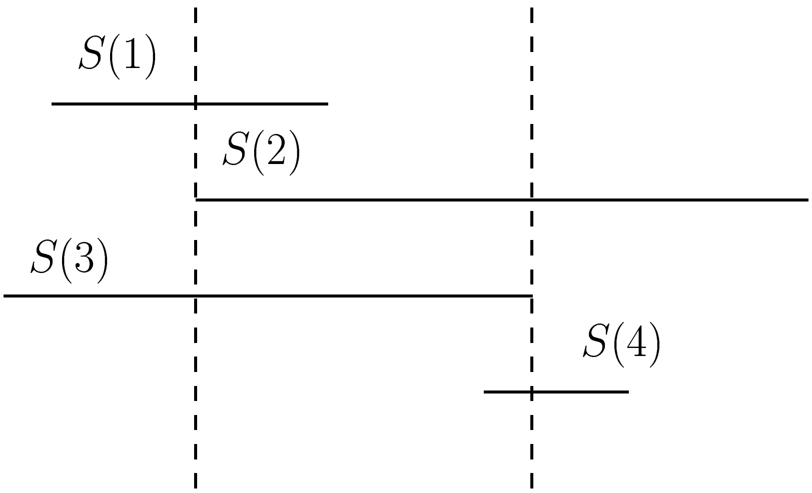

Figure 1.4: Precision Sacrificed to Preserve Soundness

Chapter 2

Formalization of Sensor Fusion Functions

We assume a collection S of n sensor readings that are intended to sample some physical variable V at a time t. The reading of the j'th sensor is given by $S(j)$ and consists of a pair representing an interval; the lower bound of the interval is denoted $S(j)_{lo}$ and the upper by $S(j)_{hi}$. Some sensors may be *faulty*; the maximum number of faulty sensors is denoted f (where $f < n$). Nonfaulty sensors satisfy the following property.

Definition 1 (Accuracy) *If j is a nonfaulty sensor, then $S(j)_{lo} \leq V \leq S(j)_{hi}$.*

Marzullo [Mar90] introduces the sensor fusion interval $\bigcap_{f,n}(S)$ as the smallest interval that is guaranteed to contain the correct value V. It is defined as follows.

Definition 2 (Marzullo fusion interval) *Let l be the smallest value contained in at least $n-f$ of the intervals $S(j), j = 1, \ldots, n$, and let h be the largest value contained in at least $n-f$ such intervals. Then $\bigcap_{f,n}(S)$ is the interval $[l \ldots h]$.*

Marzullo's interval can be computed as follows. First we sort the lower and upper bounds of all the sensor readings into ascending order (this takes $O(n \log n)$ time). Then we scan the sorted list from smallest to largest, maintaining an intersection count (initially zero): this increments by one for every lower bound and decrements by one for every upper bound; the lower bound l of the fusion interval is the first value where the count reaches $n-f$. The upper bound h is similarly found *mutatis mutandis*, by scanning in the opposite direction.

If we take $n = 4$ and $f = 1$, then Marzullo's interval for the example shown in Figure 1.1 is the interval between the dashed lines in Figure 1.2. When the interval corresponding to the sensor $S(4)$ shifts a little to the left, Marzullo's interval becomes that shown in Figure 1.4.

There are two essential properties of any fusion interval: it must always be *defined*, and it must be *sound*: that is, the actual value must be contained in the fusion interval. We prove these properties for Marzullo's function (Marzullo's paper [Mar90] does not make these results explicit).

Theorem 1 (Marzullo's interval is always defined) *For any collection of sensors S, the interval $\bigcap_{f,n}(S)$ is well defined provided the number of faulty sensors does not exceed f.*

Proof: By the assumption of accuracy, the correct value V lies within the interval of each nonfaulty sensor. Since there are at least $n - f$ nonfaulty sensors, there is certainly *some* $l \leq v$ and $h \geq v$ that lie in the intersection of $n - f$ intervals, and the construction of $\bigcap_{f,n}(S)$, which simply selects the least such l and greatest such h is well-defined. □

Theorem 2 (Marzullo's interval is sound) *For any collection of sensors S, the interval $\bigcap_{f,n}(S)$ contains the true value V, provided the number of faulty sensors does not exceed f.*

Proof: By the assumption of accuracy, the correct value V lies within the interval of each nonfaulty sensor. Since there are at least $n - f$ nonfaulty sensors, the construction of $\bigcap_{f,n}(S)$ must include V □

Observe that $\bigcap_{0,n}(S)$ is the "intersection" of all the readings in S, $\bigcap_{n-1,n}(S)$ is their "union," and $\bigcap_{f,n}(S)$ is contained in $\bigcap_{f',n}(S)$ when $f < f'$.

We denote the accuracy of a sensor reading $S(j)$ by $|S(j)| \stackrel{\text{def}}{=} S(j)_{hi} - S(j)_{lo}$; the accuracy of the fusion interval $\bigcap_{f,n}(S) = [l \ldots h]$ is similarly defined as the difference $h - l$.

The accuracy of the fusion interval is clearly related to the accuracy of the readings provided by nonfaulty sensors, and to the number and kind of faulty readings. The following results are proved in [Mar90].

When there are many faults, we cannot expect great accuracy of the fusion interval, Specifically, if at least half the sensors are faulty, the fusion interval may be less accurate than any sensor reading, and cannot be more accurate than the most accurate reading.

When fewer than half the sensors are faulty, however, the fusion interval may be more accurate than any sensor, and is at least as accurate as some sensor.

Theorem 3 (Theorem 1 of [Mar90]) *If fewer than half the sensors are faulty (i.e., $n \geq 2f + 1$), then $|\bigcap_{f,n}(S)| \leq \min_{2f+1}\{|S(j)|\}$, where $\min_r$ denotes the r-smallest (i.e., $n - r$-biggest) interval in S.*

Thus, when fewer than half the sensors are faulty, the accuracy of the fusion interval is at least as good as that of some sensor. However, that sensor (i.e., $S(i)$ where $|S(I)| =$

$\min_{2f+1}\{|S(j)|\}$) could be faulty and therefore arbitrarily inaccurate, so the result is not particularly helpful. When fewer than a third of the sensors are faulty, however, the accuracy of the fusion interval is bounded by that of some good sensor.

Theorem 4 (Theorem 2 of [Mar90]) *If fewer than a third of the sensors are faulty, then* $|\bigcap_{f,n}(S)| \leq \min_{f+1}\{|C(j)|\}$*, where* C *is the collection of nonfaulty sensors.*

The results above concern arbitrary failures: those where the sensor returns bad readings that are not indicated or detectable as such. Faulty sensors that are diagnosed by their local validation mechanisms can indicate this fact to the fusion algorithm, thereby making their failures detectable. In addition, sensors whose readings do not intersect $\bigcap_{f,n}(S)$ cannot be correct and may thereby be detected as faulty by the fusion algorithm. When the failure of a sensor is detectable, it can be removed from the collection of sensors, thereby reducing the values of both n and f by one and improving the ratio of n to f.

As noted earlier, for the example shown in Figure 1.1, $\bigcap_{1,4}(S)$ is the interval between the dashed lines shown in Figure 1.2. When the interval corresponding to the sensor $S(4)$ shifts a little to the left, $\bigcap_{1,4}(S)$ jumps to the interval shown in Figure 1.4 as soon as $S(4)$ overlaps $S(3)$. Lamport [Lam87] noted this undesirable property and suggested that a good fusion function should satisfy a *Lipschitz Condition*,[1] meaning that small changes in the readings of individual sensors should change the fusion interval by a correspondingly small amount. Lamport considered some plausible modifications to $\bigcap_{1,4}(S)$ but the solutions that he found operate on point, rather than interval, sensor readings, and therefore do not make full use of the available information (an example is the fault-tolerant midpoint used in some clock synchronization algorithms [WL88]).

An interval-based fusion method that does satisfy the Lipschitz Condition has recently been introduced by Schmid and Schossmaier [SS01]. Their Fault-Tolerant Interval (FTI) fusion function $\mathcal{F}_n^f(S)$ is defined as follows.

Definition 3 (FTI fusion interval) *Let* $l = \max_{f+1}\{S(j)_{lo}\}$ *be the* $f+1$*'st largest of the lower bounds on the collection of sensor readings* S *and let* $h = \min_{f+1}\{S(j)_{hi}\}$ *be the* $f+1$*'st smallest upper bound. Then* $\mathcal{F}_n^f(S)$ *is the interval* $[l \ldots h]$.

If we take $n = 4$ and $f = 1$, then the FTI interval for the example shown in Figure 1.1 is the interval between the dashed lines in Figure 2.1; this should be compared to the corresponding interval for $\bigcap_{1,4}(S)$ shown in Figure 1.2. When the interval corresponding to the sensor $S(4)$ shifts a little to the left, the FTI interval is unchanged, whereas $\bigcap_{1,4}(S)$ jumps to that shown in Figure 1.4.

[1]The classical Lipschitz Condition is an inequality that guarantees a unique solution to the differential equation $y' = f(x, y)$. In the context considered here, it refers to the requirement that small changes in the arguments should result in small changes to its value, for a suitably defined metric.

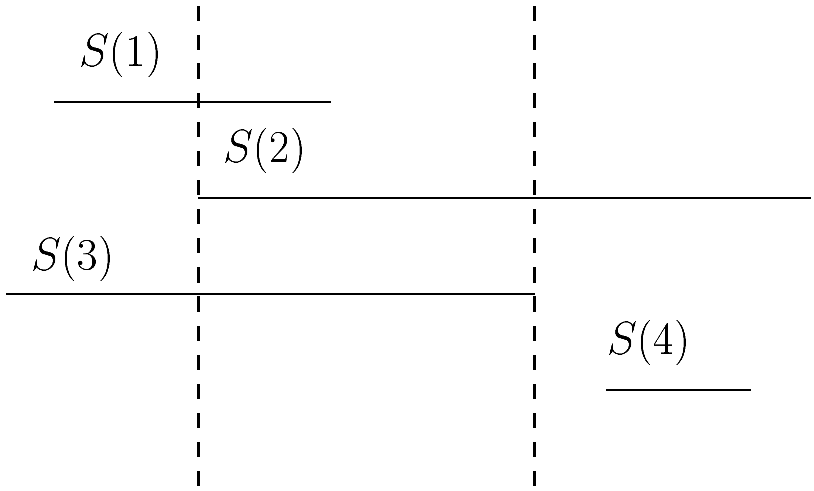

Figure 2.1: FTI Fusion Interval

Relationships between $\bigcap_{f,n}(S)$ and $\mathcal{F}_n^f(S)$ are established in the following theorem.

Theorem 5 (Lemma 2 of [SS01])

1. $\mathcal{F}_n^f(S) \supseteq \bigcap_{f,n}(S)$
2. $\mathcal{F}_n^f(S) = \bigcap_{f,n}(S)$ *if there is no* $S(j)$ *disjoint from* $\bigcap_{f,n}(S)$
3. $\mathcal{F}_n^{n-1}(S) = \bigcap_{f-1,n}(S)$
4. $\mathcal{F}_n^0(S) = \bigcap_{0,n}(S)$

Notice that Result 2 in the list above does not mean that Marzullo's method plus diagnosis and exclusion of any $S(j)$ disjoint from $\bigcap_{f,n}(S)$ produces the same result as $\mathcal{F}_n^f(S)$. Certainly, as noted on Page 7, any $S(j)$ disjoint from $\bigcap_{f,n}(S)$ must be faulty: in the case of Figure 1.1, this means we could discard $S(4)$ and calculate $\bigcap_{1,3}(S\backslash\{S(4)\})$. But the result would be that shown in Figure 2.2, which is not the same as $\mathcal{F}_4^1(S)$ of Figure 2.1. (It is the same as $\mathcal{F}_3^1(S\backslash\{S(4)\})$, as required by the theorem.)

Result 1 in the list above serves to establish soundness of the FTI construction.

Theorem 6 (The FTI interval is sound) *For any collection of sensors S, the interval $\mathcal{F}_n^f(S)$ contains the true value V, provided the number of faulty sensors does not exceed f.*

Proof: This follows from the soundness of $\bigcap_{f,n}(S)$ and the inclusion established in result 1 of the previous Theorem. □

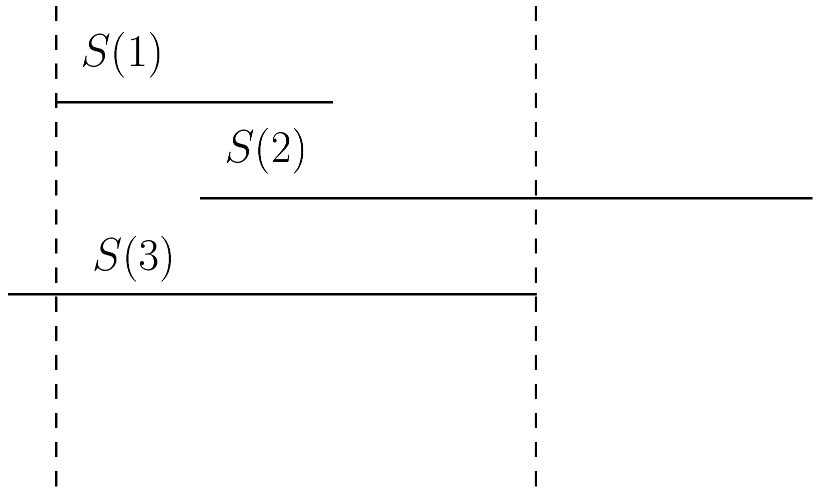

Figure 2.2: Marzullo's Fusion Interval with Diagnosis

Schmid and Schossmaier [SS01] establish that their function $\mathcal{F}_n^f(S)$ does satisfy the Lipschitz Condition, and that it is optimal (i.e., the intervals produced by any fusion function that satisfies the Lipschitz Condition must include $\mathcal{F}_n^f(S)$).

Schmid and Schossmaier also establish bounds on the accuracy of $\mathcal{F}_n^f(S)$; they show that it has exactly the same worst-case behavior as $\bigcap_{f,n}(S)$, but may produce slightly less accurate results in non-worst-case scenarios.

2.1 Formal Specification and Verification in PVS

We present a formal specification and verification for some of the concepts introduced above. The formalization uses PVS [ORSvH95].

To begin, we define a theory parameterized by the constants `n` and `f`, where the former is required to be a positive natural number, and the latter is a natural number strictly less than `n` (notice this exploits PVS's support for *dependent typing*). We then define the `index` type that will identify sensors as the numeric range `1, ..., n`

```
sensors [n: posnat, f:below(n)]: THEORY
BEGIN

index: TYPE = subrange(1,n)

IMPORTING finite_sets@finite_sets_below,
          finite_sets@finite_sets_minmax,
          finite_sets@finite_sets

index_finite: LEMMA is_finite_type[index]
```

Next we import various theories from the standard `finite_sets` library, and state the lemma that `index` is a finite type. This requires us to exhibit an injection from `index` to some finite initial segment of the naturals: following the initial proof command `(EXPAND "is_finite_type")`, PVS presents the following sequent.

```
Rule? (EXPAND "is_finite_type")
Expanding the definition of is_finite_type,
this simplifies to:
index_finite :

  |-------
{1}   (EXISTS N, (g: [index -> below[N]]): injective?(g))
```

A suitable injection is supplied by the command `(INST + "n" "lambda (i:index):i-1")` and PVS then finishes off the proof after the additional command `(EXPAND "injective?")`.

Next, we specify the lemma `index_sets_finite`, which establishes that any set defined over the `index` type is finite. This lemma is needed to discharge numerous TCCs that arise during the development.

```
index_sets_finite: LEMMA forall (x:setof[index]): is_finite(x)
```

The proof of this result simply applies the following prelude lemma as a rewrite using the command `(REWRITE "finite_type_set[index]")`

```
finite_type_set:     LEMMA is_finite_type IMPLIES is_finite(S)
```

and then applies the previous lemma with the command `(USE "index_finite"))`.

The reason we need finite sets is to be able to use the cardinality function `card`. The first lemma using this function establishes that the cardinality of the full set of `index` is `n`.

```
card_index_fullset: LEMMA card(fullset[index]) = n
```

The proof of this result establishes a bijection to the first `n` natural numbers. The previous lemma is used to discharge the TCC generated by PVS to ensure that the construction yields a finite set.

```
(REWRITE "card_bij[index]")
(("1" (INST + "lambda (i:index):i-1")
   (("1" (GRIND)) ("2" (GRIND))))
 ("2" (REWRITE "index_sets_finite")))
```

Next, we define a sensor `sample` as a record consisting of a pair of real numbers with field identifiers `lo` and `hi`, respectively, where the latter is required to be no less than the former (dependent typing again). The `CONTAINING` annotation enables PVS to discharge the TCC to show this type is nonempty.

```
i,j: VAR index

sample: TYPE = [# lo: real, hi: { x:real | x>=lo } #]
           CONTAINING (# lo:=1, hi:=1 #)

sensor(i): sample

actual: real
```

We then define the collection of sensors as an array `sensors` of type `sample` (so that $S(i)_{lo}$ of the informal mathematical presentation corresponds to `S(i)‘lo` in PVS), and introduce the actual value of the sampled physical quantity as the real constant `actual` (cf. V in the informal presentation).

Then we define the `faulty` sensors as some uninterpreted set of `index`, and `ok` as its complement (the set of nonfaulty sensors).

```
faulty: finite_set[index]

ok: finite_set[index] = { i | NOT faulty(i) }
```

We specify axiomatically the assumptions that the readings of nonfaulty sensors contain the actual value, and that there are no more than `f` faulty sensors.

```
good_sensor: AXIOM ok(i) IMPLIES
        (sensor(i)‘lo <= actual AND sensor(i)‘hi >= actual)

max_faults: AXIOM card(faulty) <= f

min_good: LEMMA card(ok) >= n-f
```

We state the lemma that the number of nonfaulty sensors must be at least `n-f`. The proof uses the `card_diff` lemma from the `finite_sets` library, and the `card_index_fullset` lemma introduced above.

```
(EXPAND "ok")
(LEMMA "card_diff_subset[index]")
(INST - "faulty" "fullset[index]")
(("1"
  (GROUND)
  (("1"
    (CASE-REPLACE
     "difference(fullset[index], faulty)
           = { i | NOT faulty(i) }" :HIDE?  T)
    (("1" (USE "max_faults")
          (REWRITE "card_index_fullset")
          (ASSERT))
     ("2" (HIDE -1 2) (GRIND) (APPLY-EXTENSIONALITY :HIDE? T))))
   ("2" (GRIND))))
 ("2" (REWRITE "index_sets_finite")))
```

For an arbitrary real value `v`, we define `intersect(v)` to be the set of sensors whose interval contains `v`.

```
v: VAR real
intersect(v): finite_set[index] =
     { i |sensor(i)`lo <= v AND sensor(i)`hi >= v }
```

Now we are able to state the first important result: the `actual` value is in the intersection of at least `n-f` intervals.

```
actual_intersect: THEOREM card(intersect(actual)) >= n-f
```

The proof is fairly straightforward: we split the intervals containing the `actual` value into those that are faulty and those that are nonfaulty; we use `min_good` to establish that there are least `n-k` of the latter, and we are done.

```
(AUTO-REWRITE "index_sets_finite")
(EXPAND "intersect")
(CASE-REPLACE
 "{i | sensor(i)`lo <= actual AND sensor(i)`hi >= actual} =
  union(ok,
   {i | faulty(i) AND sensor(i)`lo <= actual
                  AND sensor(i)`hi >= actual})" :HIDE? T)
(("1"
  (REWRITE "card_disj_union")
  (("1" (USE "min_good") (ASSERT)) ("2" (HIDE 2) (GRIND))))
 ("2"
  (HIDE 2)
  (APPLY-EXTENSIONALITY :HIDE? T)
  (EXPAND* "union" "member")
  (REWRITE "good_sensor")
  (("1" (GRIND)) ("2" (GRIND)))))
```

A corollary to this result provides a basis for fault detection: any value that is not in the intersection of at least n−f intervals cannot be the correct value.

```
detection: COROLLARY card(intersect(v)) < n-f => v /= actual
```

This is proved straightforwardly from the previous theorem.

```
(SKOSIMP) (USE "actual_intersect") (ASSERT)
```

Now we need to establish that the interval specified by $\bigcap_{f,n}(S)$ is well defined and contains the `actual` value. We begin by defining `lo_vals` as the set of `lo` values that lie to the left of the `actual` value.

```
lo_vals: finite_set[real] =
   { v | EXISTS i: sensor(i)`lo = v AND v <= actual }

lo_vals_nonempty: LEMMA nonempty?(lo_vals)
```

We then state a lemma that asserts this set is nonempty. Its proof is a straightforward consequence of the fact that any nonfaulty sensor must contain the `actual` value (so its `lo` end must be to the left of `actual`).

```
(LEMMA "min_good")
(USE "nonempty_card[index]")
(ASSERT)
(EXPAND "nonempty?")
(EXPAND "empty?")
(SKOSIMP)
(EXPAND "member")
(USE "good_sensor")
(GRIND)
```

Next we refine `lo_vals` to yield `lo_cands` (the candidates for the left end of the fusion interval); these are those `lo` values that lie within the intersection of at least n−k sensor intervals.

```
lo_cands: finite_set[real] =
    intersection(lo_vals, { v | card(intersect(v)) >= n-f })
```

We will need to establish that this set is nonempty; we exhibit the rightmost (largest) value in `lo_vals` for this purpose.

```
rightmost_lo: real = max[real, <=](lo_vals)

lo_cands_nonempty: LEMMA nonempty?(lo_cands)
```

The proof of this nonemptiness lemma is quite difficult.

```
(EXPAND "nonempty?")
(EXPAND "empty?")
(REWRITE "forall_not")
(EXPAND "member")
(INST + "rightmost_lo")
(EXPAND "lo_cands")
(EXPAND "intersection")
(SPLIT)
(("1" (GRIND))
 ("2"
  (EXPAND "member")
  (LEMMA "rightmost_lo")
  (USE "max_lem[real,<=]")
  (("1"
    (GROUND)
    (USE "actual_intersect")
    (CASE "subset?(intersect(actual),intersect(rightmost_lo))")
    (("1" (USE "card_subset[index]") (ASSERT))
     ("2"
      (HIDE -1 -2 -5 2)
      (EXPAND "subset?")
      (SKOSIMP)
      (EXPAND "member")
      (EXPAND "intersect")
      (EXPAND "lo_vals")
      (SKOSIMP)
      (ASSERT)
      (INST - "sensor(x!1)`lo")
      (EXPAND "lo_vals")
      (INST + "x!1"))))
   ("2" (USE "lo_vals_nonempty") (EXPAND "nonempty?") (PROPAX))
   ("3" (USE "total_le")))))
```

Given that `lo_cands` is nonempty, we can now define the `leftedge` of the fusion interval as its minimum value.

```
leftedge: real = min[real, <=](lo_cands)

left_soundness: THEOREM leftedge <= actual
```

We then prove the “left half” of the soundness theorem: the `leftedge` lies to the left of the `actual` value. The proof is straightforward given properties of the `min` function.

```
(LEMMA "min_lem[real,<=]")
(("1"
  (INST - "lo_cands" "leftedge")
  (("1" (GRIND))
   ("2" (LEMMA "lo_cands_nonempty") (EXPAND "nonempty?") (PROPAX))))
 ("2" (LEMMA "total_le") (PROPAX)))
```

The construction for the “rightedge” of the fusion interval is exactly the dual of that for `leftedge` and is omitted.

Chapter 3

Conclusion

Modern smart sensors provide diagnostic information on their own performance and an estimate of the error in their sampled value. To allow higher-level sensor validation and fusion to be performed in an application-independent manner, it is convenient to integrate these items of information into a sensor reading that represents a *range* of values (given by the lower and upper bounds of the range). The sensor fusion problem is then to combine several such readings from different sensors into a best consensus value and, optionally, to perform higher-level validation by comparing the reading from one sensor against those from others.

We described the "fusion functions" $\bigcap_{f,n}(S)$ of Marzullo [Mar90] and $\mathcal{F}_n^f(S)$ of Schmid and Schossmaier [SS01] and presented some of their properties. $\mathcal{F}_n^f(S)$ is generally to be preferred to $\bigcap_{f,n}(S)$ because it satisfies a "Lipschitz Condition" (small changes in sensor readings produce small changes in its output), and is optimal among all such functions.

We used PVS formally to prove the well-definedness and soundness of $\bigcap_{f,n}(S)$ (i.e., it always contains the correct value), from which soundness of $\mathcal{F}_n^f(S)$ also follows.

Bibliography

[KN97] Herman Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. In *6th IEEE Workshop on Future Trends in Distributed Computing*, pages 310–315, Tunis, Tunisia, October 1997. IEEE Computer Society.

[Lam87] Leslie Lamport. Synchronizing time servers. Technical Report 18, DEC Systems Research Center, Palo Alto, CA, June 1987.

[Mar90] Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, November 1990.

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[SS01] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101–111, May 2001.

[WL88] J. Lundelius Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, April 1988.