8-2016

Avoiding Fake Boundaries in Set Interval Computing

Anthony Welte
École Nationale Supérieure de Techniques Avancées Bretagne, tony.welte@gmail.com

Luc Jaulin
École Nationale Supérieure de Techniques Avancées Bretagne, lucjaulin@gmail.com

Martine Ceberio
The University of Texas at El Paso, mceberio@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-16-56

Avoiding Fake Boundaries in Set Interval Computing

Anthony Welte and Luc Jaulin
Lab STICC
École Nationale Supérieure
de Techniques Avancées Bretagne
(ENSTA Bretagne)
2 rue François Verny
29806 Brest, France
Emails: tony.welte@gmail.com,
lucjaulin@gmail.com

Martine Ceberio and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, Texas 79968, USA
Emails: mceberio@utep.edu,
vladik@utep.edu

***Abstract*—Set intervals techniques are an efficient way of dealing with uncertainty in spatial localization problems. In these techniques, the desired set (e.g., set of possible locations) is represented by an expression that uses intersection, union, and complement of input sets – which are usually only known with interval uncertainty. To find the desired set, we can, in principle, perform the corresponding set-interval computations one-by-one. However, the estimates obtained by such straightforward computations often contain extra elements – e.g., fake boundaries. In this paper, we show that we can eliminate these fake boundaries (and other extra elements) if we first transform the original set expression into an appropriate DNF/CNF form.**

I. Formulation of the Problem

Location and mapping problems are important. In many practical situations, we are interested in location and mapping: we want to find the location of an object, and we want to find the exact boundaries of a region.

In many cases, we can use the GPS signals to get reasonably accurate locations of different objects. This possibility is based on the fact that in most situations, electromagnetic propagate in the atmosphere with a known (and practically constant) speed and along straight lines.

For underwater objects, however, determining the exact location is not easy: radio-signals like the GPS signals do not penetrate in water. In principle, we can use sound signals to "ping" the object and thus, determine its location. However, due to inhomogeneity of water and to the presence of many potential obstacles, the direction and speed of a sound signal may change as the signal propagates.

Set computation: a useful tool for solving location and mapping problems. To locate an underwater object, we usually perform several measurements. Based on each measurement, we can find the set S of all the locations x which are consistent with the measurement results. In this case, if we perform n measurements, and find the corresponding sets $S_1, \ldots, S_n$, then we can conclude that the actual location x belongs to all these sets. In this case, the set S of all possible locations x is the intersection of the n sets corresponding to n measurement results: $S = S_1 \cap \ldots \cap S_n$.

We also know that the underwater object is in the water, so it cannot be inside the 3-D areas that were already identified as underwater rocks or peers. If we denote the corresponding "impossible-to-be" sets by $I_1, \ldots, I_m$, then we can get a better description of the set of possible locations S as the difference

$$S = (S_1 \cap \ldots \cap S_n) - (I_1 \cup \ldots \cup I_m),$$

where $A - B$ denotes the set difference.

This is an idealized situation, when we are sure that all the sensor recordings describe a signal reflected by the object. In practice, we may have outliers – recordings which are caused by some external noise or by a reflection from a nearby object. For such outlier reflections, the corresponding set S_i describes the location of a different object; as a result, the intersection of this set with the others sets S_j (that describe reflections from the object of interest) may be empty.

One of the techniques that helps to locate an object in such situations is based on knowing the maximum possible number of outliers q. In this case, instead of taking the intersection of all n sets S_i – i.e., instead of considering the set of all the elements that belong to all n sets, we consider the set of all the elements that belong to at least $n - q$ different sets S_i. Such a "q-relaxed intersection" can also be described in terms of union and intersection: namely, it can be described as

$$S = \bigcup_{A} \bigcap_{i \in A} S_i,$$

where we consider all possible subsets $A \subseteq \{1, \ldots, n\}$ with at least $n - q$ elements.

This idea implicitly assumes that all the sensors are equally reliable. In practice, different sensors may have different reliability. This reliability can be gauged by the probability p_i that the signal coming out of the i-th sensor actually reflects the location of the desired object. Then, the probability that the signal from the i-th sensor is an outlier is equal to $1 - p_i$.

Different sensors are usually independent. So, if a location x appears as possible based on the data provided by two sensors i and j, then the probability that this location is not real – i.e.,

the probability that both sensors malfunctioned – is equal to the product $(1-p_i)\cdot(1-p_j)$. In general, for each location x, if $A \subseteq \{1,\ldots,n\}$ is the set of all the sensors i for which x the possible location (i.e., for which $x \in S_i$), then the probably that this is a wrong location – i.e., that all the sensors from this set malfunctioned – is equal to the product $\prod\limits_{i\in A}(1-p_i)$. It is reasonable to conclude that the location x is possible if the probability of a mistake is sufficiently small: smaller than some threshold p_0:

$$\prod_{i\in A}(1-p_i) \le p_0.$$

For computational purposes, it is convenient to replace the product with the sum by taking minus logarithm of both sides; then, this condition takes the form

$$\sum_{i\in A} w_i \ge w_0,$$

where we denoted $w_i \stackrel{\text{def}}{=} -\ln(1-p_i)$ and $w_0 \stackrel{\text{def}}{=} -\ln(p_0)$. The resulting set of possible locations then takes the following form:

$$S = \bigcup_{A:\,\sum\limits_{i\in A} w_i \ge w_0} \left(\bigcap_{i\in A} S_i\right).$$

When all the sensors are equally reliable, i.e., when all the probabilities $p_1 = \ldots = p_n$ are equal, and thus,

$$w_1 = \ldots = w_n,$$

the condition $\sum\limits_{i\in A} w_i \ge w_0$ simply means that the set A contains at least $\dfrac{w_0}{w_1}$ elements. So, for

$$q = n - \frac{w_0}{w_1},$$

this means that we consider all the subsets with at least $n-q$ elements.

The above description includes the cases when the location appears as possible based on the signals from all the sensors. In some cases, however, when we know that a certain percentage of sensors is bound to malfunction, we may want to dismiss locations that appear on too many sensors – that would probably mean that the signal is too strong and is, thus, not a reflection from the object of interest. This leads to more complex schemes.

We can consider more sophisticated combination schemes. In all these cases, the desired set A is described as an expression that combines the input sets $A_1,\ldots,A_n,\ldots$ by using the three basic set operations: union, intersection, and complement. Thus, for localization problems, it is important to be able, given sets $A_1,\ldots$, to compute the set A described by such an expression. Such computation is known as *set computation*.

Need for set intervals. In many practical situations, we know the inputs sets only approximately. For example, in practice, we only have an approximate information about the 3-D location I_1 of an underwater rock – one of the locations where an underwater object cannot be. In the ideal case, when we know the exact 3-D map of this rock, for each spatial location x, we know whether x belongs to this set or not. In practice:

- for some locations x, we know that $x \in I_1$;
- for some other locations x, we know that $x \notin I_1$; however,
- for some locations x, we do not know whether $x \in I_1$ or $x \notin I_1$.

Such a situation can be naturally described by listing two sets:

- the set $\underline{I}_1$ of all the locations x that we know are inside I_1, and
- the set $\overline{I}_1$ of all the locations that *can be* inside I_1, i.e., locations x for which we either know that $x \in I_1$, or we do not know whether $x \in I_1$ or not.

In this case, the only information that we have about the actual (unknown) set I_1 is that this set is in between $\underline{I}_1$ and $\overline{I}_1$:

$$\underline{I}_1 \subseteq I_1 \subseteq \overline{I}_1.$$

Interval and set-interval computations are indeed very useful in location and mapping problems, especially for underwater objects; see, e.g., [1], [2], [3], [4], [5], [6], [7], [8].

Resulting computational problem. The need to consider set intervals in set computations leads to the following computational problem.

We have a set-theoretic expression $A = f(A_1,\ldots,A_N)$ that expressed the desired set A as a result of a sequence of basic set-theoretic operations (union, intersection, and complement) applied to the original sets $A_1,\ldots,A_N$.

In general, we do not know the sets A_i. Instead, for each i, we know the lower set $\underline{A}_i$ and the upper set $\overline{A}_i$ for which $\underline{A}_i \subseteq \overline{A}_i$. The only information that we have about the unknown set A_i is that $\underline{A}_i \subseteq A_i \subseteq A_i \subseteq \overline{A}_i$. In other words, for each i, we know the corresponding *set interval*

$$\mathbf{A}_i = \left[\underline{A}_i, \overline{A}_i\right] \stackrel{\text{def}}{=} \{A_i : \underline{A}_i \subseteq \overline{A}_i\}.$$

For different sets $A_i \in \mathbf{A}_i$, in general, we get different sets $A = f(A_1,\ldots,A_N)$. Our objective is to find the class of all such sets A:

$$\mathcal{A} = \{f(A_1,\ldots,A_N) : A_1 \in \mathbf{A}_1,\ldots,A_N \in \mathbf{A}_N\}.$$

How this problem is solved now. It is known (see, e.g., [10]) how to compute the range for the case when the set operation $f(A_1,\ldots)$ is simply one of the three basic set operations. In this case, we have explicit formulas for the corresponding range $\mathcal{A}$:

- for the union $f(A_1,A_2) = A_1 \cup A_2$, we have

$$\mathcal{A} = \left[\underline{A}_1 \cup \underline{A}_2, \overline{A}_1 \cup \overline{A}_2\right];$$

- for the intersection $f(A_1,A_2) = A_1 \cap A_2$, we have

$$\mathcal{A} = \left[\underline{A}_1 \cap \underline{A}_2, \overline{A}_1 \cap \overline{A}_2\right];$$

- for the complement $f(A_1, A_2) = A_1 - A_2$, we have

$$\mathcal{A} = \left[\underline{A}_1 - \overline{A}_2, \overline{A}_1 - \underline{A}_2\right].$$

In general, we can:

- *parse* the expression $f(A_1, \ldots, A_n)$, i.e., represent the formula $f(A_1, \ldots, A_n)$ as a sequence of elementary set-theoretic operations, and then
- perform computations step-by-step, replacing each elementary set operation with the corresponding operation with set intervals.

One can prove, by induction, that as a result, we always get an *enclosure* $\mathbf{A}' \supseteq \mathcal{A}$ for the desired range; see, e.g., [9].

Problem: fake boundaries. It is known that while the above procedure always leads to an enclosure for the desired class $\mathcal{A}$, the resulting class $\mathbf{A}'$ is often larger than the desired class $\mathcal{A}$, i.e., contains many unneeded sets.

This can be easily illustrated on the following toy example. Let U be the universal set, and let us assume that we know nothing about the set $A_1 \subseteq U$. In this case, the range $\mathbf{A}_1$ of possible sets A_1 is simply the class of all the subsets of the universal set: $\mathbf{A}_1 = [\emptyset, U]$.

Suppose now that we want to compute the range of the function $f(A_1) = A_1 \cup (U - A_1)$. Of course, for every set A_1, the resulting set $f(A_1)$ is simply equal to the universal set, so the actual range is $\mathcal{A} = \{U\} = [U, U]$. Let us see, however, what we get if we apply the above procedure. According to the above procedure, we first represent the expression $f(A_1)$ as a sequence of elementary set-theoretical operations:

- first, we compute $A_2 \stackrel{\text{def}}{=} U - A_1$;
- then, we compute the union $A = A_1 \cup A_2$.

According to the above procedure, we perform these two operations with set intervals:

- first, we compute

$$\mathbf{A}_2 = U - \mathbf{A}_1 = [U, U] - \left[\underline{A}_1, \overline{A}_1\right] =$$

$$[U, U] - [\emptyset, U] = [U - U, U - \emptyset] = [\emptyset, U];$$

- then, we compute

$$\mathbf{A}' = \mathbf{A}_1 \cup \mathbf{A}_2 =$$

$$[\emptyset, U] \cup [\emptyset, U] = [\emptyset \cup \emptyset, U \cup U] = [\emptyset, U].$$

Thus, instead of the single set U, we get the class of all possible subsets of U.

We can give more realistic examples where the resulting class has unnecessary sets. For example, let us assume that we have three sets A, B, and C, and we are computing the expression

$$X = (A \cup B \cup C) \cap (A \cup B \cup (U - C)).$$

One can easily check that this expression is equivalent to $X = A \cup B$, so the actual range is equal to

$$\mathbf{X} = \left[\underline{X}, \overline{X}\right] = \left[\underline{A} \cup \underline{B}, \overline{A} \cup \overline{B}\right].$$

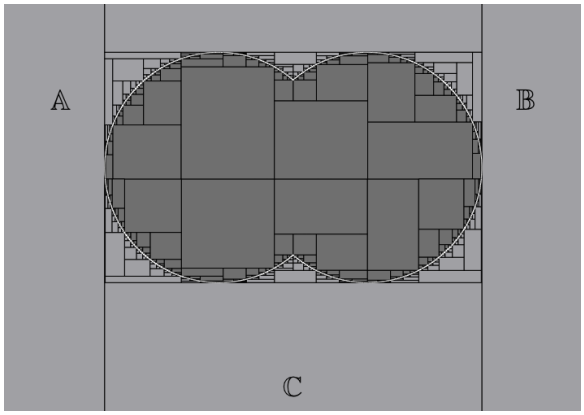

Fig. 1. Actual range $\mathbf{X}$

On the example, when all three sets are disks with uncertain boundary, the desired class $\mathbf{X}$ is shown in Fig. 1.

What will happen, however, if we apply the above algorithm to the original expression? To compute this expression:

- first, we compute the union $A_1 \stackrel{\text{def}}{=} A \cup B \cup C$;
- then, we compute the difference $A_2 \stackrel{\text{def}}{=} U - C$;
- after that, we compute the union $A_3 \stackrel{\text{def}}{=} A \cup B \cup A_2$; and
- finally, we compute the intersection $A = A_1 \cap A_3$.

In this case, the above algorithm leads to the following result:

- first, we compute the range of $A_1 = A \cup B \cup C$ as

$$\mathbf{A}_1 = \left[\underline{A} \cup \underline{B} \cup \underline{C}, \overline{A} \cup \overline{B} \cup \overline{C}\right];$$

- then, we compute the range of $A_2 = U - C$, as

$$\mathbf{A}_2 = [U, U] - \left[\underline{C}, \overline{C}\right] = \left[U - \overline{C}, U - \underline{C}\right];$$

- after that, we use the ranges for A, B, and A_2, to estimate the range of $A_3 = A \cup B \cup A_2$, as

$$\mathbf{A}_3 = \left[\underline{A} \cup \underline{B} \cup (U - \overline{C}), \overline{A} \cup \overline{B} \cup (U - \underline{C})\right];$$

- finally, we estimate the range of the intersection A as $\mathbf{A}' = \left[\underline{A}', \overline{A'}\right]$, where

$$\underline{A}' = (\underline{A} \cup \underline{B} \cup \underline{C}) \cap (\underline{A} \cup \underline{B} \cup (U - \overline{C})) \text{ and}$$

$$\overline{A'} = (\overline{A} \cup \overline{B} \cup \overline{C}) \cap \overline{A} \cup \overline{B} \cup (U - \underline{C})).$$

We can see that the upper bound $\overline{A'}$, in addition to the desired values $\overline{A} \cup \overline{B}$, also contains all the values from the "boundary" $\overline{C} - \underline{C}$ of the set interval C; see Fig. 2.

These fake boundaries is what we need to eliminate.

It is, in principle, possible to eliminate fake boundaries. In [9], we have proven:

- that the range $\mathcal{A}$ always has the form of a set interval $\mathcal{A} = \left[\underline{A}, \overline{A}\right]$ for appropriate sets $\underline{A}$ and $\overline{A}$, and
- that it is, in principle, possible to compute both sets $\underline{A}$ and $\overline{A}$.

Specifically:

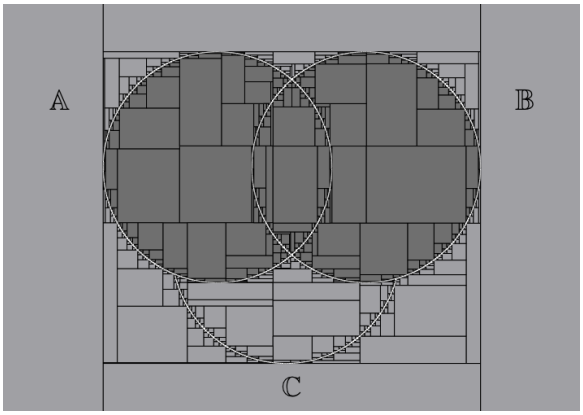

Fig. 2. An estimate $\mathbf{X}'$ with a fake boundary

- we get exactly the upper set $\overline{A}$ if we apply the above step-by-step algorithm to the equivalent canonical DNF form of the expression $f(A_1, \ldots, A_N)$, and
- we get exactly the lower set $\underline{A}$ if we apply the above step-by-step algorithm to the canonical CNF form of the expression $f(A_1, \ldots, A_N)$.

The notions of the canonical DNF and CNF forms come from propositional logic – which makes perfect sense since there is a 1-1 correspondence between set operations and propositional formulas:

- the condition $x \in A_1 \cup A_2$ means that
$$(x \in A_1) \vee (x \in A_2);$$
- the condition $x \in A_1 \cup A_2$ means that
$$(x \in A_1) \,\&\, (x \in A_2); \text{ and}$$
- the condition that $x \in A_1 - A_2$ means that
$$(x \in A_1) \,\&\, \neg(x \in A_2).$$

By replacing union with "or", intersection with "and", etc., we can thus assign, to each set operation $f(A_1, \ldots, A_N)$, a propositional formula $F(a_1, \ldots, a_N)$ for which a point x belongs to the set $f(A_1, \ldots, A_N)$ if and only the formula $F(a_1, \ldots, a_N)$ is true for the variables a_i describing whether x belongs to A_i or not:

$$x \in f(A_1, \ldots, A_N) \Leftrightarrow F(x \in A_1, \ldots, x \in A_N).$$

For each propositional formula, we can build a canonical DNF form by enumerating all the combinations of truth variables $(a_1, \ldots, a_N)$ for which this formula is true.

For example, the above set operation

$$(A \cup B \cup C) \cap (A \cup B \cup (U - C))$$

corresponds to the propositional formula

$$(a \vee b \vee c) \,\&\, (a \vee b \vee \neg c).$$

By enumerating all possible tuples (a, b, c) for which this propositional formula is true, we can form the canonical DNF form. Specifically:

- this propositional formula is true when $a = b = c =$"true"; this leads to the term $a \,\&\, b \,\&\, c$;
- this formula is also true when $a = b =$"true" and $c =$"false"; this leads to $a \,\&\, b \,\&\, \neg c$;
- it is true when $a =$"true", $b =$"false", and $c =$"true"; this leads to $a \,\&\, \neg b \,\&\, c$;
- it is true when $a =$"true" and $b = c =$"false"; this leads to $a \,\&\, \neg b \,\&\, \neg c$;
- it is true when $a =$"false" and $b = c =$"true"; this leads to $\neg a \,\&\, b \,\&\, c$;
- finally, it is true when $a =$"false", $b =$"true", and $c =$"false"; this leads to $\neg a \,\&\, b \,\&\, \neg c$.

The formula is true if one of these cases is true. So, our formula $F(a, b, c)$ has the following canonical DNF form:

$$(a \,\&\, b \,\&\, c) \vee (a \,\&\, b \,\&\, \neg c) \vee (a \,\&\, \neg b \,\&\, c) \vee$$

$$(a \,\&\, \neg b \,\&\, \neg c) \vee (a \,\&\, b \,\&\, c) \vee (\neg a \,\&\, b \,\&\, \neg c).$$

This propositional formula corresponds to the following set function:

$$(A \cap B \cap C) \cup (A \cap B \cap (U - C)) \cup$$

$$(A \cap (U - B) \cap C) \cup (A \cap (U - B) \cap (U - C)) \cup$$

$$((U - A) \cap B \cap C) \cup ((U - A) \cap B \cap (U - C)).$$

One can easily check that if we compute $\overline{X}$ by applying the above step-by-step procedure to this formula, we get exactly the desired upper bound $\overline{X} = \overline{A} \cup \overline{B}$.

To compute the canonical CNF form, we, vice versa, list all the tuples $(a_1, \ldots, a_N)$ for which the original propositional formula is *false*. Then, we say that the propositional formula is true if the tuple is different from each of these false-producing tuples.

Let us show how this procedure works on the example of the same $(A \cup B \cup C) \cap (A \cup B \cup (U - C))$ and propositional formula $(a \vee b \vee c) \,\&\, (a \vee b \vee \neg c)$.

- This formula is false when $a = b =$"false" and $c =$"true". To avoid this tuple, we need to make sure that either a is true, or n is true, or c is false. The corresponding term is $a \vee b \vee \neg c$.
- This formula is also false when $a = b = c =$"false". To avoid this tuple, we need to make sure that either a is true, or n is true, or c is true. The corresponding term is $a \vee b \vee c$.

The formula is true for some tuple if this tuple is different from both false-inducing tuples, i.e., if

$$(a \vee b \vee \neg c) \,\&\, (a \vee b \vee c);$$

this is the canonical CNF form of the original formula. This propositional formula corresponds to the following set operation:

$$(A \cup B \cup (U - C)) \cap (A \cup B \cup C).$$

One can easily check that if we compute the lower set $\underline{X}$ by applying the above step-by-step algorithm to this set operation, we will get the exact lower set

$$\underline{X} = \underline{A} \cup \underline{B}.$$

Problem: using canonical DNF and CNF forms requires too much computation time. The main problem with the above idea is that often, it requires too many operations. For example, in the above example, the canonical DNF form requires computing 6 intersections of 3 set intervals each – and then computing the union of the resulting set intervals.

In general, when we have N sets, we can have 2^N different true-false tuples $(a_1, \ldots, a_N)$, For each of these tuples, the corresponding propositional formula is either true or false. If for a tuple, the given formula is true, then this tuple leads to a term in the canonical DNF form. If for this tuple, the given formula is false, we get a term in the canonical CNF form. To perform the above computations, we need to use both the DNF form (to compute $\overline{A}$) and the CNF form (to compute $\underline{A}$). Thus, overall, we need to compute the set interval values of 2^N terms.

In some practical situations, when we have many sensors, the number N can be huge; in this case, 2^N can be astronomically, unrealistically huge. So, a natural question is: how can we perform set interval computations faster and still eliminate all fake boundaries?

What we do in this paper. In this paper, we show how computations-without-fake-boundaries can be performed much faster.

II. How to Avoid Fake Boundaries Faster: Main Idea

Main idea. Our main idea is to use *general* DNF and CNF forms instead of the *canonical* ones.

For a propositional formula, a DNF form is a *disjunction* ("or"-combination) of *conjunctions*, i.e., "and"-combinations of propositional variables and their negations. In set-theoretic terms, a DNF form is thus a union of intersections of sets and their complements.

For example, for the above formula, $a \vee b$ is a DNF form, in which each of the conjunctions a and b consists of only one term. In this case, $A \cup B$ is the corresponding set expression.

Alternatively, we could use the following DNF expression:

$$(a \,\&\, b) \vee (a \,\&\, \neg b) \vee (\neg a \,\&\, b),$$

which corresponds to the set operation

$$(A \cap B) \cup (A \cap -B) \cup (-A \cap B),$$

where we denoted $-A \stackrel{\text{def}}{=} U - A$.

Similarly, for a propositional formula, a CNF form is a *conjunction* ("and"-combination) of *disjunctions*, i.e., "or"-combinations of propositional variables and their negations. In set-theoretic terms, a DNF form is thus an intersection of unions of sets and their complements.

For example, for the above formula, $a \vee b$ is a CNF form, with only one disjunction $a \vee b$. In this case, $A \cup B$ is the corresponding set expression.

Let us show that by using the general DNF and CNF forms, we can indeed get the same exact bounds $\underline{A}$ and $\overline{A}$ as by using the canonical DNF and CNF forms.

Proposition 1. *Let $f(A_1, \ldots, A_N)$ be a set operation in the DNF form, and let $\mathbf{A}_1, \ldots, \mathbf{A}_N$ be set intervals. Then, if we apply the above step-by-step algorithm to this data*

$$(f(A_1, \ldots, A_n), \mathbf{A}_1, \ldots, \mathbf{A}_N),$$

the resulting upper set $\overline{A'}$ will be equal to the upper set $\overline{A}$ of the corresponding range

$$\mathcal{A} = \left[\underline{A}, \overline{A}\right] = f(\mathbf{A}_1, \ldots, \mathbf{A}_N).$$

Proof.

1°. Due to the above-mentioned result from [9], the range $\mathcal{A}$ has the form of a set interval $\mathcal{A} = \left[\underline{A}, \overline{A}\right]$. Thus, the upper set $\overline{A}$ is equal to the union of all the possible sets from this range – i.e., to the union of all the sets $A = f(A_1, \ldots, A_N)$ corresponding to different combinations of sets $A_i \in \mathbf{A}_i$.

So, to prove that the set $\overline{A'}$ is equal to the desired set $\overline{A}$, it is sufficient to prove that the set $\overline{A'}$ is equal to the union of all such sets $A = f(A_1, \ldots, A_N)$.

2°. Let us first prove that the union $\overline{A}$ of all the sets $A = f(A_1, \ldots, A_N)$ is indeed contained in the resulting set $\overline{A'}$.

To prove this, we will prove that for each tuple $(A_1, \ldots, A_N)$ with $A_i \in \mathbf{A}_i$ for all i, the set $A = f(A_1, \ldots, A_N)$ is a subset of $\overline{A'}$. Indeed, the set operation $f(A_1, \ldots, A_N)$ has a DNF form

$$f(A_1, \ldots, A_N) = (A_i \cap -A_j \cap \ldots \cap A_k) \cup (\ldots) \cup \ldots$$

When we apply the above step-by-step algorithm to this form, when computing $\overline{A'}$, we replace A_i with

$$\overline{A}_i \supseteq A_i$$

and $-A_j$ with

$$-\underline{A_j} \supseteq -A_j.$$

For each term, this replacement makes it larger (or the same), so each conjunction is contained in the result of the corresponding replacement:

$$(A_i \cap -A_j \cap \ldots \cap A_k) \subseteq \left(\overline{A}_i \cap -\underline{A}_j \cap \ldots \cap \overline{A}_k\right).$$

Since this inclusion holds for each conjunction, it holds for their union as well:

$$(A_i \cap -A_j \cap \ldots \cap A_k) \cup (\ldots) \cup \ldots \subseteq$$

$$\left(\overline{A}_i \cap -\underline{A}_j \cap \ldots \cap \overline{A}_k\right) \cup (\ldots) \cup \ldots,$$

i.e., indeed, $A \subseteq \overline{A'}$.

3°. To complete the proof, we need to show that the set $\overline{A'}$ produced by the algorithm is contained in the union $\overline{A}$ of all the sets $A = f(A_1, \ldots, A_N)$ corresponding to $A_i \in \mathbf{A}_i$.

To prove this, we will show that every element $x \in \overline{A'}$ belongs to a set $A = f(A_1, \ldots, A_N)$ for appropriately chosen sets $A_i \in \mathbf{A}_i$. Indeed, let $x \in \overline{A'}$. Since the set $\overline{A'}$ is defined as a union of several conjunctions (intersections), the fact that the element x belongs to this union means that it belongs to one of these intersections, e.g., to a set of the type $\overline{A}_i \cap -\underline{A}_j \cap \ldots \cap \overline{A}_k$. This means that for the appropriate choice of the sets $A_1, \ldots$, namely, for $A_i = \overline{A}_i$, $A_j = \underline{A}_j$, $\ldots$, the element x belongs to the corresponding intersection from the DNF expression $f(A_1, \ldots, A_N)$.

Since x belongs to this intersection, and the set $f(A_1, \ldots, A_N)$ is a union of several such intersections, we thus conclude that the element x belongs to the set $A = f(A_1, \ldots, A_N)$ – and hence, that x belongs to the union $\overline{A}$ of all such sets.

The proposition is proven.

Proposition 2. *Let $f(A_1, \ldots, A_N)$ be a set operation in the CNF form, and let $\mathbf{A}_1, \ldots, \mathbf{A}_N$ be set intervals. Then, if we apply the above step-by-step algorithm to this data*

$$(f(A_1, \ldots, A_n), \mathbf{A}_1, \ldots, \mathbf{A}_N),$$

the resulting lower set $\underline{A}'$ will be equal to the lower set $\underline{A}$ of the corresponding range

$$\mathcal{A} = \left[\underline{A}, \overline{A}\right] = f(\mathbf{A}_1, \ldots, \mathbf{A}_N).$$

Proof.

1°. Due to the above-mentioned result from [9], the range $\mathcal{A}$ has the form of a set interval $\mathcal{A} = \left[\underline{A}, \overline{A}\right]$. Thus, the lower set $\underline{A}$ is equal to the intersection of all the possible sets from this range – i.e., to the intersection of all the sets $A = f(A_1, \ldots, A_N)$ corresponding to different combinations of sets $A_i \in \mathbf{A}_i$.

So, to prove that the set $\underline{A}'$ is equal to the desired set $\underline{A}$, it is sufficient to prove that the set $\underline{A}'$ is equal to the intersection of all such sets $A = f(A_1, \ldots, A_N)$.

2°. Let us first prove that the intersection $\underline{A}$ of all the sets $A = f(A_1, \ldots, A_N)$ indeed contains the resulting set $\underline{A}'$.

To prove this, we will prove that for each tuple $(A_1, \ldots, A_N)$ with $A_i \in \mathbf{A}_i$ for all i, the set $A = f(A_1, \ldots, A_N)$ is a superset of $\underline{A}'$. Indeed, the set operation has a CNF form

$$f(A_1, \ldots, A_N) = (A_i \cup -A_j \cup \ldots \cup A_k) \cap (\ldots) \cap \ldots$$

When we apply the above step-by-step algorithm to this form, when computing $\underline{A}'$, we replace A_i with

$$\underline{A}_i \subseteq A_i$$

and $-A_j$ with

$$-\overline{A_j} \subseteq -A_j.$$

For each term, this replacement makes it smaller (or the same), so each disjunction contains the result of the corresponding replacement:

$$(A_i \cup -A_j \cap \ldots \cup A_k) \supseteq \left(\underline{A}_i \cup -\overline{A}_j \cup \ldots \cup \overline{A}_k\right).$$

Since this inclusion holds for each disjunction, it holds for their intersection as well:

$$(A_i \cup -A_j \cup \ldots \cup A_k) \cap (\ldots) \cap \ldots \supseteq$$

$$\left(\underline{A}_i \cup -\overline{A}_j \cup \ldots \cap \underline{A}_k\right) \cap (\ldots) \cap \ldots,$$

i.e., indeed, $A \supseteq \underline{A}'$.

3°. To complete the proof, we need to show that the set $\underline{A}'$ produced by the algorithm contains the intersection $\underline{A}$ of all the sets $A = f(A_1, \ldots, A_N)$ corresponding to $A_i \in \mathbf{A}_i$.

We will prove this by contradiction. Let us assume that for some $x \in \underline{A}$, we have $x \notin \underline{A}'$. Since the set $\underline{A}'$ is defined as an intersection of several disjunctions (unions), the fact that the element x does not belongs to this intersection means that it does not belong to one of these intersecting unions, e.g., to a set of the type $\underline{A}_i \cup -\overline{A}_j \cup \ldots \cup \underline{A}_k$. This means that for the appropriate choice of the sets $A_1, \ldots$, namely, for $A_i = \underline{A}_i$, $A_j = \overline{A}_j$, $\ldots$, the element x does not belong to the corresponding union from the CNF expression $f(A_1, \ldots, A_N)$.

Since x does not belong to this union, and the set $f(A_1, \ldots, A_N)$ is an intersection of several such unions, we thus conclude that the element x does not belongs to the set $A = f(A_1, \ldots, A_N)$ – and hence, that x does not belongs to the intersection $\underline{A}$ of all such sets A. This contradicts to our assumption that x belongs to this intersection. Thus, indeed, every element $x \in \underline{A}$ belongs to $\underline{A}'$, i.e., $\underline{A} \subseteq \underline{A}'$.

The two inclusions, from Parts 2 and 3 of this proof, imply that $\underline{A} \subseteq \underline{A}'$ and $\underline{A}' \subseteq \underline{A}$. thus, $\underline{A} = \underline{A}'$. The proposition is proven.

Conclusion. Thus, to perform set interval computations and avoid fake boundaries, it is not necessary to transform the original expression into canonical DNF and CNF forms – any DNF and CNF forms will do.

The above example shows that CNF and DNF forms can indeed be much shorter than the canonical ones, so we can indeed speed up computations – without introducing fake boundaries.

III. How Do We Get Shorter DNF and CNF Forms

Main idea. How can we get shorter DNF and CNF forms? To get the canonical DNF forms, we start with all the tuples for which the corresponding propositional formula is true. For each tuple, we can then write down the corresponding conjunction.

If we have two conjunctions that differ only by one variable, i.e., which have the form $F \,\&\, v$ and $F \,\&\, \neg v$, then, we can easily see, we can replace the part $(F \,\&\, v) \vee (F \,\&\, \neg v)$ of the original DNF formula with the equivalent simpler term F.

Similarly, to get a CNF form, we start with all the tuples for which the corresponding propositional formula is false. For each tuple, we can then write down the corresponding disjunction.

If we have two disjunctions that differ only by one variable, i.e., which have the form $G \vee v$ and $G \vee \neg v$, then, we can easily see, we can replace the part $(G \vee v) \,\&\, (G \vee \neg v)$ of the original CNF formula with the equivalent simpler term G.

By applying this procedure again and again, we can get shorter and shorter expressions.

DNF example. Let us show how this idea can work on the above example. We start with the canonical DNF form

$$(a \,\&\, b \,\&\, c) \vee (a \,\&\, b \,\&\, \neg c) \vee (a \,\&\, \neg b \,\&\, c) \vee$$
$$(a \,\&\, \neg b \,\&\, \neg c) \vee (a \,\&\, b \,\&\, c) \vee (\neg a \,\&\, b \,\&\, \neg c)$$

that describes all the tuples for which the original formula $(a \vee b \vee c) \,\&\, (a \vee b \vee \neg c)$ is true.

By looking at the above formula, we immediately see the pairs of conjunctions that differ by only one variable and be thus combined together:

- the conjunctions $a \,\&\, b \,\&\, c$ and $a \,\&\, b \,\&\, \neg c$ can be combined into a single conjunction $a \,\&\, b$;
- the conjunctions $a \,\&\, \neg b \,\&\, c$ and $a \,\&\, \neg b \,\&\, \neg c$ can be combined into a single conjunction $a \,\&\, \neg b$, and
- the conjunctions $\neg a \,\&\, b \,\&\, c$ and $\neg a \,\&\, b \,\&\, \neg c$ can be combined into a single conjunction $\neg a \,\&\, b$.

After these replacements, the original DNF formula is simplified into the following form:

$$(a \,\&\, b) \vee (a \,\&\, \neg b) \vee (\neg a \,\&\, b).$$

This form can be further simplified:

- by combining $a \,\&\, b$ and $a \,\&\, \neg b$, we get a, and
- by combining $a \,\&\, b$ and $\neg a \,\&\, b$, we get b.

Thus, we get the simplified DNF form $a \vee b$.

We could reach this form differently. We could:

- combine the conjunctions $a \,\&\, b \,\&\, c$ and $a \,\&\, \neg b \,\&\, c$ into a single conjunction $a \,\&\, c$;
- combine the conjunctions $a \,\&\, b \,\&\, c$ and $a \,\&\, \neg b \,\&\, c$ into a single conjunction $b \,\&\, c$; combine the conjunctions $a \,\&\, b \,\&\, \neg c$ and $a \,\&\, \neg b \,\&\, \neg c$ into a single conjunction $a \,\&\, \neg c$;
- combine the conjunctions $a \,\&\, b \,\&\, \neg c$ and $\neg a \,\&\, b \,\&\, c$ into a single conjunction $b \,\&\, \neg c$.

Then, we would get the new DNF form

$$(a \,\&\, c) \vee (b \,\&\, c) \vee (a \,\&\, \neg c) \vee (b \,\&\, \neg c).$$

Then, new combinations are possible; we could:

- combine $a \,\&\, c$ and $a \,\&\, \neg c$ into a single conjunction a, and
- combine $b \,\&\, c$ and $b \,\&\, \neg c$ into a single conjunction b.

Thus, we will get the same short DNF form $a \vee b$.

CNF example. In the CNF case, we start with the canonical CNf form

$$(a \vee b \vee c) \,\&\, (a \vee b \vee \neg c)$$

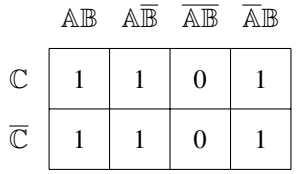

Fig. 3. Karnaugh map of the original formula

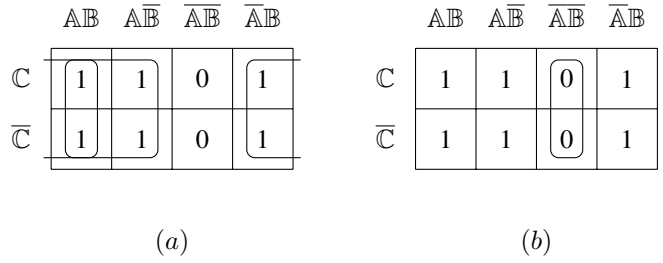

Fig. 4. Karnaugh map illustrating simplification of DNF (a) and CNF (b) forms

that describes all the tuples (a, b, c) for which the original propositional formula is false.

For this formula, there is only one possible combination: we can combine the disjunctions $a \vee b \vee c$ and $a \vee b \vee \neg c$ into a single disjunction $a \vee b$.

Karnaugh maps: a graphical representation of this idea. The above idea can be graphically represented by a *Karnaugh map*, where:

- cells corresponds to tuples,
- 1 (= "true") 0 (= "false") in a cell indicates whether the original formula is true or false for the corresponding tuple, and
- tuples differing by only variable are neighbor.

The Karnaugh-map representation of the original propositional formula is given on Fig. 3, and the above DNF and CNF reductions are illustrated on parts (a) and (b) of Fig. 4.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation grants CAREER 0953339, HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation. This research was performed during Anthony Welte's visit to the University of Texas at El Paso.

The authors are thankful to all the participants of the Summer Workshop on Interval Methods SWIM'2016 (Lyon, France, June 19–22, 2016) for valuable discussions.

REFERENCES

[1] Q. Brefort, L. Jaulin, M. Ceberio, and V. Kreinovich, "If we take into account that constraints are soft, then processing constraints becomes

algorithmically solvable", *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions CIES'2014*, Orlando, Florida, December 9–12, 2014, pp. 1–10.

[2] Q. Brefort, L. Jaulin, M. Ceberio, and V. Kreinovich, "Towards fast and reliable localization of an underwater object: an interval approach", *Journal of Uncertain Systems*, 2015, Vol. 9, No. 2, pp. 95–102.

[3] B. Desrochers and L. Jaulin, "Computing a guaranteed approximation for the zone explored by a robot", *IEEE Transaction on Automatic Control*, 2016.

[4] B. Desrochers and L. Jaulin, "Relaxed intersection of thick sets", *Abstracts of the 17th International Symposium on Scientific Computing, Computer Arithmetic, and Verified Numerical Computation SCAN'2016*, Uppsala, Sweden, September 26–29, 2016.

[5] G. S. Franco and L. Jaulin, "Avoiding fake boundaries in interval analysis", *Abstracts of the Summer Workshop on Interval Methods SWIM'2016*, Lyon, France, June 19–22, 2016.

[6] L. Jaulin, "Range-only SLAM with occupancy maps: a set-membership approach", *IEEE Transactions on Robotics*, 2011, Vol. 27, No. 5, pp. 1004–1010.

[7] L. Jaulin, E. Walter, and O. Didrit, "Guaranteed robust nonlinear parameter bounding", *Proceedings of IMACS/IEEE-SMC CESA'96 Multi-Conference on Computational Engineering in Systems Applications CESA'96, Symposium on Modelling, Analysis and Simulation*, Lille, France, July 9–12, 1996, pp. 1156–1162.

[8] J. Sliwka, L. Jaulin, M. Ceberio, and V. Kreinovich, "Processing Interval Sensor Data in the Presence of Outliers, with Potential Applications to Localizing Underwater Robots", *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics SMC'2011*, Anchorage, Alaska, October 9–12, 2011, pp. 2330–2337.

[9] J. T. Yao, Y. Y. Yao, V. Kreinovich, P. Pinheiro da Silva, S. A. Starks, G. Xiang, and H. T. Nguyen, "Towards more adequate representation of uncertainty: from intervals to set intervals, with the possible addition of probabilities and certainty degrees", *Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZ-IEEE'2008*, Hong Kong, China, June 1–6, 2008, pp. 983–990.

[10] Y. Y. Yao and X. Li, "Comparison of rough-set and interval-set models for uncertainty reasoning", *Fundamenta Informaticae*, 1996, Vol. 27, pp. 289–298.