Applying Theory to Practice

RONALD FAGIN, *IBM Almaden Research Center, San Jose, CA.*

Abstract

By making use of three IBM case studies involving the author and colleagues, this paper is about applying theory to practice. In the first case study, the system builders (or practitioners) initiated the interaction. This interaction led to the following problem. Assume that there is a set of objects, each with multiple attributes, and there is a numerical score assigned to each attribute of each object. In the spirit of real-valued logics, there is a scoring function (such as the min or the average), and a ranking of the objects is obtained by applying the scoring function to the scores of each object's attributes The problem is to find the top k objects, while minimizing the number of database accesses. An algorithm is given that is optimal in an extremely strong sense: not just in the worst case or the average case, but (up to a constant factor) in every case! Even though the algorithm is only 8 lines long (!), the paper containing the algorithm won the 2014 Gödel Prize, the top prize for a paper in theoretical computer science. The interaction in the second case study was initiated by theoreticians, who wanted to lay the foundations for 'data exchange', in which data is converted from one format to another. Although this problem may sound mundane, the issues that arise are fascinating, and this work made data exchange a new subfield, with special sessions in every major database conference. This work won the 2020 Alonzo Church Award, the highest prize for research in logic and computation. The third case study, specifically on real-valued (or 'fuzzy') logic, arose as part of a large 'Logical Neural Nets' (LNN) project at IBM. The inputs to, say, an 'and' gate could each be any numbers in the interval [0,1]. The system builders of LNN wanted a sound and complete axiomatization for real-valued logic, so that they could arrive at truth values given other truth values whenever possible. This recent work provides a sound and complete axiomatization for a large class of real-valued logics, including the most common ones. It also allows weights, where the importance of some subformulas can be greater than that of other subformulas. This paper is aimed at both theoreticians and system builders, to show them the mutual benefits of working together. This is via the three case studies mentioned above: two initiated by the system builders, and one by the theoreticians. The moral for the theoreticians is to show by example how to apply theory to practice, and why applying theory to practice can lead to better theory. The moral for the system builders is the value of theory, and the value of involving theoreticians. This paper is written in a very informal style. In fact, it is based closely on a talk on 'Applying theory to practice' that the author has presented a number of times.

1 Case study #1: Garlic (1996)

The following scenario arose from IBM's Garlic project [5] around 1996. Laura Haas, then the manager of the Garlic project, and later an IBM Fellow and Director of Computer Science at IBM Almaden (and currently Dean at UMass-Amherst) dropped by my office and said, 'Mr. Database Theoretician, we've got a problem with Garlic, our multimedia database system!'

What was Laura's problem? Garlic was 'on top of' (i.e. accessed) various database systems, including IBM's DB2 and QBIC ('Query by Image Content' [19]). In QBIC, objects can be accessed based on color, shape or texture. The problem, as Laura told me, was that the answers to queries in DB2 are sets or bags (in Laura's case, sets), whereas the answers to queries in QBIC are sorted lists. Laura asked how to combine the results.

It turned out that I knew that the answers to queries in QBIC were not sorted lists, but scored lists (where each result in the answer has a numerical score). I knew this because I had once written a paper about QBIC [14]. But sets are also scored lists, where values are 1 ('in the set') or 0 ('not in the set'). So Laura's problem resolves to combining scored lists.

How do we make sense of the query (Artist = 'Beatles') $\wedge$ (AlbumColor = 'Red')? Here it is probably a list of albums by the Beatles, sorted by how red they are. What about (Artist = 'Beatles') $\vee$ (AlbumColor = 'Red')? And what about (Color = 'Red') $\wedge$ (Shape = 'Round')?

Vol. 35, No. 7, © The Author(s) 2023. Published by Oxford University Press.
https://doi.org/10.1093/logcom/exad066

This reminded me of real-valued (or 'fuzzy') logic. (A good textbook on fuzzy logic is [16].) In real-valued logic, conjunction ($\wedge$) is often taken to be the min, and disjunction ($\vee$) to be the max (this is in the Gödel real-valued logic). So I told Laura, 'Use real-valued logic.' Certainly (AlbumColor = 'Red'), (Color = 'Red') and (Shape = 'Round') need real-valued logic, since they can take on arbitrary values in [0, 1]. Even (Artist = 'Beatles') can be dealt with in real-valued logic, where the possible values are only 0 and 1. Laura's response to my suggestion to use real-valued logic was, 'I like your solution. But we also need an efficient algorithm that can find the top k results while minimizing database accesses.' I came back, a few days later, with an algorithm, now called 'Fagin's Algorithm' (FA) [15] that finds the top k with only $\sqrt{n}$ database accesses, where n is the number of items in the database, when there are two scored lists to combine. Laura's response was 'Good, that beats linear! But we database people are spoiled, and are used to only $\log n$ accesses. Be smarter and get me a $\log n$ algorithm.' I came back a few days later and told her, 'I proved that you can't do better than $\sqrt{n}$.' Having $\sqrt{n}$ accesses vs. n accesses makes a big difference. Assume for example a database with n =10,000,000 entries. Assume also 1000 accesses per second. The n accesses takes almost 3 hours, whereas $\sqrt{n}$ accesses takes around 3 seconds.

FA works for arbitrary monotone scoring functions (where 'monotone' means that increasing the scores of the arguments cannot decrease the overall score). It was implemented in Garlic (where arbitrary monotone scoring functions were allowed, including not only min and max, but also mean and median), and influenced other IBM products, including Watson Bundled Search System, InfoSphere Federation Server and WebSphere Commerce. The conference paper [9] and the journal version [15] introducing FA have together almost 1500 citations (according to Google Scholar).

In 2001, Amnon Lotem, Moni Naor and I introduced a new algorithm, the 'Threshold Algorithm' (TA) [12], to replace FA. Before TA is presented below, let us clarify the problem that it solves.

The are m attributes (such as Redness and Roundness). Each object in a database has a score x_i for attribute i. The objects are given in m scored lists, one list per attribute. The goal is to find the top k objects according to a monotone scoring function, while minimizing access to the lists.

Let f be the scoring function. Popular choices for f are the min (often used in real-valued logic) and the average. Let $x_1, \ldots, x_m$ be the scores of object R under the m attributes. Then $f(x_1, \ldots, x_m)$ is the overall score of object R. We may sometimes write $f(R)$ to mean $f(x_1, \ldots, x_m)$. A scoring function f is *monotone* if whenever $x_i \leq y_i$ for every i, then $f(x_1, \ldots, x_m) \leq f(y_1, \ldots, y_m)$. It would certainly be peculiar to use a scoring function that is not monotone.

There are two modes of access: sorted (or sequential) access, which can obtain the next object and its score for attribute i (where objects are sorted based on their score for attribute i), and random access, which can obtain the score of object R for attribute i. We wish to minimize total number of accesses. The naïve algorithm retrieves every score of every object, and this is too expensive.

We now give an algorithm (the 'Threshold Algorithm', or TA) for finding the top k objects, when the scoring function is monotone.

- Do sorted access in parallel to each of the m scored lists.
- As each object R is seen under sorted access:
  - Do random access to retrieve all of its scores $x_1, \ldots, x_m$.
  - Compute its overall score $f(x_1, \ldots, x_m)$.
  - If this is one of the top k answers so far, remember it.
- For each list i, let t_i be the score of the last object seen under sorted access.
- Define the threshold value T to be $f(t_1, \ldots, t_m)$. When k objects have been seen whose overall score is at least T, stop.

- Return the top k answers.

The key to TA is the halting rule, which says that when k objects have been seen whose overall score is at least T, stop. We now show correctness of the halting rule.

Suppose the current top k objects have scores at least T (the current threshold). Assume (by way of contradiction) that there are objects R and S where R is unseen, where S is in the current top k, but $f(R) > f(S)$; we shall derive a contradiction. Assume that R has scores $x_1, \ldots, x_m$. Therefore, $x_i \leq t_i$ for every i (since R has not been seen). So $f(R) = f(x_1, \ldots, x_m) \leq f(t_1, \ldots, t_m) = T \leq f(S)$. (The first inequality holds by monotonicity of f.) Thus, $f(R) \leq f(S)$. This is a contradiction, since $f(R) > f(S)$.

We now define the notion of *instance optimality*. Let $\mathcal{A}$ be a class of algorithms, and let $\mathcal{D}$ be the class of legal inputs. For A in $\mathcal{A}$ and D in $\mathcal{D}$, the cost of running algorithm A on input D is denoted $\text{cost}(A, D)$, which is non-negative. An algorithm A in $\mathcal{A}$ is *instance optimal* over $\mathcal{A}$ and $\mathcal{D}$ if there are constants c_1 and c_2 such that for every algorithm A' in $\mathcal{A}$ and input D in $\mathcal{D}$ we have $\text{cost}(A, D) \leq c_1 \text{ cost}(A', D) + c_2$. The constant c_1 is called the *optimality ratio*.

Intuitively, for an instance-optimal algorithm A, the adversary can pick her own input D and her own algorithm A', which may be fine-tuned to working well on D, and yet the instance-optimal algorithm A does essentially as well (up to a constant multiple) on D as A'.

We now discuss optimality versus instance optimality. Intuitively, instance optimality corresponds to optimality in every instance, as opposed to just the worst case or the average case. There are many algorithms that are optimal in a worst-case sense, but are not instance optimal. An example is binary search. In the worst case, binary search requires $\log(n)$ probes. However, when there is a positive answer (the search-for value is present), the adversary can obtain this positive answer in one probe.

We now consider the issue of whether TA is instance optimal. The intuition as to why it is instance optimal is that an algorithm cannot stop any sooner, since the next object to be explored might have the threshold value. But life is a bit more delicate.

Let us define a *wild guess* to be a random access for an attribute i of object R where R has not been sequentially accessed before. Neither FA nor TA use wild guesses. In fact, a subsystem might not allow wild guesses. The following theorem is proven in [12].

THEOREM.
Assume that the aggregation function f is monotone. Let $\mathcal{D}$ be the class of all databases. Let $\mathcal{A}$ be the class of all algorithms that correctly find the top k answers for f for every database and that do not make wild guesses. Then TA is instance optimal over $\mathcal{A}$ and $\mathcal{D}$.

An amusing side remark is that when I showed this result to the late David Johnson, he said that I should not call an algorithm instance optimal unless it not only satisfies the rules given for instance optimality, but in addition has the minimal possible optimality ratio. This inspired me to calculate the optimality ratio of TA: it turns out to be $m + m(m-1)c_R/c_S$, where m is the number of attributes, c_R is the cost of a random access and c_S is the cost of a sequential access. And I was able to show this indeed is best possible!

When I told Laura about TA, I explained that it is even better than FA, since it is optimal in a stronger sense. Laura's response was, 'But Ron—you told me that your algorithm is optimal!' I told her, 'Well, Laura, there is optimal, and then there is optimal.'

We submitted the paper to PODS '01. I was worried that the Threshold Algorithm was so simple that the paper would be rejected. So I called it a 'remarkably simple algorithm'. And it worked—the paper won the PODS Best Paper Award! The paper was very influential. It has over 2600 citations, and it won the PODS Test of Time Award in 2011. Because of FA and TA, I won the IEEE Technical

Achievement Award in 2011. Most importantly, in 2014, the paper won the Gödel Prize, the top prize for a paper in theoretical computer science. In 2016, the paper was selected in the first set of 'Gems of PODS'.

Bottom line about this work: a paper that arose by resolving a practical problem won the Gödel Prize!

2 Case study #2: Clio (2003)

After the Garlic project, Laura Haas started a new project, called Clio [17], to do data exchange, where data is converted from one format to another. For example, it might convert XML to relational, or one relational database to another with different column headings.

Because things had gone so well for me in Garlic, I followed Laura, and attended Clio meetings for a year. Then four of us (Phokion Kolaitis, Lucian Popa, Renée Miller and I) decided that we should start from scratch and lay the foundations for data exchange [10].

As an example of converting one relational database to another with different column headings, assume that the Alpha company has an Employee database with a relation that has two columns: EMP (which includes names of employees) and MGR (which gives the manager of the employee). Assume that Alpha merges with another company Beta, and that the schema of the Employee database of Beta has two relations: one with columns EMP and DEPT (which tells the department of the employee), and the other with columns DEPT and MGR (which tells the manager of the department). So Alpha decides to modify its database schema to match that of Beta. There is a relationship between the source schema and target schema (the 'schema mapping') specified by a tuple-generating dependency (tgd). Such tgds were used in the past to help specify 'normal forms' for relational databases [8]. If EM is the Alpha Employee-Manager relation, and if ED is the Beta Employee-Department relation and DM is the Beta Department-Manager relation, the tgd would be

$$EM(e, m) \Rightarrow \exists d(ED(e, d) \wedge DM(d, m)). \tag{1}$$

Let's say that the Alpha database relation (with column EMP and MGR) has the following tuples: (Gödel, Hilbert), (Turing, Hilbert) and (Hilbert, Gauss). When converted to the Beta format, what should the result be?

We now mention three possibilities. The first possibility is to name the department after the manager. So the new ED relation would have tuples (Gödel, Hilbert), (Turing, Hilbert) and (Hilbert, Gauss), and the new DM relation would have tuples (Hilbert, Hilbert) and (Gauss, Gauss).

The second possibility is to introduce new null values d_1 and d_2. Then the new ED relation would have tuples (Gödel, d_1), (Turing, d_1) and (Hilbert, d_2), and the new DM relation would have tuples (d_1, Hilbert) and (d_2, Gauss).

The third possibility is to consider the possibility that Hilbert is the manager of more than one department, and introduce new null values d_1, d_2 and d_3, Then the new ED relation would have tuples (Gödel, d_1), (Turing, d_2) and (Hilbert, d_3), and the new DM relation would have tuples (d_1, Hilbert), (d_2, Hilbert) and (d_3, Gauss).

Which solution should we take? A 'universal solution' [10] is one that is as general as possible; it has a homomorphism into each solution. We prefer the third solution, since it is universal. When the late Hector Garcia-Molina asked me what was wrong with the second solution, I told him that with the second solution, Gödel and Turing are in the same department. But that need not be the case. They could be in different departments, each managed by Hilbert.

On the other hand, perhaps we have a target constraint specified by the following equality-generating dependency (egd):

$$DM(d, m) \wedge DM(d', m) \Rightarrow (d = d').$$

This egd says that each manager manages at most one department. If this egd is a target constraint, then the second solution is universal.

How do we obtain a universal solution? There is a well-known mechanical procedure called the 'chase', originally used as a tool in database design [1]. We use the chase to generate the target from the source efficiently. For example, consider the tgd (1). From EM(Gödel, Hilbert), create ED(Gödel, d) and DM(d, Hilbert), where d is a newly introduced 'labeled null'. The egds tell when to equate labeled nulls.

A question arose as to how to compose schema mappings. For example, assume that we have a schema mapping Σ_{12} that converts schema S_1 to schema S_2, a schema mapping Σ_{23} that converts schema S_2 to schema S_3, and we wish to obtain the result of composing Σ_{12} and Σ_{23}, to obtain a schema mapping that converts schema S_1 directly to schema S_3. Phokion Kolaitis, Lucian Popa, Wang-Chiew Tan and I studied composition of schema mappings [11]. To our surprise, we found that composition can take us out of first-order logic! We found the right language for composition ('second–order tgds'). And we gave an algorithm for composition.

We now mention some measures of success for our work on data exchange. It is used in IBM's DB2 Control Center, Rational Data Architect and Content Manager: they use universal solutions, they use our algorithm to produce a universal solution and they use our algorithm to compose schema mappings.

Our initial paper [10] on data exchange won the 2013 International Conference on Database Theory (ICDT) Test of Time Award. We were told that our initial paper was the second most highly cited paper of the decade in the journal TCS (the most highly cited was a survey paper). This paper [10] has over 1700 citations. Our paper [11] on composition of schema mappings won 2014 PODS Test of Time Award, and our follow-up composition paper (with Arenas and Nash) [2] won the 2010 ICDT Best Paper Award.

This work created a new subfield: special sessions on data exchange arose in every major database conference. For the 2003 initial paper and 2004 composition paper, we won the 2020 Alonzo Church Award (the highest award for logic and computation) 'for an outstanding contribution represented by a paper or small group of papers within the past 25 years'.

3 Case study #3: Real-Valued Logics (2020)

In classical logics, formulas take on either value 0 ('False') or 1 ('True'). In real-valued (or 'fuzzy') logics, formulas can take on arbitrary real values (typically restricted to being in [0,1]). Our work on real-valued logics arose as part of the LNN (Logical Neural Nets) project at IBM [20]. The inputs to, say, an 'and' gate could each be any number in [0,1]. The system builders of LNN wanted a sound and complete axiomatization for real-valued logic, so that they could arrive at truth values given other truth values whenever possible. Our recent work [13], which we shall now describe, gives a sound and complete axiomatization that, by being parametrized, covers a large class of real-valued logics, including all of the most common ones. It also allows weights, where the importance of some subformulas can be greater than that of other subformulas.

The two most common real-valued logics are Gödel logic and Łukasiewicz logic. Assume that σ_1 has real value s_1, and σ_2 has real value s_2. In Gödel logic, the conjunction $\sigma_1 \wedge \sigma_2$ has value

$\min\{s_1, s_2\}$, the disjunction $\sigma_1 \vee \sigma_2$ has value $\max\{s_1, s_2\}$, and the negation $\neg\sigma_1$ has value 1 if $s_1 = 0$ and 0 otherwise. In Gödel logic, conjunction and disjunction are defined just as in Zadeh's original fuzzy logic [22], but negation is defined differently, where Zadeh defined $\neg\sigma_1$ to have value $1 - s_1$. In Łukasiewicz logic, the conjunction $\sigma_1 \wedge \sigma_2$ has value $\max\{0, s_1 + s_2 - 1\}$, the disjunction $\sigma_1 \vee \sigma_2$ has value $\min\{1, s_1 + s_2\}$ and the negation $\neg\sigma_1$ has value $1 - s_1$.

Not surprisingly, there are anomalies in real-valued logic. The most well-known theorem dealing with this issue is the Bellman-Giertz Theorem [4] (as modified by Yager [21] and by Dubois/Prade [6]), given below:

THEOREM.
Assume that we want the property that whenever σ_1 and σ_2 are formulas involving only conjunction and disjunction (in particular, no negation), then σ_1 and σ_2 are equivalent in ordinary propositional logic if and only if they always take on the same the same value in the real-valued logic. Then in the real-valued logic, conjunction and disjunction must be given by min and max, respectively.

For example, the conjunction $x \wedge x$ equals equals x in Gödel logic (where conjunction is given by the min), but not in Łukasiewicz logic. Thus, let $x = 0.6$. In Łukasiewicz logic, the value of $x \wedge x$ is $\max\{0, 0.6 + 0.6 - 1\}$, which equals 0.2, not 0.6.

Although Gödel logic preserves equivalence in formulas not involving negation, it does not preserve equivalence in formulas involving negation. Thus, $x \vee \neg x$ is not a valid formula in Gödel logic: if $x = 0.5$ then $\neg x$ has value 0, so $x \vee \neg x$ has value $\max\{0.5, 0\} = 0.5$, not 1.

In [13], certain sentences about real-valued formulas and their values are introduced. What sentences should be allowed? Let F be the set of formulas (such as $\sigma_1 \vee \sigma_2$, where σ_1 and σ_2 are formulas). Let θ be a real number in [0,1]. A natural choice is (σ, θ), where σ is a formula in F. Intuitively, (σ, θ) says that the real value of σ is θ. But this has expressivity problems. What does $(\sigma_1 \vee \sigma_2, 0.6)$ imply about sentences (σ_1, θ), where $\vee$ is the max? It implies an infinite disjunction $\vee_{\{\theta:\, \theta \leq 0.6\}}(\sigma_1, \theta)$. This suggests allowing sentences (σ, S), where $S \subseteq [0, 1]$. Then the infinite disjunction becomes $(\sigma_1, [0, 0.6])$. But even these sentences $(\sigma; S)$ are not sufficiently expressive, as we shall discuss shortly. We note that formulas equivalent to $(\sigma; S)$ have been considered in the literature [7, 18] in the special case where S is an interval.

So, there is a need to enrich the class of sentences. Write $(\sigma_1, \sigma_2; S)$ to mean that if s_1 is the value of σ_1, and s_2 is the value of σ_2, then $(s_1, s_2) \in S$. Let $f_\vee(s_1, s_2)$ be the value of $\sigma_1 \vee \sigma_2$ when the value of σ_1 is s_1, and the value of σ_2 is s_2. For example, in Łukasiewicz logic, $f_\vee(s_1, s_2) = \min(1, s_1 + s_2)$. Let $S' = \{(s_1, s_2) : f_\vee(s_1, s_2) = 0.6\}$. Then what we can infer from $(\sigma_1 \vee \sigma_2, \{0.6\})$ about the values of σ_1 and σ_2 is exactly $(\sigma_1, \sigma_2; S')$. It is shown in [13] that there is a sentence $(\sigma_1, \sigma 2; S'')$ that is not equivalent in Łukasiewicz logic or Gödel logic to any sentence of the form $(\sigma; S)$. This shows the inadequacy of sentences of the form $(\sigma; S)$.

In [13], a sentence is taken to be of the form $(\sigma_1, \ldots, \sigma_k; S)$. This sentence says that if s_i is the value of σ_i, for $1 \leq i \leq k$, then $(s_1, \ldots, s_k) \in S$.

We now give the formulation of [13]. We assume a finite set of primitive propositions. A *model* M is a function that assigns a value $M(A) \in [0, 1]$ to each primitive proposition A. Let us hold fixed a real-valued logic. Then for each model M and for each of our sentences σ, the sentence σ is either true or false. In the former case, we say that M is a model of σ. If Σ is a set of sentences, then we say that M is a model of Σ if M is a model of each member or Σ.

Some examples of the inference rules of [13] are:

From $(\sigma_1, \ldots, \sigma_k; \mathrm{S})$ infer $(\sigma_1, \ldots, \sigma_k; S')$ if $\mathrm{S} \subseteq S'$.

From $(\sigma_1, \ldots, \sigma_k; S_1)$ and $(\sigma_1, \ldots, \sigma_k; S_2)$ infer $(\sigma_1, \ldots, \sigma_k; S_1 \cap S_2)$.

We now give an inference rule that depends on the real-valued logic under consideration. For each real-valued binary connective α, let $f_\alpha(s_1, s_2)$ be the value of $\sigma_1 \,\alpha\, \sigma_2$ when the value of σ_1 is s_1 and the value of σ_2 is s_2. For example, in Gödel logic, $f_\wedge(s_1, s_2) = \min\{s_1, s_2\}$. For each real-valued unary connective ρ, let $f_\rho(s)$ be the value of $\rho\sigma$ when the value of σ is s. For example, in Łukasiewicz logic, $f_\neg(s) = 1 - s$.

In the sentence $(\sigma_1, \ldots, \sigma_k; S)$, let us say that the tuple $(s_1, \ldots, s_k)$ in S is *good* if (a) $s_m = f_\alpha(s_i, s_j)$ whenever σ_m is $\sigma_i \,\alpha\, \sigma_j$ and α is a binary connective (such as $\wedge$), and (b) $s_j = f_\rho(s_i)$ whenever σ_j is $\rho\sigma_i$ and ρ is a unary connective (such as $\neg$). Note that being 'good' is a local property of a tuple s in S (i.e. it depends only on the tuple s and not on the other tuples in S). Of course, if the real-valued logic under consideration has higher-order connectives (ternary, etc.), then we would modify the definition of a good tuple in the obvious way.

We then have the following inference rule, which is the key inference rule of [13].

From $(\sigma_1, \ldots, \sigma_k; \mathrm{S})$ infer $(\sigma_1, \ldots, \sigma_k; S')$ when S' is the set of good tuples of S.

Let γ_1 be the premise $(\sigma_1, \ldots, \sigma_k; S)$ and let γ_2 be the conclusion $(\sigma_1, \ldots, \sigma_k; S')$ of this inference rule. Then γ_1 and γ_2 are logically equivalent, and S' is as small as possible so that γ_1 and γ_2 are logically equivalent.

A simple example of a valid sentence is $(A, B, A \vee B; S)$ where $S = \{(s_1, s_2, s_3)\colon\ s_1 \in [0,1], s_2 \in [0,1], s_3 = f_\vee(s_1, s_2)\}$. This is derived from the valid sentence $(A, B, A \vee B; [0,1]^3)$ by applying the key inference rule.

An axiomatization is *finite-strongly complete* if whenever Γ is a finite set of sentences and τ is a single sentence that is a logical consequence of Γ (i.e. every model of Γ is a model of τ) then there is a proof of τ from Γ. An axiomatization is *weakly complete* if the above holds when Γ is the empty set. Thus, an axiomatization is weakly complete if there is a proof of each valid sentence. An axiomatization is *sound* if whenever there is a proof of τ from Γ, then Γ logically implies τ.

THEOREM.
The axiomatization of [13] is sound and finite-strongly complete.

Thus, the axiomatization allows us to derive exactly what information can be inferred about the combinations of values of a collection of formulas given information about the combinations of values of several other collections of formulas.

For various real-valued logics in the literature, there were sound and complete axiomatizations (some finite-strongly complete, some weakly complete). By being parametrized, the axiomatization in [13] is the first that works simultaneously for a large class of real-valued logics, including all of the most common ones.

We now discuss making use of weights. In LNN's we want to weight the importance of the subformulas. For example, for the formula $\sigma_1 \vee \sigma_2$, we may want to assign weight w_1 to σ_1, and weight w_2 to σ_2. In Łukasiewicz logic, if the value of σ_1 is s_1, and the value of σ_2 is s_2, we may take the weighted value of $\sigma_1 \vee \sigma_2$ to be $\min\{1, w_1 s_1 + w_2 s_2\}$.

The axiomatization in [13] extends easily to include weights. For example, instead of using $f_\vee(s_1, s_2)$, we would use $f_\vee(s_1, s_2, w_1, w_2)$.

THEOREM.
The axiomatization of [13] that includes weights is sound and finite-strongly complete.

With Ryan Riegel, we developed an efficient decision procedure for deciding implication in Łukasiewicz and Gödel logics in important special cases. Along with the axiomatization, this decision procedure appears in a joint paper that Ryan Riegel and I wrote with Alexander Gray [13].

Guillermo Badia, Carles Noguera and I [3] extended the sound and finite-strongly complete axiomatization to cover not just propositional real-valued logic, but also first-order real-valued logic.

4 Conclusions

There are conclusions to be drawn from these case studies: conclusions for both system builders and theoreticians.

For system builders, the main message is to consult with theoreticians, for multiple reasons:

- Explaining the problem is useful by itself.
- Principled approaches can improve your product.
- Better or new algorithms can differentiate your product.
- Algorithm analysis can provide performance expectations and provide product guarantees.
- Abstractions can expand the function of your product.

For theoreticians, the main message is that involvement with system builders can help your theory, for multiple reasons:

- Novel questions will be asked.
- New models and new, interesting areas of study will arise.
- Implementation can reveal weaknesses in the theory.
- The theory will be relevant.
- And most important, you will have **practical impact**!

Acknowledgements

The author is grateful to Sasha Rubin for very helpful comments and suggestions that improved the presentation of this paper.

References

[1] A. Aho, C. Beri and J. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems (TODS)*, **4**, 297–314, 1979.

[2] M. Arenas, R. Fagin and A. Nash. Composition with target constraints. In *Logical Methods in Computer Science*, vol. 7, pp. 1–38, 2011.

[3] G. Badia, R. Fagin and C. Noguera. New foundations of reasoning via real-valued first-order logics. arXiv preprint, arXiv:2207.00086, 2023.

[4] R. Bellman and M. Giertz.. On the analytic formalism of the theory of fuzzy sets. In *Information Sciences*, vol. 5, pp. 149–156, 1973.

[5] M. Carey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Tholmas, J. Williams and E. Wimmers. Towards heterogeneous multimedia information systems: the garlic approach. In *RIDE-DOM '95 (5th Int'l Workshop on Research Issues in Data Engineering: Distributed Object Management)*, pp. 124–131, 1995.

[6] D. Dubois and H. Prade. Criteria aggregation and ranking of alternatives in the framework of fuzzy set theory. In *Framework of Fuzzy Set Theory, in Fuzzy Sets and Decision Analysis*, H. J. Zimmermann, L. A. Zadeh and B. Gaines., eds, pp. 209–240. TIMS Studies in Management Sciences, 1984.

[7] F. Esteva, L. Godo and E. Marchioni. Fuzzy logics with enriched language. In *Chapter VIII of the Handbook of Mathematical Fuzzy Logic*, P. Cintula, P. Hájek and C. Noguera., eds, vol. 2, pp. 627–711. College Publications, 2011.
[8] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, **2**, 262–278, 1977.
[9] R. Fagin. Fuzzy queries in multimedia database systems. In *ACM Symposium on Principles of Database Systems (PODS)*, pp. 1–10, 1998.
[10] R. Fagin, P. G. Kolaitis, R. J. Miller and L. Popa. Data exchange: semantics and query answering. In *International Conference on Database Theory (ICDT)*, pp. 207–224, 2003.
[11] R. Fagin, P. G. Kolaitis, L. Popa and W.-C. Tan. Composing schema mappings: second-order dependencies to the rescue. In *ACM Symposium on Principles of Database Systems (PODS)*, pp. 83–94, 2004.
[12] R. Fagin, A. Lotem and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, **66**, 614–656, 2003.
[13] R. Fagin, R. Riegel and A. Gray. Foundations of reasoning with uncertainty via real-valued logics. arXiv preprint, arXiv:2008.02429v2, 2023.
[14] R. Fagin and L. Stockmeyer. Relaxing the triangle inequality in pattern matching. *International Journal of Computer Vision*, **30**, 219–231, 1993.
[15] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, **58**, 83–99, 1999.
[16] P. Hájek. *Metamathematics of Fuzzy Logic*, vol. 4. Springer Science & Business Media, 1998.
[17] M. A. Hernández, R. J. Miller, L. Haas, L. Yan, C. T. H. Ho and X. Tian. Clio: a semi-automatic tool for schema mapping. In *System Demonstration. ACM SIGMOD International Conference on Management of Data (SIGMOD)*, **30**, 2001.
[18] N. Murray and V. Rosenthal. Signed formulas: a liftable meta-logic for multiple-valued logics. In *International Symposium on Methodologies for Intelligent Systems*, pp. 275–284. Springer, 1993.
[19] W. Niblack, R. Barberand, W. Equitz, M. Flickner, E. Glasman, D. Petkovic and P. Yanker. The QBIC project: querying images by content using color, texture and shape. In *Proc. Conf. On Storage and Retrieval for Image and Video Databases, San Jose, CA, SPIE*, vol. 1908, pp. 173–181, 1993.
[20] R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, S. Ikbal, H. Karanam, S. Neelam, A. Likhyani and S. Srivastava. Logical neural networks. arXiv preprint, arXiv:2006.13155, 2020.
[21] R. Yager. Some procedures for selecting fuzzy set-theoretic operations. *International Journal General Systems*, **8**, 115–124, 1982.
[22] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, **30**, 407–428, 1975.

Received 26 April 2023