Constraint databases: A tutorial introduction[*]

Jan Van den Bussche
Limburg University, Belgium

1 Introduction

We give a tutorial introduction to the basic definitions surrounding the idea of constraint databases, and survey and indicate some of the achieved research results on this subject. This paper is not written as a scholarly piece, nor as polished course notes, but rather as something like the transcript of an invited talk I gave at a meeting bringing together researchers from finite model theory, database theory, and computer-aided verification, which was held at Schloss Dagstuhl in October 1999.

Very recently a great, nicely polished, book came out with all the details, and covering the state of the art in constraint databases up to, say, mid 1999 [20]. I recommend it! You should see this paper merely as an appetizer for the book. I will also not try to be complete in my bibliographical references. Again, see the book for that.

2 Genesis and growth of the idea

In 1988, Paris Kanellakis visited IBM Watson Research Center. At that time, a group of people around Jean-Louis Lassez back there were active in the subject of *constraint logic programming* [8]. Constraint logic programming is a generalization of standard logic programming. Recall that standard logic programming has *matching* as a basic operation. For example, if we have a rule

$$R(2, x, y) \leftarrow S(x, y)$$

and a goal

$$R(z, u, u),$$

then the goal will match with the head of the rule by the matching

$$z = 2 \text{ and } u = x = y,$$

yielding a new goal

$$S(u, u).$$

In constraint logic programming, one starts from the observation that matching is only a very simple kind of *constraint solving*, and generalizes standard logic programming by replacing matching by constraint solving. For example, in the system CLP($\Re$) developed at IBM Watson, you can program with linear inequality constraints over the reals. So, if you match a rule

$$R(x, y) \leftarrow x < y$$

and a goal

$$R(2z, z),$$

the new goal becomes

$$2z < z,$$

which the system can "solve" as $z < 0$.

In view of the applicability of standard logic programming to databases by using Datalog programs as a query language [25], Paris's insight was to try to do the same with constraint logic programming. Development of this idea with Gabi Kuper and Peter Revesz lead to a seminal paper entitled "Constraint Query Languages" at PODS'90, which introduced the idea of constraint databases (a full journal version appeared later [18]).

In the '90s, the topic was vigorously taken up and pursued by a relatively small group of enthusiasts, mostly from the database theory community. As a result, currently, 10 years later, constraint databases have become a relatively mainstream database research topic, with practical as well as theoretical motivations, as should be the case with any good research topic.

3 Classical database theory: The Universe of Discourse neglected

In our tutorial introduction to the notion of a constraint database, let us begin by reviewing rela-

[*]Database Principles Column.
Column editor: Leonid Libkin, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 08974. E-mail: libkin@research.bell-labs.com.

tional databases as they are classically formalized in database theory.

A relational database is traditionally formalized as a collection of finite relations over some universe $\mathbf{U}$ of atomic values. This "Universe of Discourse" typically has a *structure* of its own. For example, you will have numbers in it, with predicates like $<$ and functions like $+$ defined on them, and strings, with predicates such as `like` in SQL and operations such as concatenation. In SQL we routinely write queries like

> **select** $x + y$
> **from** R
> **where** $x < y$.

However, this structure on $\mathbf{U}$ has often been neglected in classical database theory. For example, it is no problem to define the relational algebra or calculus (mathematically, first-order logic) *with* interpreted functions and predicates, but in research on the theory of query languages these were often left out [1]. (Here it has to be said, to be fair, that already without them a lot of difficult issues arose, and it made sense to try to solve these issues first without the added complications arising from interpreted functions and predicates.)

4 A direct approach

A direct approach to formalize databases and query languages in the presence of structure on the universe $\mathbf{U}$ is the following. Consider $\mathbf{U}$ as a *structure* in the sense of mathematical logic. So it is a (typically infinite) set equipped with predicates and functions on that set. We will hopefully not confuse the reader by denoting just the set by $\mathbf{U}$, and also the whole structure by the same symbol $\mathbf{U}$. Now we define a relational database simply as an *expansion* of the structure $\mathbf{U}$ with a finite number of additional, *finite* relations on $\mathbf{U}$.

For example, $\mathbf{U}$ could be the following structure:

$$\mathbf{U} = (\mathbf{U}; \mathit{Number}, <, +, \mathit{String}, \texttt{like}, \texttt{concat}).$$

Here, imagine that the set $\mathbf{U}$ is the union of the set of all integers and the set of all strings over the ASCII alphabet. The unary predicate *Number* is true of the integers and false of the strings, and the unary predicate *String* is true of the strings and false of the integers. The predicates $<$ and `like` and the functions $+$ and `concat` were already mentioned above. Then a database $\mathbf{D}$ with three relations R, S, T will be an expansion of $\mathbf{U}$, and thus be of the form

$$(\mathbf{U}; R, S, T).$$

The relation R in $\mathbf{D}$ could contain, for example the two tuples

$$(\text{john}, 30) \text{ and } (\text{mary}, 25),$$

among others.

We can now continue to use the popular logic-based query languages such as first-order logic (FO), Datalog, etc. The structure on $\mathbf{U}$ is now automatically fully visible to queries expressed in these languages, because it is part of the databases themselves in the way we formalized databases. For example, the SQL query above is written in FO as follows:

$$\{x + y \mid R(x, y) \wedge x < y\}.$$

5 Two immediate problems with the direct approach

The universe $\mathbf{U}$ is usually infinite, and is now formally the domain of any database, since we defined a database as an expansion of $\mathbf{U}$. This implies, however, that the variables in our logic formulas range over the whole of $\mathbf{U}$. Consequently, the result of a query may well be *infinite*, as in the following example:

$$\{x + y \mid S(x) \vee T(y)\}.$$

This is, of course, nothing but the classical *safety problem* we all know and love, and occurs as well without structure on $\mathbf{U}$, as in the following example:

$$\{(x, y) \mid S(x) \vee T(y)\}.$$

Even without an infinite result, we still have the problem of *effective evaluation*: how are we going to compute the result of a query if variables range over an infinite universe?

6 Definable relations

We can deal with the problem of infinite results as follows. We begin by noting that an FO query might just query the universe, without looking at the database relations at all. For example, on a universe consisting of integers, the query

$$\{x \mid \exists y\, x = y + y\}$$

retrieves the infinite set of all even integers. Relations defined by FO formulas that look only at the predicates and functions of the universe, not at database relations, are called *definable relations*. We will represent such definable relations, which will often be infinite, simply by their defining formulas. This *symbolic representation* is a first major idea in constraint databases.

7 Constraint databases: First definition

In databases we usually insist that output relations can later be used as input relations. This principle of compositionality, or *closure* as we often call it in databases, is important for example in views.

Hence, we should allow, instead of only finite relations over **U**, *any definable relations over* **U** in a database! Again, such relations will be represented symbolically by their defining formulas. This idea of a database consisting of definable relations represented by formulas is a second major idea in constraint databases.

So we come to our first definition of what a constraint database is (we will later refine it). A constraint database over **U** is a finite collection of first-order formulas over **U**:

$$(\varphi_R, \varphi_S, \varphi_T, \dots).$$

Each formula defines a (possibly infinite) relation over **U**:

$$(R, S, T, \dots).$$

The constraint database *represents* the infinite structure obtained from expanding the structure of the universe **U** with these relations, for example:

$$(\mathbf{U}; <, +, ; R, S, T).$$

We should point out at this point that classical finite relation databases can be viewed as special cases of constraint databases, as follows. The only structure on **U** that we have are constant symbols for all elements of **U**, and the equality predicate of course.[1] A finite relation like, for example, R:

john	30
mary	25

can then always be defined by a formula, for example φ_R:

$$\begin{gathered}(x_1 = \text{john} \wedge x_2 = 30)\\ \vee(x_1 = \text{mary} \wedge x_2 = 25)\end{gathered}$$

8 Plug-in evaluation

How do we solve the problem of effective evaluation described above? Answer: we simply don't evaluate! All we do is plug in the definitions of the database relations in the query formula. For example, if we have the query

$$\forall x((\exists y\, x = y + y) \rightarrow S(x))$$

on a database where relation S is defined by formula φ_S, we plug this formula in the query formula and obtain a definition of the result of the query on that database:

$$\forall x((\exists y\, x = y + y) \rightarrow \varphi_S(x))$$

Note that this is a formula over **U** only: it does not mention any database relations, and thus is indeed the symbolic representation of a definable relation as we introduced above, and as we agreed to represent symbolically using formulas. Note also that the query was actually a yes/no query, and correspondingly, our result formula is a sentence (formula without free variables) which will, on **U** evaluate to either true or false, as desired.

9 Testing emptiness of definable relations

So far, so good. It is all well to just use formulas over **U** as symbolic representations of possibly infinite, definable relations over **U**. But what can we do with these representations? Can we compute with them? For one thing, can we even decide whether the relation defined by a given formula over **U** is empty or not?

We can do this if the first-order theory of **U**, that is, the set of all sentences over **U** that evaluate to true on **U**, is a decidable set. There are many examples of structures that do *not* have a decidable theory. Two famous examples are the natural numbers with plus and times: $(\mathbb{N}; +, \times)$, and the strings over some alphabet Σ with the single-letter strings as constants and concatenation as function: $(\Sigma^*; (a)_{a\in\Sigma}, \texttt{concat})$.

However, there are also a few structures that *do* have a decidable theory:

- $(\mathbb{Z}; +, 0, 1, <)$: the integers with their order and addition, but without multiplication;
- the same with the rationals $\mathbb{Q}$ instead of the integers $\mathbb{Z}$;
- $(\mathbb{R}; +, \times, 0, 1, <)$: the reals, even *with* multiplication!
- The algebra of all terms built using a given signature of function symbols;
- Boolean algebras.

[1]Recall that in logic, constant symbols are 0-ary function symbols.

It is with such universes that we can play the game of constraint databases.

Of course there is also the issue of computational complexity. Typically, decidable theories have a huge complexity: it is typically not easy to determine the truth of a sentence. However, this complexity can often be isolated in the *number of quantifiers of the sentence*. For example, Basu's algorithms for the theory of the reals [2] split up into a combinatorial part, which deals with the actual polynomials occurring in the formula (obtained by applying plus and times to variables), and in an algebraic part, which deals with the actual number of variables and quantifiers, the degree of the polynomials, etc. It is only in the algebraic part that the algorithms are exponential. The combinatorial part is polynomial-time. In constraint databases, we would like similarly to keep the algebraic part really simple so that the algorithms have a better chance of being efficient in their totality. This brings us to the next point.

10 Quantifier elimination

The method of quantifier elimination is an old technique from the field of model theory in mathematical logic [11, 16]. When we try to apply it to a structure **U**, we try to express every formula over **U** as a boolean combination of "base formulas". If we can do with just the atomic formulas as base formulas, then we say that "**U** *has* quantifier elimination": in this case every formula can be written as a quantifier-free one. Often, however, the atomic formulas alone are not enough, and other formulas must be used as base formulas as well.

So, the idea of applying quantifier elimination to **U** amounts to expand **U** with suitable definable functions or predicates, until we actually have quantifier elimination. For example, take the structure $(\mathbb{Z}; +, 0, 1, <)$. We cannot express the formula

$$\exists x\, y = x + x + x + 2$$

without quantifiers, but if we add all modulo functions, we can: then it becomes equivalent to the quantifier-free formula

$$y \bmod 3 = 2.$$

The method of quantifier elimination can be successfully applied to all the structures we mentioned above with a decidable theory.

11 Constraint databases: Second definition

We are now ready for an improvement of our first definition of what constraint databases are. The idea is to use only structures **U** with a decidable theory, and which have quantifier elimination. Formulas in the constraint databases can then without loss of expressive power be required to be *quantifier free*. Query evaluation is still plug-in evaluation as explained earlier, but the result of the plug in still has the quantifiers from the query formula. We eliminate these (remember that we now assume that the universe has quantifier eliminiation), and the resulting quantifier-free formula is the real result of the query.

Working only with quantifier-free formulas helps tremendously in keeping the complexity of testing emptiness of our definable relations down. Although quantifier elimination algorithms typically have a very high computational complexity, we now have a handle to control this complexity. Indeed, as we just saw, we only have to eliminate the quantifiers that were in the query, and the query is fixed, regardless of how large the database is to which we apply it. So, the number of quantifiers to be eliminated is fixed, and hence complexity stays manageable.

We should immediately admit, however, that we are painting here a very optimistic picture of constraint query evaluation. In practice we still have to worry about other things, such as the sizes of the logical terms resulting from quantifier elimination, which could also become very big. But, it remains indisputable that having only quantifier-free formulas in the database helps a lot.

12 Research in constraint databases

Research in constraint databases could be classified as follows:

1. Classical database theory topics reconsidered, now that the universe has a structure. So here we are dealing with the special case of *finite* databases over **U**.

2. New research topics made possible by the new possibilities given by full constraint databases, notably, to represent infinite relations.

However, numerous links between 1 and 2 exist. Sometimes we have to do type-1 research before we even can begin to do type-2 research.

In the remainder of this paper I will give some samples of the two kinds of research in constraint databases.

13 O-minimal structures

If we know something more about our universe $\mathbf{U}$, apart from what we already agreed on (decidable theory, quantifier elimination), we can obtain further results. One nice property is that of *o-minimality.* This property is an abstraction of the nice properties the universe of the reals has in connection with the "tame" topological behaviour of definable sets in the reals [26].

Concretely, a structure $\mathbf{U}$ is called *o-minimal* if it is *ordered*: one of its predicates is a total ordering on $\mathbf{U}$, and in addition the following holds:

1. Take a formula over $\mathbf{U}$:
$$\varphi(x_1, \dots, x_k).$$
2. Substitute values $a_2, \dots, a_k$ for $x_2, \dots, x_k$. Note that we leave x_1 free!
3. Then the set
$$\{x_1 \in \mathbf{U} \mid \varphi(x_1, a_2, \dots, a_k) \text{ true in } \mathbf{U}\}$$
is a finite union of *intervals* in the given order on $\mathbf{U}$ (where we agree that an isolated point is also an interval).

The prototypical example of an o-minimal structure is that of the reals.

14 Natural-active collapse

One result we obtain when the structure is o-minimal is natural-active collapse. It is a result about finite databases, so this is type-1 research. Consider a finite database over $\mathbf{U}$. We all know what its *active domain* is: it is the finite set of atomic elements of $\mathbf{U}$ actually occurring in the relations in the database.

Recall that when we write a query such as
$$\exists r \forall x \forall y (R(x, y) \rightarrow x^2 + y^2 = r^2),$$
the quantifiers range over the whole of $\mathbf{U}$. For example, if $\mathbf{U}$ is the reals, this query says that all pairs (x, y) in relation R lie on a common circle with the origin as center. It is crucial that the quantifier $\exists r$ ranges over *all* reals and not just over the active domain of the input database. Indeed, it is not because all pairs lie on a common circle, that the radius of that circle actually is in the active domain!

However, in this particular example, we can rewrite the query so that it is okay to let quantifiers range over the active domain only. Indeed, we note that the common radius (or rather, its square), if it exists, equals $x_0^2 + y_0^2$ for some arbitrary pair (x_0, y_0) in relation R. So, we can equivalently rewrite the query as follows:
$$\exists x_0 \exists y_0 \forall x \forall y (R(x, y) \rightarrow x^2 + y^2 = x_0^2 + y_0^2),$$
and here it is perfectly okay if the quantifiers range over the active domain only.

Now is this just a peculiar example, or can we *always* pull such a trick? If $\mathbf{U}$ is o-minimal, you can indeed always do this, and there even is an algorithm that will do the rewriting for you [6].

15 Safety

Another issue of type 1, so finite databases only. Let's return to the safety problem: is the result of my query always finite on finite inputs? For example, the following query is not safe if the universe is the rationals or the reals (which are densely ordered):
$$\{z \mid \exists x \exists y (R(x, y) \wedge x < z < y)\}$$
The following query is safe:
$$\{z \mid \exists x \exists y (R(x, y) \wedge z = x + y)\}$$

Recall that we basically dismiss this problem in constraint databases: we simply allow infinite relations (represented symbolically as formulas). But in many applications, especially if the input is finite, it can still be nice if you can somehow guarantee that your output is also finite. Without a structure on $\mathbf{U}$, this is well known: we have the class of "syntactically safe" relational calculus formulas [24]. Can we do something similar with a structure on the universe?

Again under the assumption that $\mathbf{U}$ is o-minimal, the answer is "yes" [7]. One can use special formulas over $\mathbf{U}$ called "algebraic" to specify finite sets of values. We now accompany every FO query with such an algebraic formula, with the following range-restricted semantics: we only evaluate the query within the finite upper bound of values given by the algebraic formula, so that it is enforced to be safe. Then all safe queries can be specified in this way. In other words, if a query happens to be safe, you can always precompute, by an algebraic formula, the finite set of values it will need to form its result relation.

16 Conjunctive query containment

Still on finite databases, another natural question we may ask is whether the classical problem of conjunctive query containment remains decidable when we now have a structure on our universe **U**. Given our basic assumption that the theory of **U** is decidable, the answer is "yes" [17, 3].

Of course conjunctive queries also make perfect sense on general constraint databases with infinite relations. Much less is known about containment testing in this setting, however.

17 Spatial databases as constraint databases

Now some attention to type-2 research. Using the reals as our universe **U**, and using definable k-ary relations over the reals, we get what is known in real algebraic geometry as the "semi-algebraic sets in $\mathbb{R}^k$. This is a very well-behaved class of geometrical figures, and a lot is known about their topological and geometrical properties [10, 4]. So, it seems very natural to use constraint databases over the reals to represent spatial data [22]. So, every spatial datum is a semi-algebraic set, which is symbolically stored as a defining formula over the reals.

More traditional approaches to spatial databases [27, 23] represent spatial data as polyhedral subdivisions of the real space, or using a finite number of spatial abstract datatypes such as point, line segment, polyline, circle, arc segment, etc. This allows for more efficient implementations of specific operations on spatial data. However, elegant, flexible, closed, logical query languages are much harder to get in the traditional approaches than in the constraint approach.

Indeed, consider the following spatial queries:

- Set S of points in the plane has dimension two:

  $$\exists x_0 \exists y_0 \exists \varepsilon > 0 \, \forall x \forall y (d((x,y),(x_0,y_0)) > \varepsilon^2 \rightarrow S(x,y)),$$

  where $d((x,y),(x_0,y_0))$ stands for $(x-x_0)^2 + (y-y_0)^2$ (Euclidean distance).

- Give me the border of S:

  $$\{(x,y) \mid \forall \varepsilon > 0 \, \exists x_1 \exists y_1 \exists x_2 \exists y_2 \; d((x,y),(x_1,y_1)) < \varepsilon \wedge d((x,y),(x_2,y_2)) < \varepsilon \wedge S(x_1,y_1) \wedge \neg S(x_2,y_2)\}$$

- Do S_1 and S_2 overlap? Is simply the same as asking whether their intersection has dimension two (see first query).

- Do S_1 and S_2 "touch"? Is imply the same as asking whether all points on the intersection lie on the borders (see second query).

These examples serve to illustrate that you can already express quite a lot with just the basic operators of first-order logic FO. There is no need to add a special order for interior, for border, for overlap, for touch, and so on, as is the case in more traditional approaches to spatial database query languages.

However, other interesting spatial database queries are not expressible in FO. A case in point is topological connectivity. This inability can be rigorously proved, and the proof is a nice illustration of a link between infinite constraint databases and finite ones. To give an idea of this proof, we have to talk first about another phenomenon which is interesting in its own right.

18 Arithmetical collapse

Let's go back to finite databases for a second. Although we now have structure on our universe **U**, some queries are not interested in this structure, and continue to view the elements in the database as abstract atomic data elements. A simple example is the query on a given finite set "are there at least five elements in this set?" Clearly, for the outcome of this query it does not matter whether the elements are, say, integers on which addition is defined, or the elements are just abstract elements without any interpreted predicates and functions. We call such queries *generic*.

Now suppose we have an FO query which happens to be generic. This query might still use the interpreted predicates and functions in its formula, but that seems not very useful, as the outcome of the query is indifferent to these. This basic intuition can be formally proven: generic FO queries can always be expressed by a classical relational calculus formula that does not mention the structure of **U**, but only the database relations [21]. This is the principle of *arithmetical collapse*.

To be correct we should add that there is one small exception: if the universe if ordered, then we sometimes still need to use the order, even though the query is generic. But that's okay, it won't hurt for what we are going to say next.

19 Why the topological connectivity query is not in FO

Consider finite sets S of, say real numbers (we are back to spatial database queries so the universe is the reals). It turns out to be possible to write an FO query ψ that, for any such input S, outputs an infinite region in the plane that is topologically connected if and only if the cardinality of S is even [15]. Now *suppose* topological connectivity *were* expressible in FO, by some query formula γ. Then the composed query $\psi; \gamma$ clearly is the parity query. The parity query is generic, so by arithmetical collapse and natural-active collapse we can rewrite $\psi; \gamma$ to an FO query that is a standard relational calculus query: it does not use the structure on the reals: addition, multiplication. But this is a contradiction: it is well known that such a standard relational calculus query *cannot* express the parity query [1].

20 Other constraint database research topics

Research topics that were also actively studied, but which we did not touch upon in this tutorial, are the following. See the book [20] for information.

- *Aggregate* operators, such as volume in a spatial context.
- *Linear* constraint databases: we consider only regions definable by formulas that do not use multiplication, only linear equations and inequalities. Some nice theoretical results have been obtained, for example on encoding such databases into finite databases, and it is an important practical case where a rather complete implemented constraint database system exists (the DEDALE system).
- The question of understanding exactly which *topological queries* can be expressed in FO, and how to extend FO towards those we cannot express (two recent papers in this direction not covered by the book are by Grohe and Segoufin [14] and by Benedikt, Grohe, Libkin and Segoufin [5]).
- Adding recursion to constraint query languages. Here a major issue is termination of the recursion, given that universes and relations are now infinite. Two recent papers are by Kreutzer [19] and by Geerts and Kuijpers [13].

Just a few open research directions are:

- Concrete algorithms and implementations. The DEDALE system stands out in the linear constraint context. We need good implemented systems also for other universes.
- Play the constraint database game for other universes than the typical numeric ones, for example term algebras [12, 9].
- In practice we will need "mixed" constraint databases, where the universe is many-sorted, containing many different types of atomic elements, all with their own interpreted functions and predicates, and possibly some functions and predicates mixing the different types. This has not yet been adequately formalized.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] S. Basu. *Algorithms in Semi-Algebraic Geometry.* PhD thesis, New York University, 1996.

[3] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *Journal of Computer and System Sciences*, 59(1):94–115, 1999.

[4] R. Benedetti and J.-J. Risler. *Real Algebraic and Semi-Algebraic Sets.* Hermann, 1990.

[5] M. Benedikt, M. Grohe, L. Libkin, and L. Segoufin. Reachability and connectivity queries in constraint databases. In *Proceedings 19th ACM Symposium on Principles of Database Systems*, 2000.

[6] M. Benedikt and L. Libkin. Relational queries over interpreted structures. *Journal of the ACM*, 47, 2000.

[7] M. Benedikt and L. Libkin. Safe constraint queries. *SIAM Journal on Computing*, 29:1652–1682, 2000.

[8] F. Benhamon and A. Colmerauer, editors. *Constraint Logic Programming: Selected Research.* MIT Press, 1993.

[9] A. Blumensath and E. Grädel. Automatic structures. In *Proceedings 15th IEEE Symposium on Logic in Computer Science*, 2000.

[10] J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry.* Springer-Verlag, 1998.

[11] C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, 3rd edition, 1990.

[12] E. Dantsin and A. Voronkov. Expressive power and data complexity of query languages for trees and lists. In *Proceedings 19th ACM Symposium on Principles of Database Systems*, 2000.

[13] F. Geerts and B. Kuijpers. Linear approximation of planar spatial databases using transitive-closure logic. In *Proceedings 19th ACM Symposium on Principles of Database Systems*, 2000.

[14] M. Grohe and L. Segoufin. On first-order topological queries. In *Proceedings 15th IEEE Symposium on Logic in Computer Science*, 2000.

[15] S. Grumbach and J. Su. Queries with arithmetical constraints. *Theoretical Computer Science*, 173(1):151–181, 1997.

[16] W. Hodges. *Model Theory.* Cambridge University Press, 1993.

[17] O.H. Ibarra and J. Su. On the containment and equivalence of database queries with linear constraints. In *Proceedings 16th ACM Symposium on Principles of Database Systems*, pages 32–43, 1997.

[18] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.

[19] S. Kreutzer. Fixed-point query languages for linear constraint databases. In *19th ACM Symposium on Principles of Database Systems*, 2000.

[20] L. Libkin, G. Kuper, and J. Paredaens, editors. *Constraint Databases.* Springer, 2000.

[21] M. Otto and J. Van den Bussche. First-order queries on databases embedded in an infinite structure. *Information Processing Letters*, 60:37–41, 1996.

[22] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings 13th ACM Symposium on Principles of Database Systems*, pages 279–288. ACM Press, 1994.

[23] D. Thompson R. Laurini. *Fundamentals of Spatial Information Systems.* Number 37 in APIC Series. Academic Press, 1992.

[24] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.

[25] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.

[26] L. van den Dries. *Tame Topology and O-Minimal Structures.* Cambridge University Press, 1998.

[27] Michael F. Worboys. *GIS: A Computing Perspective.* Taylor & Francis, 1995.