A Retrospective on Constraint Databases*

Peter Revesz
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588, USA
revesz@cse.unl.edu

ABSTRACT

In this paper we give a review of constraint databases, a field that was started by Paris Kanellakis, Gabriel Kuper and the author. The review includes basic concepts of data representation, constraint query languages, and query evaluation. We also illustrate applications of constraint databases in the areas of model checking, data mining, trust management, Diophantine polynomial equations, and moving objects.

Categories and Subject Descriptors

H.2 [**Database Management**]: Languages, Logical Design

General Terms

Languages, Management, Security

Keywords

constraint databases, variable elimination, data mining, Diophantine equations, model checking, moving points, trust management

1. INTRODUCTION

In this paper we review some of the basic concepts of *constraint databases* that were introduced in 1990 in [20, 21] by Paris Kanellakis, Gabriel Kuper and the author, who was a student of Paris Kanellakis between 1985 and 1991 and obtained his Ph.D. degree in 1991 at Brown University with a doctoral dissertation on this topic [29].

Since its introduction, constraint databases developed into an interesting and active subfield of database systems. There are several recent books on the subject, including the edited comprehensive volume [24] and the introductory textbook [32].

*This work was supported in part by USA National Science Foundation grant EIA-0091530 and a Gallup Research Professorship.

PCK50, June 8, 2003, San Diego, California, USA

The organization of this paper is the following. Section 2 describes the basic concepts of constraint databases giving several examples. Section 3 describes Datalog queries of constraint databases. Section 4 describes the main techniques for the evaluation of Datalog queries of constraint databases. Section 5 discusses model checking, Section 6 discusses data mining, Section 7 discusses trust management, Section 8 discusses Diophantine polynomial equations, and Section 9 discusses moving objects as sample applications of constraint databases. Section 10 discusses menu-based application development on top of a constraint database system. Finally, Section 11 gives some conclusions and directions for further research in this area.

2. CONSTRAINT DATABASES

Consider a shallow river with some stones in it as shown in Figure 1. We can represent the stones in a relational database as follows:

Stone

X	Y
0	19
6	8
15	12
25	5

The above relational database is equivalent to the following constraint database, where comma means "and":

Stone

X	Y		
x	y	$x = 0,$	$y = 19$
x	y	$x = 6,$	$y = 8$
x	y	$x = 15,$	$y = 12$
x	y	$x = 25,$	$y = 5$

In the above we used only *equality constraints* of the form $u = c$ where u is a variable and c is a constant. We call each row of the table a constraint tuple. The intended meaning of a constraint tuple is that any instantiation of the variables that satisfies the constraint belongs to the relation. For example, if we instantiate x by 6 and y by 8 in the second row, then we get the constraint $6 = 6$ and $8 = 8$, which is obviously true. Hence $(6, 8)$ satisfies the constraint in the second row and belongs to the *Stone* relation as expected.

Any relational database can be translated similarly into a constraint database. However, constraint databases can

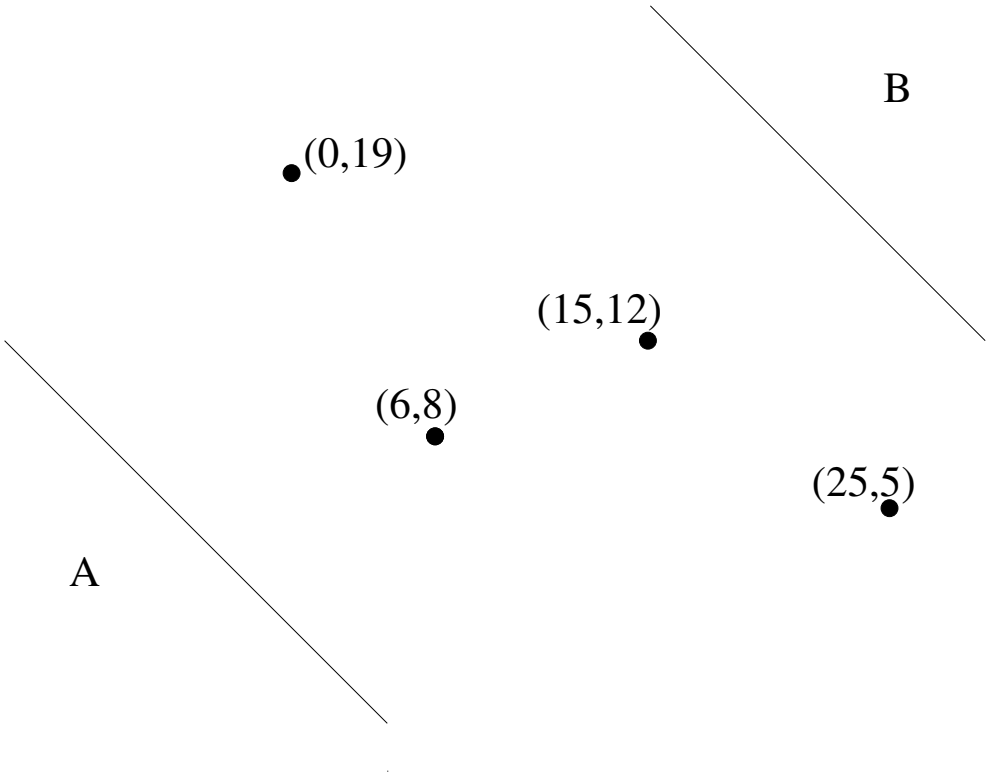

Figure 1: A shallow river with some stones.

represent much more. In particular, we can express the set of points that belong to the two river banks as follows:

Bank

Name	X	Y	
n	x	y	n = "A", $-x - y \geq 0$
n	x	y	n = "B", $+x + y \geq 41$

The above constraint database uses equality constraints over strings, and *addition constraints* of the form $\pm u \pm v \geq b$ where u and v are integer variables and b is an integer constant.

In the case of the *Stone* relation only a finite number of solutions exist. However, in the case of the *Bank* relation, even if we restrict the variables to range over the integer numbers, there is an infinite number of solutions. Hence the *Bank* relation is infinite, although it is finitely represented using constraints.

3. DATALOG WITH CONSTRAINTS

Datalog is a rule-based language that is related to Prolog, which is a popular language for implementing expert systems. Each rule is a statement saying that if some points belong to some relations, then other points must also belong to a defined relation. Each Datalog query contains a Datalog program and an input database.

We divide the set of relation names $\mathcal{R}$ into defined relation names $\mathcal{D}$ and input relation names $\mathcal{I}$. Each Datalog query consists of a finite set of rules of the form:

$$\begin{aligned} R_0(x_1, \ldots, x_k) \quad :\!- \quad & R_1(x_{1,1}, \ldots, x_{1,k_1}), \\ & \vdots \\ & R_n(x_{n,1}, \ldots, x_{n,k_n}). \end{aligned} \tag{1}$$

where each R_i is either an input relation name or a defined relation name, and the xs are either variables or constants.

The relation names $R_0, \ldots, R_n$ are not necessarily distinct. They could also be constraint relation names, such as the equality constraint relation $=$ and the addition constraint relation defined above. We write the constraint relations using the usual notation. For example, we will write $u = v$ instead of $Equal(u, v)$ and write $u + v \geq b$ instead of $Addition(u, v, b)$. When we have $u \geq v$ and $v \geq u$, for any u and v, then we also abbreviate it as $u = v$.

The preceding rule is read "R_0 is true if R_1 and ... and R_n are all true." R_0 is called the *head* and $R_1, \ldots, R_n$ is called the *body* of the rule.

We can represent as a set of Datalog rules any constraint database. For example, the input relation *Stone* can be represented by:

$$\begin{aligned} Stone(x, y) \quad :\!- \quad & x = 0, \quad y = 19. \\ Stone(x, y) \quad :\!- \quad & x = 6, \quad y = 8. \\ Stone(x, y) \quad :\!- \quad & x = 15, \quad y = 12. \\ Stone(x, y) \quad :\!- \quad & x = 25, \quad y = 5. \end{aligned}$$

Similarly, the two banks of the river can be represented by:

$$Bank(n, x, y) \quad :\!- \quad n = \text{"}A\text{"}, \quad -x - y \geq 0.$$

$$Bank(n, x, y) \quad :\!- \quad n = \text{"}B\text{"}, \quad +x + y \geq 41.$$

3.1 The Stepping Stone Problem

Imagine that we arrive at a big but shallow river and we would like to get to the other shore. The river has in it several large stones and we would like to cross the river without getting wet. The question is whether there is a way to do that assuming that the maximum size step we can take is a fixed length, say 10 units.

An instance of the stepping stone problem is shown in Figure 1. In that instance, we can cross the river starting from point $(0, 0)$ of bank A, then stepping on the stone at location $(6, 8)$, then stepping on the stone at location $(15, 12)$, and from there stepping on point $(22, 19)$ of bank B. It can be calculated that the distance between any pair of adjacent points on this path is at most 10 units.

In this problem we clearly need the Euclidean distance function, which depends on multiplication and difference. Since we do not have these operators, we will define them using Datalog with addition constraints. First we define the *difference* relation D as:

$$\begin{aligned} D(x, y, z) \quad &:\!- \quad x = y, \\ &\qquad z = 0. \end{aligned}$$

$$\begin{aligned} D(x, y, z) \quad &:\!- \quad D(x', y, z'), \\ &\qquad x = x' + 1, \\ &\qquad z = z' + 1. \end{aligned}$$

Then we define the *multiplication* relation M as:

$$M(x, y, z) \quad :\!- \quad x = 0, y = 0, z = 0.$$

$$\begin{aligned} M(x, y, z) \quad &:\!- \quad M(x', y, z'),\ D(z, z', y), \\ &\qquad x = x' + 1. \end{aligned}$$

$$\begin{aligned} M(x, y, z) \quad &:\!- \quad M(x, y', z'),\ D(z, z', x), \\ &\qquad y = y' + 1. \end{aligned}$$

We also define the *absolute difference* relation AD as follows:

$$AD(x, y, z) \quad :\!- \quad D(x, y, z),\ x \geq y.$$

$$AD(x, y, z) \quad :\!- \quad D(y, x, z),\ y \geq x.$$

Suppose that our maximum step size from stone to stone is 10 units. Then we can define using the above three relations, the *Close* relation, which contains pairs of points within 10 units of each other, as follows:

$$\begin{aligned} Close(x', y', x, y) \quad &:\!- \quad AD(x, x', dx), \\ &\qquad AD(y, y', dy), \\ &\qquad M(dx, dx, dx2), \\ &\qquad M(dy, dy, dy2), \\ &\qquad dx2 + dy2 \leq 100. \end{aligned}$$

The last line is equivalent to $-dx2 - dy2 \geq -100$, which is an addition constraint. We can find whether we can get from a point on shore A to a point on shore B via a sequence of steps over the stones in the river as follows:

$$Step(x, y) \quad :\!- \quad Bank(n, x, y),\ n = \text{“}A''.$$

$$\begin{aligned} Step(x, y) \quad &:\!- \quad Step(x', y'),\ Stone(x, y), \\ &\qquad Close(x', y', x, y). \end{aligned}$$

$$\begin{aligned} Reach(x, y) \quad &:\!- \quad Bank(n, x, y),\ n = \text{“}B'', \\ &\qquad Step(x', y'),\ Close(x', y', x, y). \end{aligned}$$

4. EVALUATION TECHNIQUES

4.1 Proof Tree

Consider a Datalog rule of the form (1). We can define a Datalog *rule instantiation* as a substitution of each variable by constants in a rule. We say that an instantiated relation R_i in the rule body is true if an only if the instantiation is a solution of a constraint tuple in R_i. We also define *rule application* as the addition of the instantiated tuple in the rule head to the relation R_0, if all the instantiated tuples in the rule body are true. For example,

$$Step(0, 0) \quad :\!- \quad Bank(\text{“}A'', 0, 0),\ \text{“}A'' = \text{“}A''.$$

is an instantiation of the first rule for the *Step* relation. We instantiated x and y by 0 and n by the string “A”. Since the tuple $(\text{“}A'', 0, 0)$ is a solution of the first constraint tuple of the *Bank* relation, and $\text{“}A'' = \text{“}A''$ is obviously true, we can add to the *Step* relation the tuple $(0, 0)$. As another example,

$$\begin{aligned} Step(6, 8) \quad &:\!- \quad Step(0, 0),\ Stone(6, 8), \\ &\qquad Close(0, 0, 6, 8). \end{aligned}$$

is an instantiation of the second rule for *Step*. In this case, $Step(0, 0)$ is true because we just added above the tuple $(0, 0)$ to the *Step* relation. $Stone(6, 8)$ is true because $(6, 8)$ satisfies the second constraint tuple of the *Stone* relation. Similarly, it can be shown that $(0, 0, 6, 8)$ satisfies the constraint relation *Close*.

By a sequence of rule applications, we can prove any particular constant tuple that is in an output relation to be in it. It is easy to draw any proof as a tree in which each internal node is the head of an instantiated rule, and its children are the instantiated relations in the body of the rule. For example, the picture in Figure 2 and the proof tree in Figure 3 show how we can get from one side of the river to the other side. We call *proof-based semantics* of a Datalog query Q and input database I the set of constant tuples that have a proof, denoted $\vdash_{Q,I}$.

4.2 Bottom-Up Constraint Evaluation

The constraint proof-based semantics of Datalog queries defines the set of constraint tuples that can be derived in the following way.

We call a *constraint instantiation* of a rule the substitution of each relation R_i in the body by a constraint tuple t_i that is in R_i. The substitution renames the variables in t_i to those variables with which R_i occurs in the rule.

Let Q be a query, I an input constraint database, and $R_0(x_1, \ldots, x_k)$ a constraint relation scheme and let t be a constraint over the variables $x_1, \ldots, x_k$. We say that $t \in R_0$ has a constraint proof using Q and I, written as $\vdash^c_{Q,I} t \in R_0$, if and only if:
$R_0 \in \mathcal{I}$ and $t \in R_0$ or
$R_0 \in \mathcal{D}$ and for some rule of form (1) in Q there is a constraint instantiation

$$R(x_1, \ldots, x_k) :\!- t_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, t_n(x_{n,1}, \ldots, x_{n,k_n}).$$

where $\vdash^c_{Q,I} t_i \in R_i$ for each $1 \leq i \leq n$ and

$$t = \exists * t_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, t_n(x_{n,1}, \ldots, x_{n,k_n}).$$

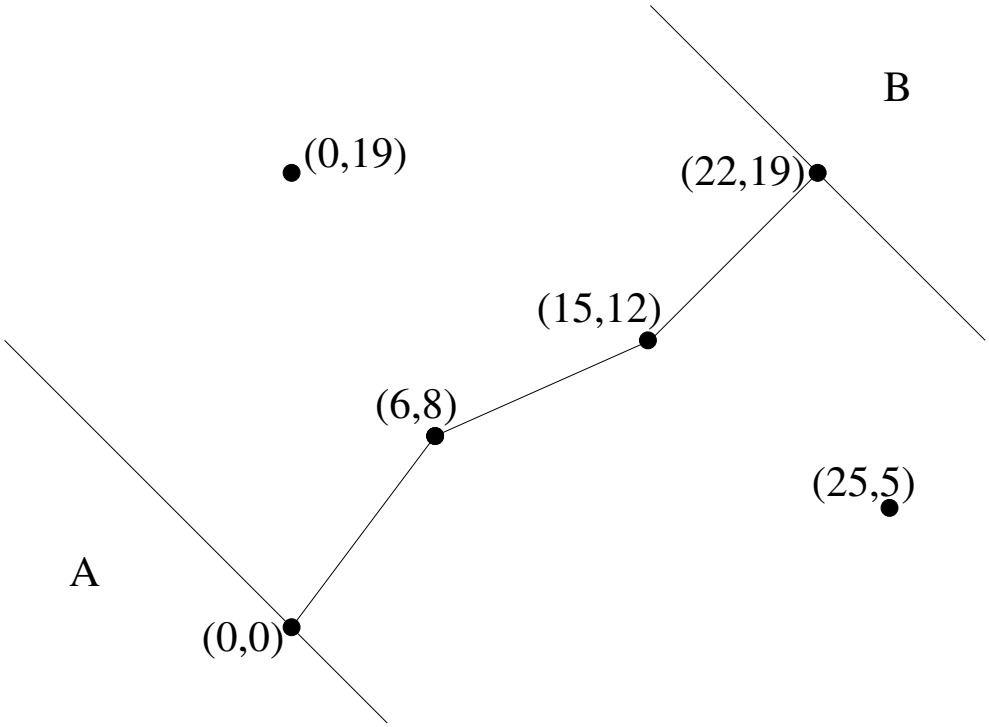

Figure 2: A solution for the stepping stone problem.

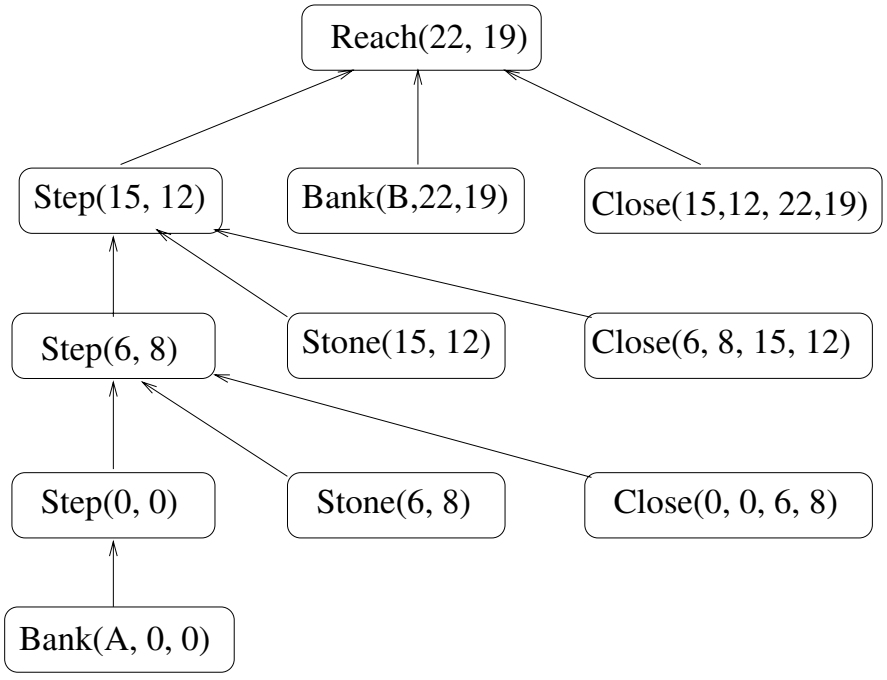

Figure 3: A proof tree for the Step relation.

where $*$ is the list of the variables in the body of the rule that do not occur in the head of the rule.

We call *constraint rule application* the constraint instantiation of a rule, elimination of the unneeded variables, and the addition of t to R_0, if t is not already *subsumed* by the other constraint tuples already in R_0. A constraint tuple is *subsumed* by other constraint tuples, if its solution space is included in the solution space of the other tuples.

The *constraint proof-based* semantics of Datalog queries is the following. Each Datalog query Q is a function that on any input constraint database I returns for each relation name R the relation $\{t \ : \vdash^c_{Q,I} t \in R\}$.

The following theorem shows that the proof-based semantics and the constraint proof-based semantics are equivalent.

THEOREM 4.1. *The following equality is true:*

$$\{(a_1, \ldots, a_k) \ : \vdash_{Q,I} R(a_1, \ldots, a_k)\} = \{Mod(t) \ : \vdash^c_{Q,I} t \in R\}$$

The *bottom-up constraint evaluation* of Datalog queries starts from an input constraint database and query and repeatedly applies some rule of the query until no new constraint tuples can be derived and added to the constraint database. We consider a constraint tuple of a relation new if there is some instantiation that makes it true but does not make any other constraint tuple of that relation true.

It is also possible to extend the least fixed point semantics similarly. Then it can be shown that the constraint least fixed point and the constraint proof-based semantics are equivalent. Hence in the following, we keep using $lfp(Q(D))$ to describe the semantics of a Datalog query Q on input constraint database D.

Sometimes we cannot evaluate $lfp(Q(D))$ precisely, but we can approximate it. The main idea behind the approximation is that in an addition constraint the value of the bound may be so small that we do not care too much about it. This leads to the idea of placing a limit l on the allowed smallest bound. To avoid smaller bounds than l, we may do two different modifications to a constraint tuple:

Modification 1: We change in the constraint tuple the value of any bound b to be $\max(b, l)$.

Modification 2: We delete from each constraint tuple any constraint with a bound that is less than l.

We can apply either of these modifications to each tuple in an input constraint database. During the constraint bottom-up evaluation we can also apply either of the modifications to the result of each rule application. In this way, we obtain modified bottom-up evaluations.

Given a query Q, input database D, and a fixed constant l, let $Q(D)_l$ and $Q(D)^l$ denote the output of the first and second modified evaluation algorithms, respectively. We can show the following.

THEOREM 4.2. *For any Datalog with integer addition constraint query Q, input database D, and constant $l < 0$, the following is true:*

$$Q(D)_l \subseteq lfp(Q(D)) \subseteq Q(D)^l$$

Further, $Q(D)_l$ and $Q(D)^l$ can be evaluated in finite time.

We can also get better and better approximations using smaller and smaller values as bounds. In particular, we have the following theorem.

THEOREM 4.3. *For any Datalog with addition constraints query Q, input database D, and constants l_1 and l_2 such that $l_1 \leq l_2 < 0$, the following hold:*

$$Q(D)_{l_2} \subseteq Q(D)_{l_1} \quad \text{and} \quad Q(D)^{l_1} \subseteq Q(D)^{l_2}$$

4.3 Evaluation of the stepping stones query

4.3.1 Difference and Absolute Difference

Let us rewrite the rules for the difference relation into the following logically equivalent form:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq 0, \quad y - x \geq 0, \\ & z \geq 0, \quad -z \geq 0. \end{aligned}$$

$$\begin{aligned} D(x,y,z) \ :\!- \quad & D(x',y,z'), \\ & x - x' \geq 1, \quad x' - x \geq -1, \\ & z - z' \geq 1, \quad z' - z \geq -1. \end{aligned}$$

Taking the first rule and renaming x by x' and z by z', we get the following:

$$\begin{aligned} D(x',y,z') \ :\!- \quad & x' - y \geq 0, \quad y - x' \geq 0, \\ & z' \geq 0, \quad -z' \geq 0. \end{aligned}$$

Now substitute the right hand side of the above into the second rule for D and get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x' - y \geq 0, \quad y - x' \geq \ \ 0, \\ & z' \geq 0, \quad -z' \geq \ \ 0, \\ & x - x' \geq 1, \quad x' - x \geq -1, \\ & z - z' \geq 1, \quad z' - z \geq -1. \end{aligned}$$

We can eliminate the variables x' and z' by adding the first and third rows and also adding the second and fourth rows. We get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq 1, \quad y - x \geq -1, \\ & z \geq 1, \quad -z \geq -1. \end{aligned}$$

This is the same as the first constraint tuple for D except that the right hand side has the bounds 1 and -1 instead of 0. We can continue similarly the constraint rule applications. For the ith application, we get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq i, \quad y - x \geq -i, \\ & z \geq i, \quad -z \geq -i. \end{aligned} \tag{2}$$

The rule applications can continue similarly l times. Hence we get one copy of Equation (2) for each $0 \leq i \leq l$. In the $(l+1)$st rule application we get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq -l+1, \quad y - x \geq l-1, \\ & z \geq -l+1, \quad -z \geq l-1. \end{aligned}$$

However, by Modification 1 we would get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq -l+1, \quad y - x \geq l, \\ & z \geq -l+1, \quad -z \geq l. \end{aligned}$$

because the bound $l-1$ is replaced with l as $\max(l-1, l) = l$. This conjunction of constraints is unsatisfiable. Hence we do not add anything to the D relation.

By Modification 2 we would get:

$$\begin{aligned} D(x,y,z) \ :\!- \quad & x - y \geq -l+1, \\ & z \geq -l+1. \end{aligned} \tag{3}$$

because the bound $l-1$ is less than l and is deleted. The conjunction of the constraints on the right hand side is satisfiable. But can we use this new constraint tuple in another rule application? If we substitute in it x by x' and z by z', then we get the following:

$$\begin{array}{rcl} D(x',y,z') & :\!- & x'-y \geq -l+1, \\ & & z' \geq -l+1. \end{array}$$

Now substitute the right hand side of the above into the second rule for D and get:

$$\begin{array}{rcllll} D(x,y,z) & :\!- & x'-y \geq & -l+1, & & \\ & & z' \geq & -l+1, & & \\ & & x-x' \geq & 1, & x'-x \geq -1, & \\ & & z-z' \geq & 1, & z'-z \geq -1. & \end{array}$$

Eliminating variables x' and z', we get:

$$\begin{array}{rcl} D(x,y,z) & :\!- & x-y \geq -l+2, \\ & & z \geq -l+2. \end{array}$$

Since we got a set of constraints that is subsumed by the previously derived constraint tuple, we do not add this to the relation D. For any l, the lower bound of $D(x,y,z)$ will be the union of Equation (2) for $0 \leq i \leq -l$ and the upper bound will be the union of the lower bound and Equation (3). Given, D, it is easy to find the closed form of AD.

4.3.2 Multiplication

We claim that the lower bound of $M(x,y,z)$ is:

$$\begin{array}{rcl} M(x,y,z) & :\!- & x \geq j,\ -x \geq -j, \\ & & y \geq i,\ -y \geq -i, \\ & & z \geq ij,\ -z \geq -ij. \end{array} \quad (4)$$

for each $0 \leq i,j \leq -l$ such that $l \leq -ij$.

We prove by induction on i and j that Equation (4) holds. The first rule of multiplication clearly implies the case for $i=j=0$. Let us assume for any $0 \leq i \leq -l$ and $0 \leq j < -l$ and prove for i, and $j+1$. We use the second rule for multiplication. After the necessary substitutions we get:

$$\begin{array}{rcllll} M(x,y,z) & :\!- & x' \geq & j, & -x' \geq & -j, \\ & & y \geq & i, & -y \geq & -i, \\ & & z' \geq & ij, & -z' \geq & -ij, \\ & & z-z' \geq & i, & z'-z \geq & -i, \\ & & x-x' \geq & 1, & x'-x \geq & -1. \end{array}$$

The first three lines of constraints come from Equation (4) with the renaming of x by x' and z by z'. The fourth and the second line come from Equation (2) with the renaming of x by z, y by z', and z by y. The last line is a rewrite of $x = x'+1$ in terms of addition constraints. After eliminating x' and z' we get:

$$\begin{array}{rcllll} M(x,y,z) & :\!- & x \geq j+1, & & -x \geq -(j+1), & \\ & & y \geq i, & & -y \geq -i, & \\ & & z \geq i(j+1), & & -z \geq -i(j+1). & \end{array} \quad (5)$$

If $l \leq -i(j+1)$, then the above is the same as Equation (4) with $j+1$ instead of j. Otherwise, if $l > -i(j+1)$, then Modification 1 changes the above to the unsatisfiable:

$$\begin{array}{rcllll} M(x,y,z) & :\!- & x \geq j+1, & & -x \geq -(j+1), & \\ & & y \geq i, & & -y \geq -i, & \\ & & z \geq i(j+1), & & -z \geq l. & \end{array}$$

hence nothing more is derived. The third rule can be used similarly to prove the condition for $i+1$ and j for any $0 \leq i < -l$ and $0 \leq j \leq -l$.

In each rule application either i or j is increased by one. If either of these increase to greater than $-l$, then again we get an unsatisfiable condition by Modification 1. Hence the lower bound in Equation (4) is exact.

Now let us find the upper bound of M. Clearly the upper bound contains all the tuples of the lower bound. However, if $l > -i(j+1)$ in Equation (5), and we use Modification 1 instead of Modification 2, then we get:

$$\begin{array}{rcllll} M(x,y,z) & :\!- & x \geq j+1, & & -x \geq -(j+1), & \\ & & y \geq i, & & -y \geq -i, & \\ & & z \geq i(j+1). & & & \end{array}$$

Since this is satisfiable, we have to see what other constraint tuples we can derive using it. It is easy to see that after applying the second rule and eliminating x and x' again, we get if $-(j+2) \leq l$:

$$\begin{array}{rcllll} M(x,y,z) & :\!- & x \geq j+2, & & -x \geq -(j+2), & \\ & & y \geq i, & & -y \geq -i, & \\ & & z \geq i(j+2). & & & \end{array}$$

We can continue applying the second rule similarly until the upper bound of x will be $-l+1$. Then by Modification 2 we get for any $0 \leq i \leq l$:

$$\begin{array}{rcl} M(x,y,z) & :\!- & x \geq -l+1, \\ & & y \geq i,\ -y \geq -i, \\ & & z \geq i(-l+1). \end{array} \quad (6)$$

By symmetry, by applying repeatedly the third rule until the lower bound of y will be $-l+1$ and using Modification 2 we get for any $0 \leq j \leq l$:

$$\begin{array}{rcl} M(x,y,z) & :\!- & x \geq j,\ -x \geq -j, \\ & & y \geq -l+1, \\ & & z \geq (-l+1)j. \end{array} \quad (7)$$

If the lower bound of both x and y reach $l+1$, then we get:

$$\begin{array}{rcl} M(x,y,z) & :\!- & x \geq -l+1, \\ & & y \geq -l+1, \\ & & z \geq (-l+1)^2. \end{array} \quad (8)$$

The upper bound of M will be the union of the lower bound in Equation (4) and the constraint tuples in Equations (6-8).

As l decreases the lower and upper approximations for both D and M get ever closer to our usual definitions of the difference and multiplication relations.

4.3.3 The Other Relations

If we choose $l < -100$, both the upper and the lower bound of *Close* will be:

$$\begin{array}{rcl} Close(x',y',x,y) & :\!- & x-x'=0,\ y-y'=0. \\ Close(x',y',x,y) & :\!- & x-x'=0,\ y-y'=1. \\ Close(x',y',x,y) & :\!- & x-x'=0,\ y-y'=2. \\ Close(x',y',x,y) & :\!- & x-x'=0,\ y-y'=3. \\ \vdots & \vdots & \vdots \\ Close(x',y',x,y) & :\!- & x-x'=6,\ y-y'=8. \\ Close(x',y',x,y) & :\!- & x-x'=8,\ y-y'=6. \end{array} \quad (9)$$

Now let us use the input relations *Stone* and *Bank* to find *Step* and *Reach*. Applying the first rule will yield:

$$Step(x, y) \quad :\!- \quad -x - y \geq 0.$$

This simply means that we can be at any point of bank A of the river. Now any application of the second rule for *Step* is going to derive only (x, y) values that are already in the *Stone* relation. For example, let's consider the following application of the second rule using the above constraint tuple for *Step*, the second constraint tuple for *Stone*, and the second to the last constraint tuple for *Close* in (9) above. We again expand each equality into a conjunction of inequalities.

$$\begin{array}{lllll} Step(x, y) & :\!- & -x' - y' \geq 0, & & \\ & & x \geq 6, & -x \geq -6, & \\ & & y \geq 8, & -y \geq -8, & \\ & & x - x' \geq 6 & x' - x \geq -6, & \\ & & y - y' \geq 6 & y' - y \geq -6. & \end{array}$$

Eliminating x' and y' we get:

$$\begin{array}{lllll} Step(x, y) & :\!- & x \geq 6, & -x & \geq & -6, \\ & & y \geq 8, & -y & \geq & -8, \\ & & & -x - y & \geq & -12. \end{array}$$

The last constraint is implied by the first two rows. Hence simplifying we get:

$$\begin{array}{lllll} Step(x, y) & :\!- & x \geq 6, & -x \geq -6, \\ & & y \geq 8, & -y \geq -8. \end{array}$$

which is equivalent to the second constraint tuple for *Stone*. This means that we can step on the second stone. Similarly, we can prove that we can also step on the stone at $(15, 12)$ and on location $(22, 19)$ of bank B of the river.

Although we used an approximation for the difference, multiplication, and absolute difference relations, the output of the other relations were exactly calculated with matching lower and upper bounds. This is because for any particular instance of this problem where the absolute value of any stone coordinate is at most some number n, we only need to find the difference and product of numbers that are at most $2n$. Hence we only need to use a finite subset of the difference and products relations on the integer numbers. The above approximation with $l = -2n$ returns the needed parts of the D and M relations.

5. MODEL CHECKING

Constraint databases also influenced automata theory. In this section we first describe timed, constraint, and refinement automata. Then we consider the problem of symbolic model checking of such automata [28, 7, 1] and explain how it may be solved using constraint databases.

Timed automata, which was introduced by Alur and Dill [2], are automata where each state has a finite number of state variables, whose values change on each transition from a state to another state. Each transition is composed of a set of guard constraints that need to be satisfied and a set of assignment statements that change the values of the state variables.

Timed automata can be generalized into *constraint automata*, which allow guard constraints that are input relations. Of course, for any input database instance, in the guard constraints input database relations can be replaced by disjunctions of conjunctions of constraints that represent the relation. The connection between Datalog queries and constraint automata is very close, because any constraint automata can be expressed as a Datalog query. Translations of constraint automata into constraint logic programming [18] or constraint Datalog queries are presented by Fribourg and Richardson [9], Delzanno and Podelski [8], and Revesz [30, 31].

Example 5.1. Consider a tree described by the input relation $Parent(x, y)$ which is true if x is a child of y in the tree. Let us find those pairs of nodes that are *independent*, that is, neither is an ancestor of the other. It can be done using the following Datalog query.

$$\begin{array}{lll} Indep(x', y') & :\!- & Parent(x', z),\ Parent(y', z), \\ & & x' \neq y'. \end{array}$$

$$\begin{array}{lll} Indep(x', y') & :\!- & Indep(x, y),\ Parent(x', x), \\ & & y' = y. \end{array}$$

$$\begin{array}{lll} Indep(x', y') & :\!- & Indep(x, y),\ Parent(y', y), \\ & & x' = x. \end{array}$$

In the above, the first rule says that if x' and y' are different children of the same parent z, then they are independent. The second rule says that if x and y are independent and x is the parent of x', then x' and $y' = y$ are also independent. Similarly, the third rule says that if x and y are independent, and y is the parent of y', then $x' = x$ and y' are also independent.

The left side of Figure 4 shows the constraint automaton that is equivalent to the above Datalog query. The stare variables are x and y, the initial state is *Initial*, and the final state is *Indep*. The convention for constraint automata is that the new values of the state variables are the primed variables. The state transitions may use other variables, but all except the primed variables are eliminated. This is is why we need the $y' = y$ and the $x' = x$ constraints in the above Datalog query.

Another type of automata is called *refinement automata*. On each transition a refinement automaton adds some constraints to a central constraint store.

Example 5.2. Instruction scheduling for single-issue processors with arbitrary latencies [36] and many other problems can be modeled as the problem of assigning a unique value for each node of a labeled tree, where the labels are positive integers, and the difference between the values of each parent and its child must be greater than or equal to the label between them.

Let the input relation $Edge(x, y, l)$ be true if x is a child of y in the tree and l is the label on the edge between them. At first we create a central constraint store which contains a variable $x.v$ for each node x of the tree. Then the following *refinement rule* adds to the constraint store the needed constraints between the values of children and parents.

$$y.v - x.v \geq l \quad :\!- \quad Edge(x, y, l).$$

The above ensures that each node and its descendants will get a distinct value. We also need to assure that independent nodes are also given different values. Clearly, the *Parent* relation of Example 5.1 can be found by projecting the first two attributes of the *Edge* relation. Then we can find the *Indep* relation as in Example 5.1. The following refinement

rule takes care of independent nodes:

$$x.v \neq y.v \quad :\!- \quad Indep(x, y).$$

Clearly, any assignment that satisfies the central constraint store gives a unique value for each node. The right side of Figure 4 shows the refinement automaton that is equivalent to the above refinement rules.

In the above two examples, the constraint and the refinement automata were separate entities. It is possible to combine them as is shown by the following example.

EXAMPLE 5.3. Consider again the stepping stone query. We can assume that *Close* is an input constraint relation, because it can be represented as shown in Equation (9) for any value of $l < -100$. Since the convention for constraint automata is that the new values of the state variables are the primed variables, to avoid any confusion, we rewrite the stepping stone query as follows:

$$\begin{aligned} Step(x', y') \quad &:\!- \quad Bank(\text{“}A''\text{”}, x', y'). \\ Step(x', y') \quad &:\!- \quad Step(x, y),\ Stone(x', y'), \\ &\qquad Close(x', y', x, y). \\ Reach(x', y') \quad &:\!- \quad Step(x, y),\ Bank(\text{“}B''\text{”}, x', y'), \\ &\qquad Close(x', y', x, y). \end{aligned}$$

This query can be represented by the constraint automata shown in Figure 5. The initial state is *Initial* and the final state is *Reach*.

Now suppose that there are fish in the river everywhere initially. We can represent this by:

$$Fish(x, y) \quad :\!- \quad x + y > 0,\ -x - y > 41.$$

Let us assume that if one steps on a stone, then the fish swim either upstream or downstream at least five units away from it. That is, if we step on any stone (a, b), then each fish will move above the line $y - x = b - a + 5$ or below the line $y - x = b - a - 5$. Further, let us assume that the fish are too frightened to swim back even when we step on the next stone. How does the fish area change? It may only shrink as we step on more stones. Let the central constraint store represent the fish area, i.e., be the *Fish* relation. We only add constraints to the central store. Therefore, we modify the above automaton as shown in Figure 6. The same can be expressed by modifying the head of the second rule of the stepping stone query. The new head of that rule will be:

$$Step(x', y'),\ ((y - x > y' - x' + 5) \vee (y - x < y' - x' - 5))$$

Whenever we add a constraint tuple to the step relation, we also take the conjunction of that constraint tuple and the constraint $(y-x > y'-x'+5) \vee (y-x < y'-x'-5)$, eliminate the variables x' and y' from it, and then obtain a constraint over x and y, which we conjoin to the *Fish* relation.

In Example 5.3 the constraint and the refinement automata cannot be separated. That is, we cannot at first find all the stones that we may step on and then eliminate all the areas five units above or below them. By doing concurrently the additions of constraints (conjoinings) to the *Fish* relation, which is the central constraint store, and the additions of constraint tuples (disjoinings) to the *Step* relation, we can be assured that at any given time the *Fish* relation reflects correctly the situation that would result if we stepped on the stones that are at that time in the *Step* relation.

We call the combined automaton a *constraint-refinement automaton*. Many questions may be asked about constraint-refinement automata. For example, the final fish area depends on what path is taken to cross the river, i.e., how we got from the initial state to the final state. We may wish to find a path that disturbs the fish the least.

Given a particular constraint or refinement automaton, *model checking* is the task of checking what properties may hold in certain states in the automaton [28, 7, 1]. Model checking is easy to do, if we can get a closed-form evaluation for the constraint queries that are equivalent to the constraint or refinement automaton. Therefore, the closed-form evaluation techniques for constraint queries are useful also for model checking.

If S is any state in a constraint automaton, then the set of possible values of the state variables in it are equivalent to the set of constant tuples in the Datalog output relation S' that corresponds to S. If S is a state in a constraint-refinement automaton, then we need all pairs of $(S', Store)$ where *Store* is the value of the central constraint store at the time when S' is the value of the Datalog output relation during any time in the operation of the automaton. If the Datalog output relations without any additions to the constraint store can be found, then all possible pairs also can be found.

For the fish query, the *Step* relation is finite, because there are only a finite number of stones and *Step* is a subset of *Stone*. The *Fish* relation is also finite, because for each *Step* relation we can add only one disjunctive constraint to it. If there are N tuples in *Stone*, then there are only 2^N possible values of *Step*, and the same number of possible values in *Fish*.

6. DATA MINING

Constraint databases are also applied in data mining. An essential problem in data mining is the representation of categories of objects. This representation is often done using decision trees. Recently, Johnson et al. [19], Geist [10], and Geist and Sattler [11] suggested that constraint databases be used instead of decision trees to represent categories of objects. The advantage is that the categories can be queried similarly to the original data elements. In this section we explain this idea and also discuss some additional ideas of applying Boolean constraint databases to data mining.

Suppose that we are interested in some high-level features or categories $A, B, C, \ldots$ based on the values of some lower-level real-valued attributes $x_1, \ldots, x_n$. That is, $A, B, C, \ldots$ are in $dom(x_1) \times \ldots \times dom(x_n)$.

Suppose also that a data mining company accumulates a huge database of such categorizations. A major task of the company is to provide for customers these categorizations or new categorizations that could be derived from the already known ones.

For example, suppose that the data mining company has available the results of two studies. The first study identified "artistic" people $S_{artistic}$ based on the attributes $a_1, \ldots, a_n$. The second study identified "reliable" people $S_{reliable}$ based on the attributes $b_1, \ldots, b_m$, where the as and the bs are not necessarily distinct.

Now suppose that an art foundation is looking for a new

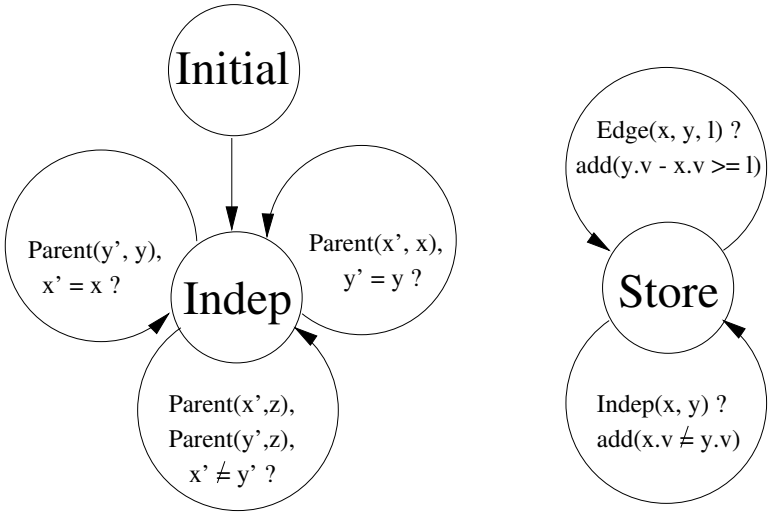

Figure 4: A constraint (left) and a refinement automaton (right) for Tree assignment.

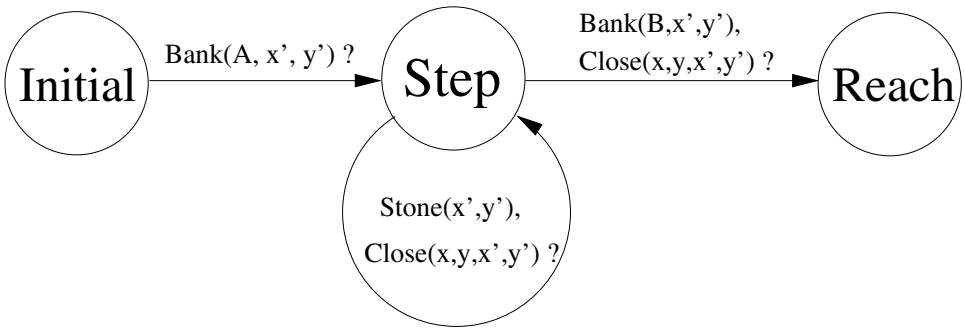

Figure 5: A constraint automaton for the Step query.

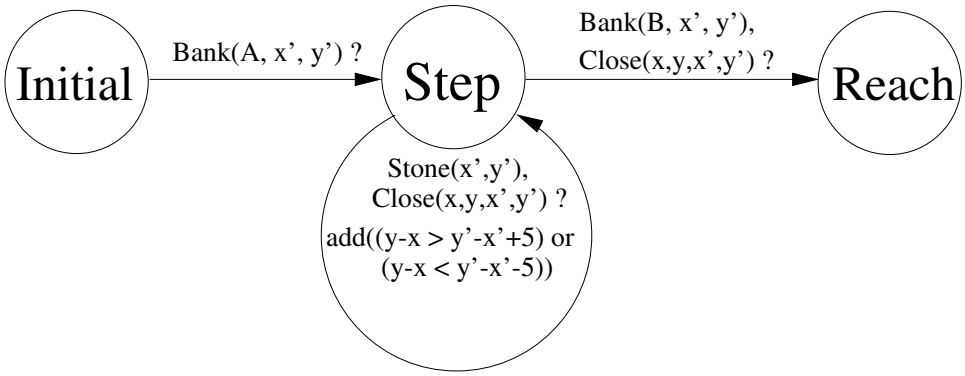

Figure 6: A constraint-refinement automaton for the Fish area.

director who must be both "artistic" and "reliable". The art foundation is interested to know what type of people would qualify for the job. Obviously, $S_{artistic} \bowtie S_{reliable}$ is the best guess, and that is what the data mining company could sell to the art foundation.

The above process is closely tied with constraint databases. There are several classification methods, such as perceptrons, that yield convex subregion descriptions of categories. Hence if one of these classification methods is used, then we can assume that each category is describable as a convex subregion of $\mathbf{R}^n$ where $\mathbf{R}$ is the set of real numbers. This convex subregion is also describable using a linear constraint database relation.

The join of the two linear constraint relations is a typical constraint database query. Using Datalog it can be expressed by:

$$\begin{aligned} S_{director}(c_1, \ldots, c_k) \quad :\!- \quad & S_{artictic}(a_1, \ldots, a_n), \\ & S_{reliable}(b_1, \ldots, b_m). \end{aligned}$$

where $c_1, \ldots, c_k$ is an alphabetical order of the distinct attributes names in $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$.

The art foundation presumably can test the potential candidates as to their aptitude for being a director by testing whether what is known about the candidates falls within the $S_{director}$ relation. This can be done by creating a constraint relation $Candidate(id, c_1, \ldots, c_k)$. Each constraint tuple describes one art director candidate using a unique id, and the values of any of the known attributes. For example,

$$Candidate(id, c_1, \ldots, c_k) \quad :\!- \quad id = 23, c_1 = 25, c_7 = 15.$$

describes the 23rd candidate using the known values for the first and the seventh attributes. The other attributes are not assigned any values, meaning that they could be anything. Then it is easy to find the fit candidates using the following constraint database query:

$$\begin{aligned} Fit(id) \quad :\!- \quad & S_{director}(c_1, \ldots, c_k), \\ & Candidate(id, c_1, \ldots, c_k). \end{aligned}$$

Constraint database techniques can be also helpful in other cases of data mining. For example, suppose that the relations $S_{artistic}$ and $S_{reliable}$ are huge relations, and before taking their join, we would like to be assured that it is not empty, that is:

$$S_{artistic} \cap S_{reliable} \neq \emptyset \tag{10}$$

Fortunately, we have some meta-information available about the categories. This meta-information forms only a small database that can be mined separately and efficiently. In particular, suppose that we already know the following by previous data mining:

$$\begin{aligned} S_{musical} &\subseteq S_{artistic} \\ S_{responsible} &\subseteq S_{reliable} \\ S_{musical} \cap S_{reliable} &\neq \emptyset \end{aligned} \tag{11}$$

The conjunction of the above three facts forms a constraint formula with four Boolean algebra variables. (The elements of this Boolean algebra are those subsets of $\mathbf{R}^n$ which are expressible using linear constraints.) Eliminating the variables $S_{musical}$ and $S_{responsible}$, yields exactly the constraint in Equation (10). Therefore, the join of the two constraint relations $S_{artistic}$ and $S_{reliable}$ is nonempty. Therefore it can be returned to the art foundation.

In the previous, we assumed that the data mining company had available $S_{artistic}$ and $S_{reliable}$ in its database. However, suppose that in fact it *does not have* any constraint database description of these categories. However, it still has the meta-information described in Equation (11). Then it can still derive Equation (10). This could still be a useful information for the art foundation, because it can strengthen the hope of finding a qualified director. Furthermore, the join of $S_{musical}$ and $S_{responsible}$ is a constraint relation that can be calculated and returned to the art foundation. This would be a lower bound of $S_{director}$. Nevertheless, this lower bound can be useful, because if candidates can be identified who fall within this lower-bound, then the art foundation can safely chose one of them.

Whenever we have a knowledgebase D like the one in Equation (11), we may get a new assertion C that we have to test against the knowledgebase. If we are interested in testing whether C is possibly true, we need to decide whether $D \wedge C$ is satisfiable. If we are interested in whether C is necessarily true, we need to decide whether $D \wedge \overline{C}$ is false. Both of these can be decided if the Boolean algebra is atomless. For more about Boolean algebras see Burris and Sankappanavar [3], Halmos [16], Helm et al. [17], Marriott and Odersky [26], and Revesz [32].

7. TRUST MANAGEMENT

Trust management is a new approach for authorization and access control in distributed systems. It is based on signed distributed policy statements expressed in a policy language. Li and Mitchell [25] has recently proposed Datalog with constraints as a foundation for trust management languages.

Their work fits very well with the Datalog with constraints work. The only new thing is that they allow variables to range over hierarchical domains. The elements of a hierarchical domain are paths that start from the root of a given tree. For example, machine domain names form a hierarchical domain. For any hierarchical domain H and $g, h \in H$, the constraint $h \prec g$ means that the path name h is an expansion of the path name g. For example, $nsf.gov \prec gov$. An alternative notation for machine names is to reverse the order and to put them in brackets. For example, $nsf.gov$ is written as $\langle gov, nsf \rangle$. The following example is modified from [25].

EXAMPLE 7.1. An entity A grants to an entity B the permission to connect to machines in the domain "stanford.edu" at port number 80, and allows B to further delegate any part of the permission from time 0 to 8. In addition, assume that B grants to another entity D the permission to connect to the host "cs.stanford.edu" and any machine in the domain "cs.stanford.edu" at any port number, with validity period from time 5 to 12.

Let us introduce a relation $Grant(x, y, h, p, v)$ which is true when entity x grants to entity B the permission to connect to machines in the domain h at port number p during the time v. Here the domains of x and y are strings, the domain of h is hierarchical, the domain of p is the integers,and the domain of v is the real numbers. Then the above policies

can be represented as follows:

$$\begin{aligned}
Grant(\text{“}A'', \text{“}B'', h, p, v) \quad &:\!\!- \quad h \prec \langle edu, stanford\rangle, \\
&\quad\; p = 80, 0 \le v, v \le 8.\\
Grant(\text{“}A'', x, h, p, v) \quad &:\!\!- \quad Grant(B, x, h, p, v), \\
&\quad\; h \prec \langle edu, stanford\rangle, \\
&\quad\; p = 80, 0 \le v, v \le 8.\\
Grant(\text{“}B'', \text{“}D'', h, p, v) \quad &:\!\!- \quad h \prec \langle edu, stanford, cs\rangle, \\
&\quad\; p = 80, 5 \le v, v \le 12.
\end{aligned}$$

From the above, we can conclude the following:

$$\begin{aligned}
Grant(\text{“}A'', \text{“}D'', h, p, v) \quad &:\!\!- \quad h \prec \langle edu, stanford, cs\rangle, \\
&\quad\; p = 80, 5 \le v, v \le 8.
\end{aligned}$$

Hierarchical domains are easy to add to the framework of constraint databases, because conjunctions of hierarchical constraints admit variable elimination as shown in [25]. In fact, we can show that Directed Acyclic Graph (DAG) domains also admit variable elimination. The domain names can for a DAG if there are several aliases for the same machine. We say that the hierarchical constraint $g = h$ holds if $g \prec h$ and $h \prec g$ both hold, that is, g and h are identical names or aliases for the same entity.

With aliases it could happen that two different domain names refer to the same entity. For example, the aliases $nsf.gov$ and $USAnsf.gov$ may refer to the same entity. This is permissible. Abbreviations are also permissible. For example, $a.b.c.d$ and $a.d$ could also refer to the same entities. However, cycles are not permissible with DAG domains. For example, $a.b.c.d$ and $a.b$ should refer to different machines otherwise there would be a cycle. This means that the hierarchical constraints $a.b = a.b.c.d$ and $a.b \prec a.b.c.d$ are not allowed.

THEOREM 7.1. *Conjunctions of Directed Acyclic Graph domains admit variable elimination.*

PROOF. Suppose we would like to eliminate the hierarchical variable x from a conjunction of hierarchical constraints S. First we rewrite S into S' by changing any hierarchical constraint of the form $g = h$ into a conjunction of $g \prec h$ and $h \prec g$. Now we need to eliminate x only from S'. After elimination, we will have a new set of constraints S'' that will consist of all the constraints in S' that do not contain the variable x, plus constraints of the form $y \prec z$ whenever there are constraints $y \prec x$ and $x \prec z$ in S'. If the y and z are constants, then we have to check that no cycle is created, that is y is not a string that is a substring of z. If it is, then we return *false*, else we return S''. □

8. DIOPHANTINE EQUATIONS

Diophantus, an ancient Greek mathematician, was one the earliest mathematicians to study integer solutions to polynomial equations. Hence polynomial equations over the integers are called Diophantine polynomial equations. Fining a general decision algorithm for Diophantine polynomial equations is known as Hilbert's 10th problem, after David Hilbert who gave this as the 10th problem in a list of major outstanding open problems in 1900 in a famous address to the International Congress of Mathematicians. It was not until 1970 that Y. Matiyasevich, based on important partial results by M. Davis and J. Robinson, proved that there is no general decision algorithm, that is, Hilbert's 10th problem is undecidable. For a complete history of this problem and its solution see Matiyasevich's book [27].

In spite of this undecidability result, it is often possible to solve various specific Diophantine polynomial equations. Solving these is often quite difficult. A well-known example is the equation

$$a^4 + b^4 + c^4 = d^4$$

which was conjectured by Euler to have no integer solutions. (In fact, Euler's conjecture was more general, saying that there is no nth power that is the sum of less than n other nth powers.) However, in 1987 Elkies gave a solution. Using a computer search, a few years later Frye gave another solution, which is the smallest:

$$414560^4 + 217519^4 + 95800^4 = 422481^4$$

It is surprising that the smallest solution to such an equation turns out to be so big.

We can give a simple computer search method for any Diophantine polynomial equation using the results of Section 3. This is because we can express any Diophantine polynomial equation using only the D and M relations of Section 3. Then by choosing repeatedly smaller ls, we get better lower and upper bounds on the solutions. If the lower bound ever becomes more than the empty-set, then we know for sure that there is a solution. Similarly, if the upper bound becomes the empty-set, then we know for sure that there is no solution. Otherwise, we have to choose a smaller l to get a better result.

EXAMPLE 8.1. We can express Euler's equation above as follows:

$$\begin{aligned}
Quad(x, m) \quad &:\!\!- \quad M(x, x, n),\ M(n, n, m).\\
Euler(a, b, c, d) \quad &:\!\!- \quad Quad(a, a'),\ Quad(b, b'),\\
&\quad\; Quad(c, c'),\ Quad(d, d'),\\
&\quad\; D(d', c', u),\ D(u, b', a').
\end{aligned}$$

For $l = -15$ the lower bound of *Euler* will be the $\emptyset$. However, as we choose smaller and smaller l, the above solution will be eventually part of the lower bound. Since reaching that requires too much calculation, let us just consider the *Quad* relation. When $l = -15$, its lower bound is:

$$Q(Quad)_l \;=\; \{(1, 1),\ (2, 16),\ (3, 81)\}$$

and its upper bound is:

$$Q(Quad)^l \;=\; Q(Quad)_l \cup \{(x, m)\ :\ x \ge 16,\ m \ge 256\}$$

where x and m are integer variables. The upper bound follows from the fact that $\{(x, x, 256) : x \ge 16\}$ and $\{(256, 256, m) : m \ge 256\}$ are both in the upper bound of the M relation.

Although the lower and the upper bounds do not match, it is possible to use the partial information to show, for example, that $(4, 178) \notin Quad$. Hence the lower and the upper bounds do not need to match to get some useful information from the approximate evaluation.

9. MOVING POINTS

There is a growing interest in representing moving objects. Cai and Revesz [33], Chomicki and Revesz [6, 5], Güting et al. [15], Kollios et al. [23], Saltenis et al. [35], and Wolfson et

al. [37] describe moving object data models and techniques to query moving objects. Constraint databases are a natural representation of moving objects as explained in this section.

Suppose that two cars, which both move linearly in the plane, want to radio-communicate with each other. Suppose also that radio communication is possible only within 3 units distance. What is the best time to attempt the radio communication? Intuitively, the best time would be when the two cars are closest to each other, hence that time instance needs to be found. We show that it can be found using only linear constraints.

The two cars can be represented by two constraint database relations $P_1(x, y, t)$ and $P_2(x, y, t)$. For example, an input database instance could be the following (see also Figure 7):

$$\begin{array}{lll} P_1(x,y,t) & :\!- & x = t,\ y = 2t + 4. \\ P_2(x,y,t) & :\!- & x = 3t,\ y = 4t. \end{array}$$

Suppose that we would like to find the time instance t when the two cars are closest to each other. We can define first the difference between the two cars at any time t as follows:

$$\begin{array}{lll} \Delta P(x,y,t) & :\!- & P_1(x_1,y_1,t),\ P_2(x_2,y_2,t), \\ & & x = x_2 - x_1,\ y = y_2 - y_1. \end{array}$$

ΔP is also a moving point in the plane as shown in Figure 8. The difference between the two cars is exactly the difference between ΔP and the origin at any time t. Therefore, the two cars are closest to each other when ΔP is closest to the origin. Now the projection of ΔP onto the plane is a line, which is the path along which ΔP travels. We can find this by:

$$\Delta Pline(x, y) \quad :\!- \quad \Delta P(x, y, t).$$

Let us now take the line which goes through the origin and is perpendicular to $\Delta Pline$. If (x_1, y_1) and (x_2, y_2) are two points on $\Delta Pline$, then the slope of $\Delta Pline$ is:

$$\frac{y_2 - y_1}{x_2 - x_1}.$$

The perpendicular line will have a negative reciprocal slope and will go through the origin. Hence its line equation is:

$$y \quad = \quad -\frac{x_2 - x_1}{y_2 - y_1} x \tag{12}$$

Now we can chose any two distinct points on the line $\Delta Pline$ for expressing the line equation. Let us choose (x_1, y_1) to be the intersection point of $\Delta Pline$ and the line perpendicular to it and going through the origin. Further, let us chose the second point (x_2, y_2) such that

$$y_2 \quad = \quad x_1 + y_1. \tag{13}$$

Clearly, this is always possible to do when the line is not vertical. Now what is the intersection point? It will satisfy Equations (12) and (13), that is:

$$\begin{array}{rcl} y_1 & = & -\dfrac{x_2 - x_1}{y_2 - y_1} x_1 \\ x_1 & = & y_2 - y_1. \end{array} \tag{14}$$

The above can be simplified to:

$$\begin{array}{rcl} y_1 & = & x_1 - x_2 \\ x_1 & = & y_2 - y_1. \end{array} \tag{15}$$

Therefore, if $\Delta Pline$ is not vertical, that is, $x_1 \neq x_2$, then the point of $\Delta Pline$ that is closest to the origin is is exactly the intersection point, hence:

$$\begin{array}{lll} Closest\_Point(x_1, y_1) & :\!- & \Delta Pline(x_1, y_1), \\ & & \Delta Pline(x_2, y_2), \\ & & y_1 = x_1 - x_2, \\ & & x_1 = y_2 - y_1, \\ & & x_1 \neq x_2. \end{array}$$

Otherwise, if $\Delta Pline$ is vertical, that is, for any two different points $x_1 = x_2$, then the closest point is:

$$\begin{array}{lll} Closest\_Point(x_1, 0) & :\!- & \Delta Pline(x_1, 0), \\ & & \Delta Pline(x_2, y_2), \\ & & x_1 = x_2, \\ & & y_2 \neq 0. \end{array}$$

The time when the two cars are closest to each other is:

$$Closest\_Time(t) \quad :\!- \quad \Delta P(x, y, t),\ Closest\_Point(x, y).$$

Finally, the shortest distance between the cars is:

$$\begin{array}{lll} Shortest(d) & :\!- & Closest\_Time(t), \\ & & P_1(x_1, y_1, t), \\ & & P_2(x_2, y_2, t), \\ & & d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2. \end{array}$$

The closest time can be found using only linear constraints, but the calculation of the shortest distance requires polynomial constraints.

Example 9.1. Let us look at what will happen when we have the input database instance P_1 and P_2 as given above. In that case, we obtain:

$$\begin{array}{lll} \Delta P(x,y,t) & :\!- & x = 2t,\ y = 2t - 4. \\ \Delta Pline(x,y) & :\!- & x = y + 4. \end{array}$$

For $Closest\_Point$ the constraint rule application yields:

$$\begin{array}{lll} Closest\_Point(x_1, y_1) & :\!- & x_1 = y_1 + 4, \\ & & x_2 = y_2 + 4, \\ & & y_1 = x_1 - x_2, \\ & & x_1 = y_2 - y_1, \\ & & x_1 \neq x_2. \end{array}$$

Eliminating x_2 and y_2 we get:

$$\begin{array}{lll} Closest\_Point(x_1, y_1) & :\!- & x_1 = 2, \\ & & y_1 = -2. \end{array}$$

Finally, the closest time is calculated as:

$$Closest\_Time(t) \quad :\!- \quad x = 2t, y = 2t - 4, x = 2, y = -2.$$

Eliminating x and y we get:

$$Closest\_Time(t) \quad :\!- \quad t = 1.$$

Therefore, the two cars are closest at time 1. It is at that time that the two cars should attempt to radio-communicate with each other. We can also calculate that:

$$Shortest(d) \quad :\!- \quad d = \sqrt{8}.$$

This is less than 3 units, hence radio communication should be possible.

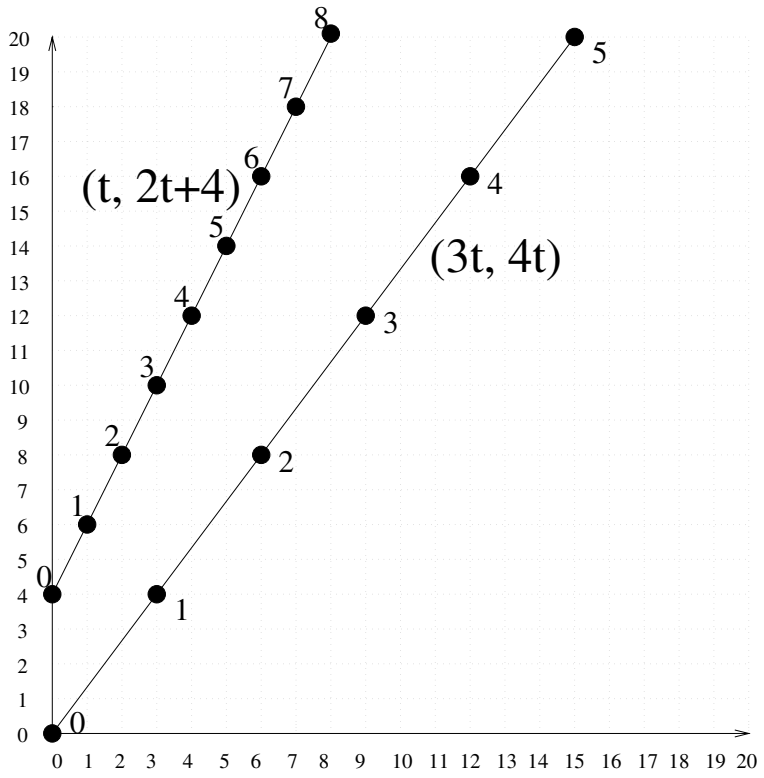

Figure 7: Two cars moving in the plane.

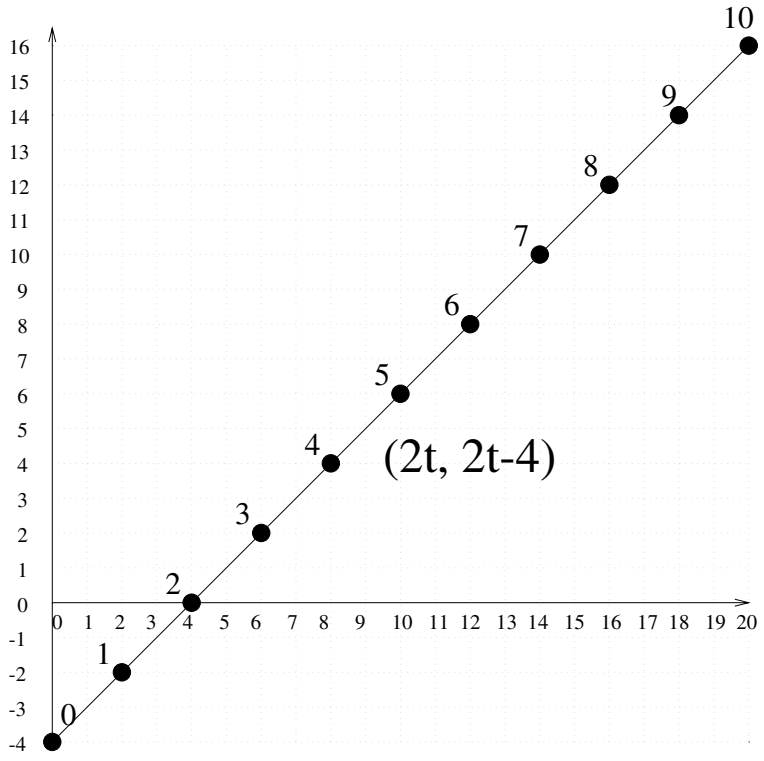

Figure 8: The ΔP moving point.

9.1 Piecewise Linear Movements

Suppose that the two moving points move not on a single straight line, but along a piecewise linear path. In this case an instance could be the following:

$$\begin{array}{lll} P_1(x,y,t,b,e) & :\!- & x=t,\ y=2t+2,\ b=0,\ e=5. \\ \vdots & \vdots & \vdots \\ P_2(x,y,t,b,e) & :\!- & x=3t,\ y=4t,\ b=0,\ e=5. \\ \vdots & \vdots & \vdots \end{array}$$

We added the attributes b and e to record the beginning and the end of each time interval. We assume that we have the same set of time intervals for the two moving points. We can similarly add these two attributes to each of the other relations we defined above. Then we can find what time would the two cars be closest if they were traveling infinitely long along their paths in each interval. The actual closest time in each interval is that time or one of the endpoint of the time interval.

$$\begin{array}{lll} Closest\_Time\_I(t,b,e) & :\!- & Closest\_Time(t,b,e), \\ & & b \le t,\ t \le e. \\ \\ Closest\_Time\_I(b,b,e) & :\!- & Closest\_Time(t,b,e), \\ & & t \le e. \\ \\ Closest\_Time\_I(e,b,e) & :\!- & Closest\_Time(t,b,e), \\ & & t \ge e. \end{array}$$

Now let us calculate the minimum distance between the cars in each interval.

$$\begin{array}{lll} Shortest\_I(d,b,e) & :\!- & Closest\_Time\_I(t,b,e), \\ & & P_1(x_1,y_1,t,b,e), \\ & & P_2(x_2,y_2,t,b,e), \\ & & d^2=(x_1-x_2)^2+(y_1-y_2)^2. \end{array}$$

Finally, to find the minimum distance in all the intervals, we use a Datalog query with the Min aggregate function:

$$Shortest(Min\langle d\rangle) \ :\!- \ Shortest\_I(d,b,e).$$

10. APPLICATION DEVELOPMENT

We have discussed the Datalog query language in this paper. The SQL query language can also be used on constraint databases. Both of these languages are designed for application programmers. In typical applications, the user only sees a list of menu options that are implemented on top of a constraint database system.

The application program can be a high level program that calls the constraint database using either Datalog or SQL queries. Next we discuss a sample application using the MLPQ (short for *Management of Linear Programming Queries*) constraint database system [34, 22].

The sample application that we consider is a simple case of production planning. Suppose that we know that a workers in b hours produce c items of some product. We assume that a, b, and c are fixed positive constant values. There are three menu options that we need to implement.

1. Find the number of workers needed to produce in y hours z items.
2. Find the number of hours needed to produce by x workers z items.
3. Find the number of items produced by x workers in y hours.

In each of the above menu options, the user is asked to enter the value of the variables in a dialog box after a prompt.

Suppose that the three menu options are already implemented in a slightly different form in MLPQ as follows.

$$\begin{array}{lll} Workers\_Needed(x) & :\!- & Items\_per\_Hours(n), \\ & & x=n(ab/c). \\ \\ Hours\_Needed(y) & :\!- & Items\_per\_Workers(m), \\ & & y=m(ab/c). \\ \\ Items\_Produced(z) & :\!- & Workers\_times\_Hours(l), \\ & & (ab/c)z=l. \end{array}$$

This implementation requires three input relations with self-explanatory names. Therefore, the only thing that the application programmer needs to do is to find the value of these input relations before calling the MLPQ system. This can be done in the embedding program.

The above illustrates how we can implement in MLPQ queries that actually require polynomial constraints. The implementation assumes that not only the raw value of the input variables are available but also their products or quotients if needed. The application developer needs only to transform the raw inputs of the user to the slightly more sophisticated inputs of the MLPQ queries. But he application developer does not need to worry about more details.

In the above example, the hidden details are not too difficult to reproduce, although they may be more difficult in other examples. We can make the following assumptions. First, the production increases linearly with the number of workers. Second, the production also increases linearly with the number of hours. From this we can derive the formula that must hold in all cases:

$$abz = xyc$$

From this we can see that the menu options can be also implemented by:

$$\begin{array}{lll} Workers\_Needed(x) & :\!- & Given(y,z),\ abz=xyc. \\ \\ Hours\_Needed(y) & :\!- & Given(x,z),\ abz=xyc. \\ \\ Items\_Produced(z) & :\!- & Given(x,y),\ abz=xyc. \end{array}$$

This is also possible to use if we have a polynomial constraint database system or a linear constraint database system that can propagate the constant values. For example, in the first menu option, since y and z are constants in each case, and a linear constraint database system with constant propagation can substitute their values into the equation $ny = z$, and then obtain a linear constraint problem.

11. CONCLUSIONS

The above brief review of constraint databases focused on some basic concepts of Datalog query evaluation and a few recent and growing application areas. Clearly, the constraint database area has many achievements. Paris Kanellakis would have been pleased to see them. However, the big question is whether constraint databases will ever be adopted by industry. We can give a few thoughts on that subject.

In general, the database systems area is very central to computer science, because databases are usually a part of any serious software development today. However, relational databases have some major limitations. Let me give a few examples.

1. Databases use query languages, but those are limited. Therefore, most software engineers do not have a deep appreciation for databases, they view them as a primitive tool.

2. Many relational databases provide only tables instead of visualizations. Even when some visualizations are provided, for example, pie charts, they are usually more primitive than the ones used in computer graphics and do not involve animation or motion, an important milestone in visualization.

3. Databases need a good user interface because they are supposed to be usable by naive users. In the early 1970s, the SQL language was cutting edge in its simplicity for naive users. However, the field of human-computer interaction has undergone major changes, and databases did not always keep up with those developments. While databases can be made web-accessible, the key component of the webpages seems to be always some menu of queries. Menus, however, loose one of the biggest advantages of relational databases, namely ad-hoc queries. The flexibility of menus even if they allow the entering of a few parameters is just not anywhere close to the expressivity of even a limited query language like SQL.

Let us look at how constraint databases improve relational databases with respect to the above three items.

First, even standard SQL or Datalog becomes a more powerful query language when it is used to query constraint databases instead of only relational databases. Hence (1) above is certainly improved and without any significant extra complexity in the query languages that need to be learned by the application developers.

Second, it was recognized early on that constraints are the last thing that users want to see. Visualization plays an important role in constraint databases such as DEDALE [13, 14] MLPQ/PReSTO [34, 22, 4], and CQA/CDB [12]. Hopefully, this positive trend continues, and future constraint database systems will be able to give computer graphics-like quality visualizations. Hence (2) is also significantly improved by constraint databases.

Third, the user interfaces of constraint databases need more development. Even with menus more should be tried, in particular, menu prompts should ask not only for specific strings or numbers but also for relationships between two or more different attributes. For example, which object is heavier? The translation from the users' entries into constraints is the easy part. The harder part is to develop a user-friendly interface that allows the entry of various constraint relationships. There is a large scope of work to be done here. Hence (3) still needs to improve. This is a nontraditional area for most database researchers, but it is important for developing practical constraint database systems.

In summary, constraint databases lift some of the limitations of relational databases. However, to be successful constraint database research needs more input from non-database and non-constraint researchers, especially from researchers in the areas of computer graphics and human-computer interaction. A healthy cooperation among these fields could yield major results in the future and make constraint databases really enticing to be adopted by industry.

12. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 1981.

[4] M. Cai, D. Keshwani, and P. Revesz. Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 430–44. Springer-Verlag, 2000.

[5] J. Chomicki and P. Revesz. Constraint-based interoperability of spatiotemporal databases. *Geoinformatica*, 3(3):211–43, 1999.

[6] J. Chomicki and P. Revesz. A geometric framework for specifying spatiotemporal objects. In *Proc. International Workshop on Time Representation and Reasoning*, pages 41–6, 1999.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[8] G. Delzanno and A. Podelski. Model checking in CLP. In *2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 74–88. Springer-Verlag, 1999.

[9] L. Fribourg and J. D. C. Richardson. Symbolic verification with gap-order constraints. In *Proc. Logic Program Synthesis and Transformation*, volume 1207 of *Lecture Notes in Computer Science*, pages 20–37, 1996.

[10] I. Geist. A framework for data mining and KDD. In *Proc. ACM Symposium on Applied Computing*, pages 508–13. ACM Press, 2002.

[11] I. Geist and K.-U. Sattler. Towards data mining operators in database systems: Algebra and Implementation. In *Proc. 2nd International Worskhop on Databases, Documents, and Information Fusions*, 2002.

[12] D. Goldin, A. Kutlu, M. Song, and F. Yang. The constraint database framework: Lessons learned from cqa/cdb. In *Proc. International Conference on Data Engineering*, 2003.

[13] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE system for complex spatial queries. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 213–24, 1998.

[14] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In *ACM Symposium on Geographic Information Systems*, pages 106–11, 1998.

[15] R.H. Güting, M.H. Böhlen, M. Erwig, C.C. Jenssen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25, 2000.

[16] P. Halmos. *Lectures on Boolean Algebras*. Springer-Verlag, 1974.

[17] R. Helm, K. Marriott, and M. Odersky. Spatial query optimization: From Boolean constraints to range queries. *Journal of Computer and System Sciences*, 51(2):197–201, 1995.

[18] J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proc. 14th ACM Symposium on Principles of Programming Languages*, pages 111–9, 1987.

[19] T. Johnson, L. V. Lakshmanan, and R. T. Ng. The 3W model and algebra for unified data mining. In *Proc. IEEE International Conference on Very Large Databases*, pages 21–32, 2000.

[20] P. C. Kanellakis, G. M. Kuper, and P. Revesz. Constraint query languages. In *Proc. ACM Symposium on Principles of Database Systems*, pages 299–313, 1990.

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

[22] P. Kanjamala, P. Revesz, and Y. Wang. MLPQ/GIS: A GIS using linear constraint databases. In C. S. R. Prabhu, editor, *Proc. 9th COMAD International Conference on Management of Data*, pages 389–93, 1998.

[23] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proc. ACM Symposium on Principles of Database Systems*, pages 261–72, 1999.

[24] G. M. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer-Verlag, 2000.

[25] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In V. Dahl and P. Wadler, editors, *Proc. 5th International Symposium on Practical Aspects of Declarative Languages*, volume 2562 of *Lecture Notes in Computer Science*, pages 58–73. Springer-Verlag, 2003.

[26] K. Marriott and M. Odersky. Negative Boolean constraints. *Theoretical Computer Science*, 160(1–2):365–80, 1996.

[27] Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

[28] K. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[29] P. Revesz. *Constraint Query Languages*. PhD thesis, Brown University, May 1991.

[30] P. Revesz. Refining restriction enzyme genome maps. *Constraints*, 2(3–4):361–75, 1997.

[31] P. Revesz. Reformulation and approximation in model checking. In B. Choueiry and T. Walsh, editors, *Proc. 4th International Symposium on Abstraction, Reformulation, and Approximation*, volume 1864 of *Lecture Notes in Computer Science*, pages 124–43. Springer-Verlag, 2000.

[32] P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, New York, 2002.

[33] P. Revesz and M. Cai. Efficient querying of periodic spatio-temporal databases. *Annals of Mathematics and Artificial Intelligence*, 36(4):437–457, 2002.

[34] P. Revesz and Y. Li. MLPQ: A linear constraint database system with aggregate operators. In *Proc. 1st International Database Engineering and Applications Symposium*, pages 132–7. IEEE Press, 1997.

[35] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 331–42, 2000.

[36] P. van Beek and K. Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. In T. Walsh, editor, *Proc. 7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 625–39. Springer-Verlag, 2001.

[37] O. Wolfson, A. Sistla, B. Xu, J. Zhou, and S. Chamberlain. DOMINO: Databases for moving objects tracking. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 547–9, 1999.