7. DATALOG and Constraints

Peter Z. Revesz

7.1 Introduction

Recursion is an important feature to express many natural queries. The most studied recursive query language for databases is called DATALOG, an abbreviation for "*Data*base *log*ic programs." In Section 2.8 we gave the basic definitions for DATALOG, and we also saw that the language is not closed for important constraint classes such as linear equalities. We have also seen some results on restricting the language, and the resulting expressive power, in Section 4.4.1.

In this chapter, we study the evaluation of DATALOG with constraints in more detail, focusing on classes of constraints for which the language is closed. In Section 7.2, we present a general evaluation method for DATALOG with constraints. In Section 7.3, we consider termination of query evaluation and define *safety* and *data complexity*. In the subsequent sections, we discuss the evaluation of DATALOG queries with particular types of constraints, and their applications.

7.2 Evaluation of DATALOG with Constraints

The original definitions of recursion in Section 2.8 discussed unrestricted relations. We now consider constraint relations, and first show how DATALOG queries with constraints can be evaluated. Assume that we have a rule of the form

$$R_0(x_1, \ldots, x_k) \text{ :- } R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), \psi \ ,$$

as well as, for $i = 1, \ldots, n$, facts (in the database, or derived) of the form

$$R_i(x_{i,1}, \ldots, x_{i,k_i}) \text{ :- } \psi_i(x_{i,1}, \ldots, x_{i,k_i}) \ ,$$

where ψ_i is a conjunction of constraints. A *constraint rule application* of the above rule with these facts as input produces the following derived fact:

$$R_0(x_1, \ldots, x_k) \text{ :- } \varphi(x_1, \ldots, x_k) \ ,$$

where φ is a quantifier-free formula equivalent to

$$\exists * \ (\psi_1(x_{1,1}, \ldots, x_{1,k_1}) \wedge \cdots \wedge \psi_n(x_{n,1}, \ldots, x_{n,k_n}) \wedge \psi)$$

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

G. Kuper et al. (eds.), *Constraint Databases*

The *bottom-up constraint fixpoint evaluation* of DATALOG queries starts from the input facts and rules, and repeatedly applies constraint rule evaluation until no new facts can be derived and added to the database. Note that this means that any new fact that can be derived is *implied by*, not necessarily equal to any of the facts that we have already derived. We call the set of input and derived facts the *bottom-up constraint least fixpoint* of the query.

Proposition 7.2.1. *For each* DATALOG *query with constraints, the bottom-up constraint least fixpoint is equivalent to the least fixpoint.* □

Remember that a constraint tuple represents a possibly infinite unrestricted relation. Hence, a finite number of given or derived constraint tuples in the result of a bottom-up constraint evaluation least fixpoint may represent an infinite least fixpoint.

7.3 Termination, Safety, and Data Complexity

Proposition 7.2.1 implies that we can always evaluate the least fixpoint of a DATALOG program and a constraint database if the following two conditions are satisfied:

1. Variable (in particular, existential quantifier) elimination from a conjunction of atomic constraints is possible, so that each application of the immediate consequence operator can effectively be evaluated.
2. There must exist an integer n such that $T_Q^{n+1}(I) = T_Q^n(I)$, so that the immediate consequence operator needs to be applied only a finite number of times.

As we have seen (Example 2.8.1), this is not the case for DATALOG with linear equalities. For another example, consider the successor constraint on integers, that is, $x + 1 = y$, where x and y are variables. Let DATALOG^{+1} denote the set of DATALOG queries with the successor constraint. It is well-known that DATALOG^{+1} has the same expressive power as Turing machines. Therefore,

Theorem 7.3.1. *It is undecidable for* DATALOG^{+1} *queries in general whether the constraint evaluation terminates.* □

Some results on restricting DATALOG to guarantee termination were presented in Section 4.4.1.

7.4 DATALOG with Dense Order Constraints

Among the simplest types of constraints are the equality and order constraints $=$, $<$, $>$, $\leq$, and $\geq$ between constants and variables. In this section, we consider the case where the domain of the variables is the set of rational

numbers $\mathbb{Q}$. In this case, we show that any conjunction of order constraints can be written as a set of *r-configurations*.

The definition of *r-configuration* assumes a fixed set of rational numbers A, which is the set of constants that occur in either the input database or the DATALOG program.

Definition 7.4.1 (R-configuration). *An* r-configuration $\xi = (\vec{f}, \vec{l}, \vec{u})$ *of size* n *consists of a sequence* $\vec{f} = (f_1, \ldots, f_n)$, *where* $\{f_1, \ldots, f_n\} = \{1, \ldots, j\}$, *for some* $j \leq n$, *and two sequences* $\vec{l} = (l_1, \ldots l_n)$ *and* $\vec{u} = (u_1, \ldots, u_n)$, *where, for* $i = 1, \ldots, n$, $l_i \in A \cup \{-\infty\}$ *and* $u_i \in A \cup \{+\infty\}$, *such that:*

1. *for* $i = 1, \ldots, n$, $l_i \leq u_i$;
2. *there is no constant* c *in* A *with* $l_i < c < u_i$;
3. *whenever* $f_i < f_j$, *then* $l_i < u_j$; *and*
4. *whenever* $f_i = f_j$, *then* $l_i = l_j$ *and* $u_i = u_j$.

The idea behind r-configurations is as follows. Consider two points $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ in $\mathbb{Q}^n$. We want to know whether they can be distinguished using the order constraints and the available constants. We say that these points *can be distinguished* if either the relative order of the $x_1, \ldots, x_n$ is different from the relative order of $y_1, \ldots, y_n$, or, for some i, $1 \leq i \leq n$, x_i is in a different relation to some constant in A than to y_i. Each r-configuration characterizes a set of nondistinguishable points. On the one hand, the sequence $\vec{f}$ describes the relative order of $x_1, \ldots, x_n$, in other words, for $1 \leq i, j \leq n$, $x_i < x_j$ if and only if $f_i < f_j$. On the other hand, for $i = 1, \ldots, n$, l_i and u_i bound x_i from below and above by constants from $A \cup \{-\infty, +\infty\}$ in the tightest fashion possible.

Example 7.4.1. Assume that $A = \{0, 1, 2, 3\}$. The sequence of numbers $(0.5, 3.5, 1.5, 1.5, 2)$ can then be represented by the r-configuration consisting of

1. $\vec{f} = (1, 4, 2, 2, 3)$,
2. $\vec{l} = (0, 3, 1, 1, 2)$, and
3. $\vec{u} = (1, +\infty, 2, 2, 2)$

Definition 7.4.2. *The formula* $F(\xi)$, *free variables* $x_1, \ldots, x_n$, corresponding *to an r-configuration* $\xi = (\vec{f}, \vec{l}, \vec{u})$, *of size* n, *is the conjunction of:*

1. $x_i < x_j$, *whenever* $f_i < f_j$,
2. $x_i = x_j$, *whenever* $f_i = f_j$,
3. $l_i < x_i < u_i$ *whenever* $l_i < u_i$, *and*
4. $x_i = l_i$ *whenever* $l_i = u_i$.

The key feature of r-configurations is:

Lemma 7.4.3. *Let ψ be a first-order formula, with at most k free variables, in the language of $<$ and constants in A. Let ξ be an r-configuration of size k and $\models F(\xi)(a_1, \ldots, a_k)$ and $\models F(\xi)(a'_1, \ldots, a'_k)$, then $\models \psi(a_1, \ldots, a_k) \Leftrightarrow \models \psi(a'_1, \ldots, a'_k)$.* □

This means that we can represent the set of points that satisfied ψ as a set of r-configurations, the number of which is at most polynomial in in $|A|$. Since under inflationary semantics we can only add r-configurations at each iteration, we obtain a polynomial-time algorithm for inflationary DATALOG$^\neg$. (The same techniques can be used to show that first-order queries have data complexity LOGSPACE.)

Theorem 7.4.4. *The least fixpoint of* DATALOG$^{=,\leq,\geq,<,>}$ *programs over the rational numbers can be evaluated in* PTIME. □

By applying the same technique to each stratum in turn, we obtain:

Theorem 7.4.5. *The perfect fixpoint of stratified* DATALOG$^{=,\leq,\geq,<,>}$ *programs over the rational numbers can be evaluated in* PTIME *data complexity.* □

7.5 DATALOG with Gap-Order Constraints

7.5.1 General Theory

Order constraints with integers as the domain are more difficult to handle. The first problem is that they are not closed under variable elimination. For example, $\exists z\,(x < z \wedge z < y)$ is not expressible as a formula in which only variables x and y occur using the equality and order constraints in the previous section; however, we can express $\exists z\,(x < z \wedge z < y)$ as a formula in which only variables x and y occur using other types of order constraints, in this case, for example, $x + 1 < y$ or $x + 2 \leq y$, with the obvious meanings. We abbreviate the two constraints above as $x <_1 y$ and $x \leq_2 y$, respectively, and call them *gap-order* constraints. We call the subscript the *gap-value*. The gap-value is usually restricted to be a *nonnegative* integer, a restriction that we continue to assume except in Section 7.5.3. It can be shown that gap-order constraint formulae are closed under variable elimination.

A conjunction of gap-order constraints can be represented by a *gap-graph*. The vertices of a gap-graph are always variables and the two special constants l and u, where l is the smallest and u is the largest constant in the program or in the input database, not including gap-values. Figure 7.5.1 gives an example gap-graph with $l = 2$ and $u = 18$. The gap-graph represents the conjunction of the constraints $x_1 <_6 x_2$, $x_1 <_0 x_3$, $2 <_9 x_2$, $2 <_3 x_3$, $x_2 <_2 18$, and $x_3 <_7 18$.

We say that a gap-graph *dominates* another gap-graph if they have the same set of directed edges and, for each edge, the label in the first graph is

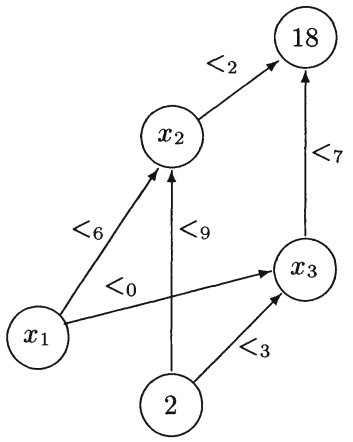

Fig. 7.5.1. Example of a gap-graph

always greater than, or equal to, the label in the second graph. It can be shown that during the constraint evaluation, it is enough to add only gap-graphs that do not dominate any gap-graphs already present. From this, we derive:

Theorem 7.5.1. *The least fixpoint of* DATALOG$^{\text{gap}}$ *programs over the integers can be evaluated in finite time.* □

7.5.2 Stratified DATALOG with Gap-Order Constraints

Stratified negation is more complicated for gap-order constraints than for order; indeed, successor constraints can be expressed using stratified negation and gap-order constraints. Therefore, by Theorem 7.3.1, general stratified DATALOG$^{\text{gap}}$ cannot be evaluated; however, it is possible to identify syntactically a safe subclass of stratified DATALOG$^{\text{gap}}$ programs. This we do as follows.

Let Π be a DATALOG$^{\text{gap}}$ program. We assign to each input constraint relation of arity k a type that is called an *arguments connection graph* or *congraph*. Intuitively, each congraph shows the possible connections via $<_g$-constraints between the arguments of a relation. Each congraph of arity k is a directed graph $C(V, E, \equiv, \not\equiv)$, where V is the set of argument variables, $E \subseteq V \times V$ is the set of edges, $\equiv \subseteq V \times V$ is the set of equalities between the argument variables, and $\not\equiv \subseteq V \times V$ is the set of inequalities between the argument variables.

We shall say that a constraint relation $p(x_1, \ldots, x_k)$ has congraph type $C(V, E, \equiv, \not\equiv)$, or is *valid* with respect to $C(V, E, \equiv, \not\equiv)$, if, for every constraint $x_i <_g x_j$ in p, the edge $(x_i, x_j) \in E$, and, for every constraint $x_i = x_j$ (or $x_i \neq x_j$) in p, $(x_i, x_j) \in \equiv$ (or $(x_i, x_j) \in \not\equiv$). We sometimes abbreviate the latter condition as $x_i \equiv x_j$ (or $x_i \not\equiv x_j$).

Note that, in the above, we assume that each relation p is rectified, that is, if p is an EDB (extensional database) relation, then it always appears in the input database, and, if it is an IDB (intentional database, in other words, a derived) relation, it always appears in the head of rules with the same list of argument variables. (In the body of the rules the relation symbol p may appear with a different list of variables than in its rectified form.) This restriction is not very significant, since it is easy to put a DATALOG program into an equivalent rectified form.

Let $C = (V, E, \equiv, \not\equiv)$ be a congraph of a relation. The *transitive closure* of C is $C^* = (V, E^*, \equiv^*, \not\equiv)$, where $\equiv^*$ is the congruence closure of the $\equiv$ relation, and $(x_i, x_j) \in E^*$ if and only if it is in E or if there are pairs (x_i, z_1), (z_2, z_3), ..., (z_l, x_j) in $E \cup \equiv$ with at least one pair in E.

Let r be any rule with variables $x_1, \ldots, x_n$ of the form

$$A_0 \text{ :- } A_1, A_2, \ldots, A_l \ .$$

Then, the *congraph of* r, $C_r = (V_r, E_r, \equiv_r, \not\equiv_r)$, is the transitive closure of the union of the congraphs of $A_1, \ldots, A_l$ after the necessary renamings. If two argument variables x_i and x_j in the rectified form(s) of some relation(s) are renamed within the rule body by the same variable, then (x_i, x_j) is added to $\equiv_r$. We define the congraph of IDBs of any semipositive DATALOG program as the output of algorithm FIND-IDB-CONGRAPHS in Figure 7.5.2.

Note that the congraph of a relation R depends on the program Π. We make this explicit by using notations such as $E_{R\Pi}$. Let Π_1 and Π_2 be two semipositive DATALOG$^{\text{gap}}$ programs. We say that Π_1 is congraph compatible with Π_2 if and only if for each relation R that is common to both Π_1 and Π_2, $E_{R,\Pi_1} \subseteq E_{R,\Pi_2}$, $\equiv_{R,\Pi_1} \subseteq \equiv_{R,\Pi_2}$, and $\not\equiv_{R,\Pi_1} \subseteq \not\equiv_{R,\Pi_2}$.

We can now define the notion of a *safe* DATALOG$^{\text{gap}}$ program. The definition is by induction on the number of strata. For the base case, a semipositive DATALOG$^{\text{gap}}$ program is *safe* if and only if the congraph of any negated EDB in the program has an empty set of edges.

A stratified DATALOG$^{\text{gap}}$ program is called *safe* if and only if it consists of $\Pi_1 \cup \cdots \cup \Pi_n$, where each Π_i is a safe semipositive DATALOG$^{\text{gap}}$ program, and each Π_i is congraph compatible with $\Pi_{i+1}, \ldots, \Pi_n$. Note that by repeatedly calling algorithm FIND-IDB-CONGRAPHS (Figure 7.5.2) on each stratum of a stratified DATALOG program, we can test whether it is safe or not.

The following property can be shown for safe stratified DATALOG$^{\text{gap}}$ programs: if R is a relation that occurs negated at least once in the program, then R is representable by a set of gap-graphs that do not contain any directed edge between pairs of vertices labeled with variables. Because of this property, safe stratified DATALOG$^{\text{gap}}$ programs can be evaluated stratum by stratum like DATALOG$^{\text{gap}}$ programs, with negated relations replaced by their complement relations before the evaluation of each stratum; however, as we move from stratum to stratum, the values of l and u may increase by an exponential. Therefore,

ALGORITHM FIND-IDB-CONGRAPHS
INPUT. A semipositive DATALOG^{gap} program Π and a congraph for each EDB.
OUTPUT. A congraph of each IDB of Π.
METHOD.
FOR each IDB relation $p_m(x_1, \ldots, x_k)$ DO
  assign to p_m a congraph $C_m = (V_m, E_m, \equiv_m, \not\equiv_m) = (\{x_1, \ldots, x_k\}, \emptyset, \emptyset, \emptyset)$;
OD;
WHILE any changes in IDB congraphs DO
  FOR each rule r with head $p_m(x_1, \ldots, x_k)$ DO
    Find $C_r = (V_r, E_r, \equiv_r, \not\equiv_r)$, the congraph of rule r;
    Let $E_m := E_m \cup \{(x_i, x_j) \mid (x_i, x_j) \in E_r)\}$;
    Let $\equiv_m := \equiv_m \cup \equiv_r$;
    Let $\not\equiv_m := \not\equiv_m \cup \not\equiv_r$;
  OD
OD.

Fig. 7.5.2. Algorithm FIND-IDB-CONGRAPHS

Theorem 7.5.2. *The result of a safe stratified* DATALOG^{gap} *program over the integers can be evaluated in finite time. The data complexity of the tuple recognition problem for safe stratified* DATALOG^{gap} *is nonelementary.* □

Let us call *semantically safe* those programs that on each valid input database have a result that is representable by gap-order constraints. Obviously, syntactic safety implies semantical safety. It is interesting to consider whether the syntactically defined subclass of queries can be extended further and further until it includes all semantically safe queries. Unfortunately,

Theorem 7.5.3. *The set of all semantically safe stratified* DATALOG^{gap} *programs over the integers cannot be described syntactically.* □

We conclude this section with an example of a safe stratified DATALOG program.

Example 7.5.1. The following stratified DATALOG^{gap} program is safe, and computes the shortest path between pairs of cities. We assume that the input relation is $Distance(x, y, s_1, s_2)$, whose tuples are of the form $x = c_1 \wedge y = c_2 \wedge s_1 <_m s_2$, meaning that from a city named c_1, the city named c_2 can be reached by a direct path of length at most $m + 1$. The program is now as follows:

$$
\begin{array}{lll}
Shortest(x, y, s) & :- & Path(x, y, 0, s), \neg Not\_Shortest(x, y, s). \\
Not\_Shortest(x, y, s_2) & :- & Path(x, y, 0, s_1), Path(x, y, 0, s_2), \\
 & & s_1 < s_2. \\
Path(x, y, s_1, s_2) & :- & Path(x, z, s_1, s_3), Distance(z, y, s_3, s_2). \\
Path(x, y, s_1, s_2) & :- & Distance(x, y, s_1, s_2).
\end{array}
$$

7.5.3 DATALOG with Unrestricted Gap-Order Constraints

If we allow the gap-values to be any integer, either nonnegative or negative, then we can express the difference between two variables. This is because the *difference constraint* $x - y = c$, where x and y are variables and c is an integer, can be expressed by the conjunction of the unrestricted gap-order constraints $x \leq_c y$ and $y \leq_{-c} x$. Difference constraints have applications in temporal reasoning, as they allow us to talk about the past.

We define DATALOG$^{\text{diff}}$ to be the class of DATALOG queries with unrestricted gap-order constraints. Theorem 7.3.1 implies that DATALOG$^{\text{diff}}$ is not evaluable in finite time in general, because $x + 1 = y$ is also a difference constraint. The evaluation of DATALOG with unrestricted gap-order constraints may not terminate, because the constraint tuples created can have smaller and smaller gap-values. For example, when we eliminate the variable y from the expression $x \leq_{-5} y \wedge y \leq_{-8} z$, we create a new constraint of the form $x \leq_{-13} z$. Interestingly, there is an asymmetry between creating larger and larger gap-values and creating smaller and smaller gap-values. This asymmetry is due to the fact that the conjunction of two gap-order constraints between a pair of variables is equivalent to the gap-order constraint with the smaller gap-value. Hence, the creation of tuples with larger gap-values becomes superfluous after a while, while the creation of tuples with smaller gap-values may still add more information during the fixpoint evaluation. This leads to the idea of placing a limit l on the smallest bound allowed. To avoid smaller bounds than l, we may make two different modifications to the evaluation method:

Modification 1 Add a new constraint tuple to an output database relation only after changing the value of any bound b to $\max(b, l)$.

Modification 2 Add a new constraint tuple to an output database relation only after deleting from it any difference constraint that has a bound less than l.

For a DATALOG$^{\text{diff}}$ query with input database D and program Π, we denote by $\text{LFP}(\Pi(D), l\uparrow)$ and $\text{LFP}(\Pi(D), l\downarrow)$ the output of the first, respectively the second, modified evaluation algorithms. We can show the following.

Theorem 7.5.4. *For any* DATALOG$^{\text{diff}}$ *program* Π*, input database* D*, and constant* l*, the following is true:*

$$\text{LFP}(\Pi(D), l\uparrow) \subseteq \text{LFP}(\Pi(D)) \subseteq \text{LFP}(\Pi(D), l\downarrow) .$$

Furthermore, $\text{LFP}(\Pi(D), l\uparrow)$ *and* $\text{LFP}(\Pi(D), l\downarrow)$ *can be evaluated in finite time.* □

We can use the above to get better and better approximations using smaller and smaller values as bounds. In particular,

Theorem 7.5.5. *For any* DATALOG$^{\text{diff}}$ *program* Π, *input database* D, *and constants* l_1 *and* l_2 *such that* $l_1 \leq l_2 \leq 0$, *the following is true:*

$$\text{LFP}(\Pi(D), l_2 \uparrow) \subseteq \text{LFP}(\Pi(D), l_1 \uparrow) ; \qquad \text{and}$$
$$\text{LFP}(\Pi(D), l_1 \downarrow) \subseteq \text{LFP}(\Pi(D), l_2 \downarrow) .$$

□

7.6 DATALOG with Linear Constraints

Linear constraints are of the form $c_1 x_1 + \cdots + c_k x_k \geq b$, where, for $i = 1, \ldots, k$, c_i is a constant and x_i is a variable, and b is constant.

We distinguish between two types of linear constraints depending on the domain of the variables and constants: the domain of *rational linear constraints* is the set of rationals $\mathbb{Q}$ and the domain of *integer linear constraints* is the set of integers $\mathbb{Z}$.

We also consider two particular subtypes of (both rational or integer) linear constraints depending on the constant coefficients. A *positive linear constraint* is a linear constraint in which each coefficient is positive. A *negative linear constraint* is a linear constraint in which each coefficient is negative. We define DATALOG$^{\text{pos}}$ to be the DATALOG queries with only positive linear, equality, and (gap)-order constraints and DATALOG$^{\text{neg}}$ to be the DATALOG queries with only negative linear, equality, and (gap)-order constraints.

Theorem 7.6.1. *The least fixpoint of any* DATALOG$^{\text{pos}}$ *or* DATALOG$^{\text{neg}}$ *query is evaluable in closed form when the domain of the variables and constants is either* $\mathbb{Q}$ *or* $\mathbb{Z}$. □

7.7 DATALOG with Polynomial Constraints

In this section, we consider DATALOG programs with polynomial constraints over the real numbers. We illustrate the expressive power of this query language by showing how to test if a two-dimensional figure is topologically connected. More precisely, we turn DATALOG into a spatial query language, in the following referred to as *spatial* DATALOG, as follows:

- The underlying domain of the variables is the set of real numbers.
- The only EDB predicate is a binary predicate S, which is interpreted as the set of points in the spatial database, or equivalently, as a binary relation over $\mathbb{R}$.
- Database relations are required to be semi-algebraic.
- Polynomial inequalities are allowed in rule bodies.

Under the bottom-up semantics, the following fundamental closure property is satisfied by a spatial DATALOG program P:

If the input relation S is a spatial database, then every derived relation R obtained by a finite number of iterations of P is also a spatial database; moreover, a finite representation of R can be computed effectively.

Of course, in general the recursion may not terminate after a finite number of iterations. It is not clear if there are restrictions on the databases under consideration, or on the syntax of allowed spatial DATALOG programs, such as those for linear constraints in Section 4.4.1, that guarantee termination. Hence, termination of particular recursive spatial queries must be established by ad hoc arguments, if at all possible.

In this section, we study the topological connectivity test as an example query. It is known that topological connectivity of two-dimensional spatial figures is *not* expressible in FO + POLY (see Chapter 4). Topological connectivity is, however, a decidable property of spatial databases and is of great importance in many spatial database applications. In analogy with the classical graph connectivity query, which cannot be expressed in the standard relational calculus, but which can be expressed in languages that typically contain a recursion mechanism (such as DATALOG), we can consider spatial DATALOG as a suitable candidate language to express the topological connectivity query.

$$
\begin{array}{rcl}
\mathit{Obstructed}(x,y,x',y') & :- & \neg S(\bar{x},\bar{y}),\ \bar{x} = a_1 t + b_1,\ \bar{y} = a_2 t + b_2, \\
 & & 0 \leq t, t \leq 1, b_1 = x, b_2 = y, \\
 & & a_1 + b_1 = x', a_2 + b_2 = y'. \\
\mathit{Path}(x,y,x',y') & :- & \neg \mathit{Obstructed}(x,y,x',y'). \\
\mathit{Path}(x,y,x',y') & :- & \mathit{Path}(x,y,x'',y''),\ \mathit{Path}(x'',y'',x',y'). \\
\mathit{Disconnected} & :- & S(x,y),\ S(x',y'),\ \neg \mathit{Path}(x,y,x',y'). \\
\mathit{Connected} & :- & \neg \mathit{Disconnected}.
\end{array}
$$

Fig. 7.7.1. A spatial DATALOG program for piecewise linear connectivity

Figure 7.7.1 gives a program which first computes a relation *Path*, containing all pairs of points of the spatial database which can be connected by a straight line segment that is completely contained in the database. Next, this program computes the transitive closure of this relation. If the computation of the transitive closure ends and consists of all possible pairs of points of the database, the program returns *true*. In fact, this program tests for what is usually referred to as *piecewise linear connectivity*, which in general is a stronger condition than connectivity. However:

Theorem 7.7.1. *The program of Figure 7.7.1 correctly tests topological connectivity of linear spatial databases.* □

To establish this theorem, two things must be proved:

Correctness: Two points in S are in the same connected component of S if and only if they can be connected by a piecewise linear curve lying entirely in S.

Termination: The number of line segments needed to connect any such pair is bounded.

The second fact guarantees that the transitive closure will terminate. The first fact then establishes the correctness of the test for connectivity performed by the program after the transitive closure is completed.

The program of Figure 7.7.1, however, cannot be guaranteed to terminate on all spatial database inputs, not even on bounded ones. One reason why this program does not work correctly on all spatial databases is that one cannot, in general, connect two points on a curved line in the database by straight line segments. Another reason is the possible presence of "cusp-like" points on the borders of databases. This is illustrated by the following example. Consider the region described by $(x^2 + y^2 \geq 1 \wedge x > -1 \wedge y < 1) \vee (x = -1 \wedge y = 0)$. Then the point $(-1, 0)$ cannot be connected by a finite number of straight line segments with any interior point of the region.

$$
\begin{array}{rcl}
\mathit{Zero}(a,b,c) & :- & 0 \leq t, t \leq 1, at^2 + bt + c = 0 \\
\mathit{Obstructed}(x,y,x',y',a_1,b_1, & & \\
\quad c_1,a_2,b_2,c_2,a,b,c) & :- & \neg S(\bar{x},\bar{y}), \\
 & & (at^2+bt+c)\bar{x} = a_1t^2 + b_1t + c_1, \\
 & & (at^2+bt+c)\bar{y} = a_2t^2 + b_2t + c_2, \\
 & & 0 \leq t, t \leq 1, cx = c_1, cy = c_2, \\
 & & (a+b+c)x' = a_1 + b_1 + c_1, \\
 & & (a+b+c)y' = a_2 + b_2 + c_2. \\
\mathit{Path}(x,y,x',y') & :- & \neg \mathit{Zero}(a,b,c), \\
 & & \neg \mathit{Obstructed}(x,y,x',y', \\
 & & a_1,b_1,c_1,a_2,b_2,c_2,a,b,c). \\
\mathit{Path}(x,y,x',y') & :- & \mathit{Path}(x,y,x'',y''),\ \mathit{Path}(x'',y'',x',y'). \\
\mathit{Disconnected} & :- & S(x,y),\ S(x',y'),\ \neg \mathit{Path}(x,y,x',y'). \\
\mathit{Connected} & :- & \neg \mathit{Disconnected}.
\end{array}
$$

Fig. 7.7.2. A spatial DATALOG program for piecewise quadratic connectivity

The program of Figure 7.7.1 can be generalized to the one depicted in Figure 7.7.2. This program computes a relation *Path* in which all pairs of points of the database are contained that either be connected by a straight line segment that is completely contained in the database, or can be connected by arbitrary segments of conic sections. The transitive closure of this relation is computed and the program returns *true* if all pairs of points of the database are in the computed relation. In fact, this second program tests for what we could call *piecewise quadratic connectivity*, which is in general a stronger

condition than connectivity, but is weaker than piecewise linear connectivity. We have the following result.

Theorem 7.7.2. *The program of Figure 7.7.2 correctly tests topological connectivity of spatial databases that can be defined in terms of at most quadratic polynomials.* □

The program indeed works correctly on the before mentioned database described by $(x^2 + y^2 \geq 1 \wedge x > -1 \wedge y < 1) \vee (x = -1 \wedge y = 0)$. The point $(-1, 0)$ can be connected to an interior point of the database by a single segment of a conic section.

The program for piecewise quadratic connectivity of Figure 7.7.2 does not correctly test connectivity on arbitrary inputs. Consider the spatial database $S = \{(x, y) \mid x^3 + y^3 = 1\}$. This database is actually an algebraic curve. Furthermore, it is an elliptic curve and has genus 1; it is known that such curves cannot be parameterized by rational functions. Although S is connected, the program for piecewise quadratic connectivity will return *false*. The program will first insert all tuples (x, y, x, y) with $(x, y) \in S$ into the *Path* relation, and will then terminate after one iteration, concluding that not all pairs of points of S have been found.

The fact that not all algebraic curves of degree three or higher (like $x^3 + y^3 = 1$) can be parameterized makes it impossible to further generalize the approach of the piecewise linear and quadratic connectivity programs to higher degrees. The use of other algebraic representations of curves such as implicit definitions are unreliable to test connectivity. First of all, not all implicitly defined curves are connected. For instance, the hyperbola $xy = 1$ is not connected. Secondly, the presence of a parameter in parameterized curve segments makes it possible to speak of "a point p being *between* points q and r on a curve." With implicit definitions, this notion of betweenness is not obvious.

On the other hand, it can be shown that topological connectivity of compact (topologically closed and bounded) planar databases can be expressed in spatial DATALOG. This uses the fact that, locally around each of its points p, a spatial database is "conical" (homeomorphic to a cone with top p and as base the intersection of a small circle around p with the database; see Chapter 10). The connectivity test for compact planar spatial databases first determines (in FO + POLY) for each point a radius within which the database is conical. Then, all pairs of points within that radius are added to the relation *Path* and, finally, the recursion of spatial DATALOG is used to compute the transitive closure of *Path*.

Theorem 7.7.3. *There exists a spatial* DATALOG *program that correctly tests topological connectivity of compact spatial databases in the plane.* □

It is open whether topological connectivity of arbitrary (not necessarily compact) spatial databases can be implemented in spatial DATALOG.

7.8 DATALOG with Boolean Equality Constraints

7.8.1 General Theory

Boolean algebras are common in many applications in computer science. First, we give an overview of Boolean algebras, and then present some results about query evaluation.

A *Boolean algebra* is a structure $B = \langle D, \wedge, \vee, ', 0, 1\rangle$ such that $\langle D, \wedge, \vee\rangle$ is a distributive lattice, $'$ is a unary operation, and 0 and 1 are two constants such that

$$\begin{array}{rclrcl} 0' & = & 1 & 1' & = & 0 \\ x \vee x' & = & 1 & x \wedge x' & = & 0 \\ x \vee 1 & = & 1 & x \wedge 0 & = & 0 \\ x \vee 0 & = & x & x \wedge 1 & = & x \end{array}$$

Stone's theorem for Boolean algebras states that every Boolean algebra is isomorphic to a field of sets, and every finite Boolean algebra is isomorphic to the power set of a finite set; thus, there is – up to isomorphism – a unique finite Boolean algebra for every cardinality which is a power of two For $m \geq 0$, the Boolean algebra of cardinality 2^{2^m} is *the Boolean algebra freely generated by m generators* and is denoted by B_m. For $m = 0$, we have $B_0 = \langle\{0, 1\}, \wedge, \vee, ', 0, 1\rangle$.

Given a set of variables V and a set of constants C (other than 0 and 1), we can build *Boolean terms* from the function and constant symbols $\wedge$, $\vee$, $'$, 0, and 1, and the elements of V and C in the usual way. A (B, σ)-*interpretation* consists of a Boolean algebra B and a mapping σ of the constant symbols C to the elements of B. Given a (B, σ)-interpretation, and an element of B for each variable in V, we can evaluate Boolean terms in the usual way to an element of B. We shall use the letter t to denote Boolean terms, or, more fully, $t(x_1, \ldots, x_n, c_1, \ldots, c_m)$, when t is a Boolean term with variables x_1, $\ldots$, x_n and constants c_1, $\ldots$, c_m. A *Boolean equation* is an expression of the form $t_1 = t_2$, with t_1 and t_2 Boolean terms. Boolean equations are evaluated in the usual way.

In order to evaluate DATALOG queries, we need a quantifier elimination method for Boolean constraints. This can be derived from Boole's Lemma, which says the following.

Lemma 7.8.1 (Boole's Lemma). *Let $t(x_1, \ldots, x_n, c_1, \ldots, c_m)$ be a Boolean term over some set of variables V and some set of constants C, and consider a (B, σ)-interpretation. The set of solutions of*

$$\exists x_1 \, (t(x_1, x_2, \ldots, x_n, c_1, \ldots, c_m) = 0)$$

is the same as the set of solutions of

$$t(1, x_2, \ldots, x_n, c_1, \ldots, c_m) \vee t(0, x_2, \ldots, x_n, c_1, \ldots, c_m) = 0 \, .$$

□

Because of the quantifier elimination provided by Boole's Lemma and the fact that the number of elements of a Boolean algebra is at most 2^{2^m} when generated (freely or nonfreely) by m constants, it is possible to show the following result.

Theorem 7.8.2. *Let Q be any* DATALOG *query with Boolean equality constraints. Given a (B, σ)-interpretation, Q can be evaluated bottom-up in closed form.* □

7.8.2 Application: Adder Circuit

We illustrate the use of Boolean constraints by an example of a simple binary adder circuit. In this case, there are no special constants ($C = \emptyset$, $m = 0$), and the freely generated Boolean algebra B_0 will consist of only two elements, that is, 0 meaning *false* and 1 meaning *true*. In the remainder of this section "$\oplus$" denotes the "exclusive or", which can be defined as $x \oplus y = (x \wedge y') \vee (x' \wedge y)$.

An adder circuit can be built from two half-adder circuits. We first define a half-adder by a single database fact.

$$\mathit{Halfadder}(x, y, z, w) \text{ :- } x \oplus y = z,\ x \wedge y = w.$$

where x and y are the input variables, z is the sum, and w is the carry. The adder circuit can then be described by the use of two half-adder circuits and an extra constraint, as follows:

$$\begin{aligned}\mathit{Adder}(x, y, c, s, d) \text{ :- } & \mathit{Halfadder}(x, y, s_1, c_1),\ \mathit{Halfadder}(s_1, c, s, c_2),\\ & d = c_1 \vee c_2.\end{aligned}$$

where x and y are the input variables, c is the carry-in, s is the sum, and d is the carry-out.

Let us now consider the bottom-up evaluation of the above program. Turning the two constraints in the body of the first rule into a single equivalent constraint, the evaluation algorithm yields

$$\mathit{Halfadder}(x, y, z, w) \text{ :- } (x \oplus y \oplus z) \vee ((x \wedge y) \oplus w) = 0.$$

The substitution into the body of the second rule yields

$$\begin{aligned}\mathit{Adder}(x, y, c, s, d) \text{ :- } & (c_1 \vee c_2) \oplus d = 0,\\ & (x \oplus y \oplus s_1) \vee ((x \wedge y) \oplus c_1) = 0,\\ & (s_1 \oplus c \oplus s) \vee ((s_1 \wedge c) \oplus c_2) = 0.\end{aligned}$$

By transforming the three constraints in the body into one and using Boole's Lemma to eliminate s_1, c_1, and c_2, the evaluation finally yields

$$\begin{aligned}\mathit{Adder}(x, y, c, s, d) \text{ :- } & (x \oplus y \oplus c \oplus s)\\ & \vee ((x \wedge y) \oplus (x \wedge c) \oplus (y \wedge c) \oplus d) = 0.\end{aligned}$$

The above is a correct description of a binary adder circuit in terms of the primitive inputs x, y, c, s, and d.

7.9 Bibliographic Notes

Proposition 7.2.1 was first proved by Jaffar and Lassez in [JL87] in the context of constraint logic programs. The basic definitions of the syntax and semantics of DATALOG queries of constraint databases were introduced in [KKR90]. Theorem 7.3.1 is common knowledge and has been observed by many people.

Rational-order constraints were studied first in [KKR90, KKR95]. The technique of r-configurations (Definition 7.4.1) was introduced there, along with Example 7.4.1 and Theorem 7.4.4. Theorems 7.4.5 and 7.5.2 are from the same paper, although they considered inflationary instead of stratified negation. Rational-order constraints and sets were combined in [GS95a].

Gap-order constraints on integers were introduced in [Rev93]. The technique of gap-graphs, as well as Theorem 7.5.1 are from this same paper. The problem whether a given relation is empty in the least fixpoint of a DATALOG with gap-order query was studied in [CM93]. Safe stratified DATALOG queries with gap-order constraints are described in [Rev95b, Rev98d]. Stolbushkin and Taitslin [ST95] raised the question whether syntactic safety can be extended to include semantical safety. Theorem 7.5.3 is from [ST98]. The application to shortest distance is also from [Rev95b, Rev98d]. Classes of safe DATALOG queries with unrestricted gap-order constraints are studied in [Rev99].

Classes of safe DATALOG queries with positive or negative linear constraints, or restricted linear constraints that express vector addition or matrix multiplication, were studied in [Rev98c]. Kuijpers, Paredaens, Smits, and Van den Bussche have studied termination properties of DATALOG programs with polynomial constraints over the real numbers [KPSV96]. They have also studied the topological connectivity test and given an implementation of a query in spatial DATALOG that correctly tests topological connectivity of linear spatial databases. In particular, they have proved Theorem 7.7.1. The generalization to databases that can be defined by at most quadratic polynomials, in particular Theorem 7.7.2, is due to Kuijpers and Smits [KS97b]. Geerts and Kuijpers have given a spatial DATALOG implementation that correctly tests connectivity of compact spatial databases in the plane [GK99a].

Chomicki and Imielinski [CI93] consider the language DATALOG_{1S} which is like DATALOG extended with an increment operator which may occur only in the first argument of relations. In DATALOG_{1S}, one cannot express the integer order relation which can be expressed in $\text{DATALOG}^{\text{gap}}$. Chomicki and Imielinski [CI93] show that the least fixpoint is evaluable for DATALOG_{1S} queries. DATALOG with modulus constraints were considered by Toman et al. in [TCR94]. DATALOG with a restricted case of vector additions is considered in [FO97].

DATALOG queries with Boolean equality constrains were studied in [BS87] and in [KKR95]. Theorem 7.8.2 was proved in [KKR95]. Lemma 7.8.1 is known as Boole's Lemma and originates from George Boole himself. The adder circuit example is from [KKR95].

The interested reader can find more references on constraint logic programming in the survey [JM94] and on DATALOG and constraint databases in the survey [Rev98a].

References

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (POPL'87)*, pages 111–119. ACM Press, 1987.

[KKR90] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proceedings of the 9th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems (PODS'90)*, pages 299–313. ACM Press, 1990.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51 (1): 26–52, 1995.

[GS95a] S. Grumbach and J. Su. Dense-order constraint databases. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'95)*, pages 66–77. ACM Press, 1995.

[Rev93] P. Z. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science (TCS)*, 116 (1/2): 117–149, 1993.

[CM93] J. Cox and K. McAloon. Decision procedures for constraint based extensions of Datalog. In *Constraint Logic Programming*. MIT Press, 1993.

[Rev95b] P. Z. Revesz. Safe stratified Datalog with integer order programs. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP'95)*, volume 976 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, 1995.

[Rev98d] P. Z. Revesz. Safe query languages for constraint databases. *ACM Transactions on Databases Systems (TODS)*, 23 (1): 58–99, 1998.

[ST95] A. P. Stolboushkin and M. A. Taitslin. Finite queries do not have effective syntax. In *Proceedings of the 14th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems (PODS'95)*, pages 277–285. ACM Press, 1995.

[ST98] A. P. Stolboushkin and M. A. Taitslin. Safe stratified Datalog with integer order does not have syntax. *ACM Transactions on Databases Systems (TODS)*, 23 (1): 100–109, 1998.

[Rev95b] P. Z. Revesz. Safe stratified Datalog with integer order programs. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP'95)*, volume 976 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, 1995.

[Rev98d] P. Z. Revesz. Safe query languages for constraint databases. *ACM Transactions on Databases Systems (TODS)*, 23 (1): 58–99, 1998.

[Rev99] P. Z. Revesz. Datalog programs with difference constraints. In *Proceedings of the Twelfth International Conference on Applications of Prolog*, pages 69–76, 1999.

[Rev98c] P. Z. Revesz. Safe Datalog queries with linear constraints. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of *Lecture Notes in Computer Science*, pages 355–369. Springer-Verlag, 1998.

[KPSV96] B. Kuijpers, J. Paredaens, M. Smits, and J. Van den Bussche. Termination properties of spatial Datalog programs. In *International Workshop on Logic in Databases (LID'96)*, volume 1154 of *Lecture Notes in Computer Science*, pages 101–116. Springer-Verlag, 1996.

[KS97b] B. Kuijpers and M. Smits. On expressing topological connectivity in spatial Datalog. In *Proceedings of the 2nd Workshop on Constraint Databases and Applications (CDB'97)*, volume 1191 of *Lecture Notes in Computer Science*, pages 116–133. Springer-Verlag, 1997.

[GK99a] F. Geerts and B. Kuijpers. Expressing topological connectivity of spatial databases. In *Proceedings of the 7th International Workshop on Database Programming Languages (DBPL'99)*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[CI93] J. Chomicki and T. Imielinski. Finite representation of infinite query answers. *ACM Transactions on Databases Systems (TODS)*, 18 (2): 181–223, 1993.

[TCR94] D. Toman, J. Chomicki, and D. S. Rogers. Datalog with integer periodicity constraints. In *Proceedings of the 11th International Symposium on Logic Programming (ILPS'g4)*, pages 189–203. MIT Press, 1994.

[FO97] L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints*, 2 (3/4): 305–335, 1997.

[BS87] W. Bittner and H. Simonis. Embedding Boolean expressions into logic programming *Journal of Symbolic Computation (JSC)*, 4 (2): 191–205, 1987.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51 (1): 26–52, 1995.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19 /20: 503–581, 1994.

[Rev98a] P. Z. Revesz. Constraint databases: A survey. In L. Libkin and B. Thalheim, editors, Semantics in Databases, volume 1358 of Lecture Notes in Computer Science. Springer-Verlag, 1998.