Theoretical Computer Science 239 (2000) 189–210

Theoretical Computer Science

www.elsevier.com/locate/tcs

Information integration using logical views ☆

Jeffrey D. Ullman

Department of Computer Science, Stanford University, Margaret Jacks Hall, Stanford, CA 94305, USA

Abstract

A number of ideas concerning information-integration tools can be thought of as constructing answers to queries using views that represent the capabilities of information sources. We review the formal basis of these techniques, which are closely related to containment algorithms for conjunctive queries and/or Datalog programs. Then we compare the approaches taken by AT&T Labs' "Information Manifold" and the Stanford "Tsimmis" project in these terms. © 2000 Elsevier Science B.V. All rights reserved.

1. Theoretical background

Before addressing information-integration issues, let us review some of the basic ideas concerning conjunctive queries, Datalog programs, and their containment. To begin, we use the logical rule notation from [25].

Example 1.1. The following

```
p(X,Z) :- a(X,Y)& a(Y,Z)
```

is a rule that talks about a, an *EDB predicate* ("Extensional DataBase", or stored relation), and p, an *IDB predicate* ("Intensional DataBase", or predicate whose relation is constructed by rules). In this and several other examples, it is useful to think of a as an "arc" predicate defining a graph, while other predicates define certain structures that might exist in the graph. That is, $a(X,Y)$ means there is an arc from node X to node Y. In this case, the rule says "$p(X,Z)$ is true if there is an arc from node X to some node Y and also an arc from Y to Z". That is, p represents paths of length 2.

In general, there is one atom, the *head*, on the left of the "if" sign, `:-`, and zero of more atoms, called *subgoals*, on the right side (the *body*). The head always has an

☆ This work was supported by NSF grant IRI-96-31952, ARO grant DAAH04-95-1-0192, and Air Force contract F33615–93–1–1339.

E-mail address: ullman@stanford.edu (J.D. Ullman)

0304-3975/00/\$ - see front matter © 2000 Elsevier Science B.V. All rights reserved.
PII: S0304-3975(99)00219-4

IDB predicate; the subgoals can have IDB or EDB predicates. Thus, here $p(X,Z)$ is the head, while $a(X,Y)$ and $a(Y,Z)$ are subgoals.

We assume that each variable appearing in the head also appears somewhere in the body. This "safety" requirement assures that when we use a rule, we are not left with undefined variables in the head when we try to infer a fact about the head's predicate.

We also assume that atoms consist of a predicate and zero or more arguments. An argument can be either a variable or a constant. However, we exclude function symbols from arguments.

1.1. Conjunctive queries

A *conjunctive query* (CQ) is a rule with subgoals that are assumed to have EDB predicates. A CQ is *applied* to the EDB relations by considering all possible substitutions of values for the variables in the body. If a substitution makes all the subgoals true, then the same substitution, applied to the head, is an inferred fact about the head's predicate.

Example 1.2. Consider Example 1.1, whose rule is a CQ. If $a(X,Y)$ is true exactly when there is an arc $X \rightarrow Y$ in a graph G, then a substitution for X, Y, and Z will make both subgoals true when there are arcs $X \rightarrow Y \rightarrow Z$. Thus, $p(X,Z)$ will be inferred exactly when there is a path of length 2 from X to Z in G.

A crucial question about CQs is whether one is *contained* in another. If Q_1 and Q_2 are CQs, we say $Q_1 \subseteq Q_2$ if for all databases (truth assignments to the EDB predicates) D, the result of applying Q_1 to D [written $Q_1(D)$] is a subset of $Q_2(D)$. Two CQs are *equivalent* if and only if each is contained in the other. It turns out that in almost all cases, the only approach known for testing equivalence is by testing containment in both directions. Moreover, in information-integration applications, containment appears to be more fundamental than equivalence, so from here we shall concentrate on the containment test.

Conjunctive queries and their containment were first studied by Chandra and Merlin [3]. Here, we shall give another test, following the approach of [21], because this test extends more naturally to the generalizations of the CQ-containment problem that we shall discuss. To test whether $Q_1 \subseteq Q_2$:
1. *Freeze* the body of Q_1 by turning each of its subgoals into facts in the database. That is, replace each variable in the body by a distinct constant, and treat the resulting subgoals as the only tuples in the database.
2. Apply Q_2 to this *canonical* database.
3. If the frozen head of Q_1 is derived by Q_2, then $Q_1 \subseteq Q_2$. Otherwise, not; in fact, the canonical database is a counterexample to the containment, since surely Q_1 derives its own frozen head from this database.

Example 1.3. Consider the following two CQs:

Q_1: `p(X,Z) :- a(X,Y) & a(Y,Z),`

Q_2: `p(X,Z) :- a(X,U) & a(V,Z).`

Informally, Q_1 looks for paths of length 2, while Q_2 looks only for nodes X and Z such that X has an arc out to somewhere, and Z has an arc in from somewhere. Intuitively, we expect, $Q_1 \subseteq Q_2$, and that is indeed the case.

In this and other examples, we shall use integers starting at 0 as the constants that "freeze" the CQ, although obviously the choice of constants is irrelevant. Thus, the canonical database D constructed from Q_1 consists of the two tuples $a(0,1)$ and $a(1,2)$ and nothing else. The frozen head of Q_1 is $p(0,2)$.

If we apply Q_2 to D, the substitution $X \to 0, U \to 1, V \to 1$, and $Z \to 2$ yields $p(0,2)$ in the head of Q_2. Since this fact is the frozen head of Q_1, we have verified $Q_1 \subseteq Q_2$.

Incidentally, for this containment test and the more general tests of following subsections, the argument that it works is, in brief:

- If the test is negative, then the constructed database is a counterexample to the containment.
- If the test is positive, then there is an implied homomorphism μ from the variables of Q_2 to the variables of Q_1. We obtain μ by seeing what constant each variable X of Q_2 was mapped to in the successful application of Q_2 to the canonical database. $\mu(X)$ is the variable of Q_1 that corresponds to this constant. If we now apply Q_1 to any database D and yield a particular fact for the head, let the homomorphism from the variables of Q_1 to the database symbols that we use in this application be v. Then μ followed by v is a homomorphism from the variables of Q_2 to the database symbols that shows how Q_2 will yield the same head fact. This argument proves $Q_1 \subseteq Q_2$.

Containment of CQs is NP-complete [3], although [22] shows that in the common case where no predicate appears more than twice in the body, then there is a linear-time algorithm for containment.

1.2. CQs with negation

An important extension of CQs is to allow negated subgoals in the body. The effect of applying a CQ to a database is as before, but now, when we make a substitution of constants for variables, the atoms in the negated subgoals must be false rather than true (i.e., each negated subgoal itself must be true).

Now, the containment test is in a slightly higher complexity class; it is complete for the class Π_2^p, problems that can be expressed as $\{w \mid (\forall x)(\exists y)\phi(w,x,y)\}$, where strings x and y are of length bounded by a polynomial function of the length of w, and ϕ is a function that can be computed in polynomial time. This test, due to Levy and Sagiv [16], involves exploring an exponential number of "canonical" databases, any one of which can provide a counterexample to the containment.

The Levy–Sagiv test is straightforward. Suppose we wish to test $Q_1 \subseteq Q_2$. We use an alphabet A of k symbols, where k is the number of variables in Q_1. We consider all databases D formed from tuples all of whose components are in A. If $Q_1(D) \subseteq Q_2(D)$ for each of these *canonical* databases, then $Q_1 \subseteq Q_2$, and if not, then not. The number of canonical databases is 2 raised to a polynomial in k; the exact polynomial depends on the number of arguments in the subgoals of Q_1 and the number of different predicates in Q_1.

A more systematic, although still exponential way to consider all the canonical databases is:

1. Consider all partitions of the variables of Q_1 and assign for each block of the partition a unique constant. Thus, we obtain a number of *basic canonical databases* $D_1, D_2, \ldots, D_k$, where k is the number of partitions of integer n, and n is the number of variables in the body of Q_1. Each D_i consists of the frozen positive subgoals of Q_1 only, not the negated subgoals.
2. For each basic canonical database D_i consider whether D_i makes all the subgoals of Q_1 true. Note that because the atom in a negated subgoal may happen to be in D_i, it is possible that D_i makes the body of Q_1 false.
3. For those D_i that make the body of Q_1 true, test whether any $Q_2(D)$ includes the frozen head of Q_1, where D is any database that:
   (a) is a superset of D_i formed by adding other tuples that use the same set of symbols as D_i, and
   (b) does not include any tuple that is a frozen negative subgoal of Q_1.

   These Ds, plus the basic canonical databases, form the set of *canonical databases* for the containment question $Q_1 \subseteq Q_2$. When determining what the frozen head of Q_1 is, we make the same substitution of constants for variables that yielded D_i.
4. If every D_i either makes the body of Q_1 false [the test of (2)] or meets the test of (3), then $Q_1 \subseteq Q_2$; otherwise, not.

Example 1.4. Let us consider the following two conjunctive queries:

Q_1: `p(X,Z) :- a(X,Y) & a(Y,Z) & NOT a(X,Z),`

Q_2: `p(A,C) :- a(A,B) & a(B,C) & NOT a(A,D).`

Intuitively, Q_1 looks for paths of length 2 that are not "short-circuited" by a single arc from beginning to end. Q_2 looks for paths of length 2 that start from a node A that is not a "universal source", i.e., there is at least one node D not reachable from A by an arc.

To show $Q_1 \subseteq Q_2$ we need to consider all partitions of $\{X, Y, Z\}$. There are five of them: one that keeps all three variables separate, one that groups them all, and three that group one pair of variables. The table in Fig. 1 shows the five cases and their outcomes.

For instance, in case (1), where all three variables are distinct, and we have arbitrarily chosen the constants 0, 1, and 2 for X, Y, and Z, respectively, the basic canonical

	Partition	Canonical database	Outcome
(1)	$\{X\}\{Y\}\{Z\}$	$\{a(0,1), a(1,2)\}$	both yield head $p(0,2)$
(2)	$\{X,Y\}\{Z\}$	$\{a(0,0), a(0,1)\}$	Q_1 body false
(3)	$\{X\}\{Y,Z\}$	$\{a(0,1), a(1,1)\}$	Q_1 body false
(4)	$\{X,Z\}\{Y\}$	$\{a(0,1), a(1,0)\}$	both yield head $p(0,0)$
(5)	$\{X,Y,Z\}$	$\{a(0,0)\}$	Q_1 body false

Fig. 1. The five basic canonical databases and their outcomes.

database D_1 is the two positive subgoals, frozen to be $a(0,1)$ and $a(1,2)$. The frozen negative subgoal NOT $a(0,2)$ is true in this case, since $a(0,2)$ is not in D_1. Thus, Q_1 yields its own head, $p(0,2)$, and we must test that Q_2 does likewise on any database D consisting of symbols 0, 1, and 2, that includes the two tuples of D_1 and does not include the tuple $a(0,2)$, the frozen negative subgoal of Q_1. Since there are six optional tuples, there are $2^6 = 64$ different canonical databases D.

However, regardless of which of these 64 databases D is, the same argument will show that $Q_2(D)$ includes the frozen head of Q_1, which is $p(0,2)$. Specifically, we use the substitution $A \to 0$, $B \to 1$, $C \to 2$, and $D \to 2$. Then the positive subgoals become true for any such D. The negative subgoal becomes NOT $a(0,2)$, as we have explicitly excluded $a(0,2)$ from any of these databases D. We conclude that the Levy–Sagiv test holds for case (1).

Now consider case (2), where X and Y are equated and Z is different. We have chosen to use 0 for X and Y; 1 for Z. Then the basic canonical database for this case is D_2, consisting of the frozen positive subgoals $a(0,0)$ and $a(0,1)$. For this substitution, the negative subgoal of Q_1 becomes NOT $a(0,1)$. Since $a(0,1)$ is in D_2, this subgoal is false. Thus, for this substitution of constants for variables in Q_1, we do not even derive the head of Q_1. We do not need to check further in this case; the test is satisfied.

The three remaining cases must be checked as well. However, as indicated in Fig. 1, in each case either both CQs yield the frozen head of Q_1 or Q_1 does not yield its own frozen head. Thus, the test is completely satisfied, and we conclude $Q_1 \subseteq Q_2$.

1.3. CQs with arithmetic comparisons

Another important extension of CQ-containment theory is the inclusion of arithmetic comparisons as subgoals. In this regard we must consider the set of values in the database as belonging to a totally ordered set, e.g., the integers or reals. When we consider possible assignments of integer constants to the variables of conjunctive query Q_1, we may use consecutive integers, starting at 0, but now we must consider not only partitions of variables into sets of equal value, but among the blocks of the partition, we must consider the relative order of their values. The basic canonical database is constructed from those subgoals that have nonnegated, uninterpreted predicates only, not those with a negation or a comparison operator.

If there are negated subgoals, then we must also consider certain canonical databases that are supersets of the basic canonical databases, as we did in Section 1.2. But if there are no negated subgoals, then the basic canonical databases alone suffice.

Example 1.5. Now consider the following two conjunctive queries, each of which refers to a graph in which nodes are assumed to be integers:

Q_1: `p(X,Z) :- a(X,Y) & a(Y,Z) & X<Y,`

Q_2: `p(A,C) :- a(A,B) & a(B,C) & A<C.`

Both ask for paths of length 2. But Q_1 requires that the first node be numerically less than the second, while Q_2 requires that the first node be numerically less than the third.

The number of different basic canonical databases is 13. We must consider the five different partitions of $\{X, Y, Z\}$, as we did in Fig. 1. However, we also have to order the blocks of each partition. For partition (1) of Fig. 1, where each variable is separate, we have six possible orders of the blocks. For partitions (2)–(4), where there are only two blocks, we have two different orders. Finally, for partition (5), with only one block, there is one order.

In this example, the containment test fails. We have only to find one of the 13 cases to show failure. For instance, consider $X = Z = 0$ and $Y = 1$. The basic canonical database D for this case is $\{a(0,1), a(1,0)\}$, and since $X < Y$, the body of Q_1 is true. Thus, $Q_2(D)$ must include the frozen head of Q_1, $p(0,0)$. However, no assignment of values to A, B, and C makes all three subgoals of Q_2 true, when D is the database. That is, in order to make subgoals $a(A,B)$ and $a(B,C)$ both true for D, we surely must use 0 or 1 for all of A, B, and C. Then to make $A < C$ true, we must have $A = 0$ and $C = 1$. But then, whether B is 0 or 1 we shall have in Q_2 a subgoal $a(0,0)$ or $a(1,1)$, neither of which is in D. Thus, D is a counterexample to $Q_1 \subseteq Q_2$.

The containment test for CQs with arithmetic from [10, 28] shows that the problem of testing containment for CQs with arithmetic comparisons is complete for Π_2^p, at least in the case of a dense domain such as the reals. [16] actually includes arithmetic comparisons in their work on negation, and we should note that the above technique works even if there are negated subgoals as well as arithmetic comparisons. There is a more general approach that works for any interpreted predicates, not just a predicate like $<$ or $\leqslant$ that forms a total order; it appears in [32]. However, this technique does not include CQs with negated subgoals.

1.4. Datalog programs

Let us now return to the original model of rules, excluding negated subgoals and arithmetic comparisons. However, we shall now consider collections of rules, which we call a *Datalog program*. Such collections of rules have a natural, least-fixedpoint interpretation, where we start by assuming the IDB predicates have empty relations. We then use the rules to infer new IDB facts, until no more facts can be inferred.

More on the semantics of Datalog, including efficient algorithms for evaluating the IDB predicates, can be found in [25, 26]. While we shall not discuss Datalog with negated subgoals here, because the meaning is debatable in some cases, the principal ideas are surveyed in [27]. Here is an example of a Datalog program and its semantics.

Example 1.6. Consider the three rules:

```
(1) p(X,Z) :- q(X,Y) & b(Y,Z).
(2) q(X,Y) :- a(X,Y).
(3) q(X,Z) :- a(X,Y) & p(Y,Z).
```

Intuitively, think of a graph with two kinds of arcs: "a-arcs" and "b-arcs". Then p and q represent certain kinds of paths. Rule (1) says that a q-path followed by a b-arc is a p-path. Rule (2) says that a single a-arc is a q-path, while rule (3) says that a-arcs followed by p-paths are also q-paths. It may not be obvious what is going on, but one can prove by an easy induction that the p-paths consist of some number $n \geqslant 1$ of a-arcs followed by an equal number of b-arcs. A q-path is the same, except it has one fewer b-arc.

To get a feel for why this description of p- and q-paths is valid, consider a particular graph G described by the a and b EDB predicates. Then rule (2) says all the paths a are in the relation for q. We can therefore use rule (1) to infer that any path of the form ab is in the relation for p; more precisely, if there is a path from node X to node Z that follows an a-arc and then a b-arc, $p(X,Z)$ is true. Next, rule (3) tells us that any path of the form aab is a q-path; rule (1) says paths of the form $aabb$ are p-paths, and so on.

Containment questions involving Datalog programs are often harder than for CQs. [23] shows that containment of Datalog programs is undecidable, while [5] shows that containment of a Datalog program in a CQ is doubly exponential. However, the important case for purposes of information integration is the containment of a CQ in a Datalog program, and this question turns out to be no more complex than containment of CQs [21].

To test whether CQ Q is contained in Datalog program P, we "freeze" the body of Q, just as we did in Section 1.1, to make a canonical database D. We then see if $P(D)$ contains the frozen head of Q. The only significant difference between containment in a CQ and containment in a Datalog program is that in the latter case we must keep applying the rules until either the head is derived, on no more IDB facts can be inferred.

Example 1.7. Consider the Datalog program from Example 1.6, which we shall call P, and the CQ Q:

```
p(A,C) :- a(A,B) & b(B,C).
```

Freezing the body of Q, we obtain the canonical database $D = \{a(0,1), b(1,2)\}$. Now, we apply P to D. Rule (2) lets us infer $q(0,1)$ from $a(0,1)$. Then, rule (1) lets us

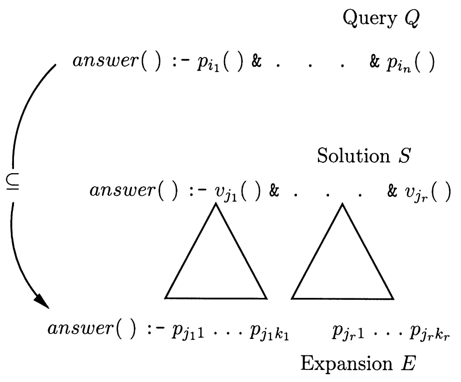
Query Q
$answer(\ ) \text{ :- } p_{i_1}(\ )\ \&\ .\ .\ .\ \&\ p_{i_n}(\ )$
Solution S
$\subseteq$
$answer(\ ) \text{ :- } v_{j_1}(\ )\ \&\ .\ .\ .\ \&\ v_{j_r}(\ )$
$answer(\ ) \text{ :- } p_{j_1 1} \cdots p_{j_1 k_1} \quad p_{j_r 1} \cdots p_{j_r k_r}$
Expansion E

Fig. 2. Constructing a query from views.

infer $p(0,2)$ from $q(0,1)$ and $b(1,2)$. Since $p(0,2)$ is the frozen head of Q, our test has concluded positively; $Q \subseteq P$.

2. Synthesizing queries from views

Query containment algorithms connect to information-integration via a concept called "synthesizing queries from views". The idea, originally studied by [4, 31], is suggested in Fig. 2. There are a number of "EDB" predicates, for which we use p's in Fig. 2. These predicates, which are not truly EDB predicates since they usually do not exist as physically stored relations, can be thought of as representing the basic concepts used in queries. There are also *views*, denoted by v's in Fig. 2, that represent resources that the integrator uses internally to help answer queries. Each view has a definition in terms of the EDB predicates, and we suppose here that these definitions are conjunctive queries.

2.1. Solving queries by views

A query Q is expressed in terms of the EDB predicates, the p's. Our problem is to find a "solution" S for the query Q. A *solution* is an expression (also a CQ in the figure) in terms of the views. When we replace the views in S by their definitions, we get an *expansion* query E, which must be equivalent to the original query Q, if the solution S is valid. An alternative formulation of the query-synthesis problem is to ask for all solutions S whose expansion E is contained in Q (perhaps properly contained). "The solution" for Q is then the union of all these partial solutions.

Example 2.1. We shall consider an example that illustrates some technical points, but suffers in realism for the sake of these points. Let us suppose that there is a single

EDB predicate $p(X,Y)$ which we interpret to mean that Y is a parent of X. Let there be two views, defined as follows:

```
v1(Y,Z) :- p(X,Y) & p(Y,Z),
v2(X,Z) :- p(X,Y) & p(Y,Z).
```

Note that the views have the same body but different heads. The first view, v_1, actually produces a subset of the relation for p: those child-parent pairs (Y,Z) such that the child is also a parent of some individual X. The second view, v_2, produces a straightforward grandparent relation from the parent relation.

Suppose that we want to query this information system for the great grandparents of a particular individual, whom we denote by the constant 0. This query is expressed in terms of the EDB predicate p by

```
q(C) :- p(0,A) & p(A,B) & p(B,C).
```

Our problem is to find a CQ whose subgoals use only the predicates v_1 and v_2 and whose expansion is equivalent to the query above. A bit of thought tells us that

```
s1(C) :- v2(0,D) & v1(D,C)
```

is a solution. That is, if we replace each of the subgoals of s_1 by the definition of the views (being careful to use unique variables in place of those variables that appear in the bodies of the view definitions but not in the heads of those definitions), we get the expansion:

```
e1(C) :- p(0,E) & p(E,D) & p(F,D) & p(D,C).
```

We can use the CQ containment test in both directions to prove that $e_1 \equiv q$. Intuitively, the subgoal $p(F,D)$ in e_1 is superfluous, since every time there is binding for E and D that makes $p(E,D)$ true, we can bind F to the same value as E and make $p(F,D)$ true.

There are other solutions that, when expanded, are contained within q, but are not equivalent to it. Some examples are:

```
s2(C) :- v1(0,D) & v2(D,C),
s3(C) :- v1(0,D) & v1(D,E) & v1(E,C),
s4(C) :- v2(0,D) & v1(D,C) & v2(C,E).
```

Solution s_2 is equivalent to q if individual 0 has a child in the database. Otherwise, 0 cannot appear as a first component in the relation for v_1, and the result of s_2 is empty. Thus, $s_2 \subseteq q$, but not conversely. Solution s_3 is actually equivalent to s_2, while s_4 gives those great grandparents of individual 0 who are themselves grandchildren.

2.2. Minimal-solution theorems

It might appear from Example 2.1 that one can only guess potential solutions for a query and test them via CQ-containment tests. However, there are theorems that limit the search and show that the problem of expressing a query in terms of views, while NP-complete, is no worse than that. As discussed in Section 1.1, we expect that queries will be short, so NP-complete problems are unlikely to be a major bottleneck in practice.

The principal idea is that any view used in a solution must serve some function in the query; a view without a function may be deleted from the solution. For example, one possible function of a view is to cover some subgoal of the query must be covered by some view. The question of when a view covers a query subgoal is a bit subtle, because two or more views may cover the same subgoal. For instance, consider Example 2.1, where both $p(E,D)$ and $p(F,D)$ from expansion e_1 "cover" $p(A,B)$ from the query. More precisely, A, E, and F may each represent a parent of individual 0, while B and D represent a parent of that parent. Note that $p(E,D)$ and $p(F,D)$ come from the expansion of $v_2(0,D)$ and $v_1(D,C)$, respectively, in solution s_1, so these two subgoals from different views each play the same role in the expansion.

Let us define a solution S for a query Q to be *minimal* if
1. $S \subseteq Q$.
2. There is no solution T for Q such that
   (a) $S \subseteq T \subseteq Q$, and
   (b) T has fewer subgoals than S.

Theorem 2.2 (Levy [11]). *If queries are CQs without negation, arithmetic comparisons, or constants in the body, then every minimal, contained conjunctive-query solution for a query Q has no more subgoals* (*uses of views*) *than Q has subgoals.*

Theorem 2.3 (Rajaraman [20]). *If queries are CQs without negation or arithmetic comparisons* (*but with constants in the body permitted, as in Example* 2.1), *then every minimal equivalent CQ-solution for a query Q has no more subgoals than the sum of the number of subgoals and number of variables in Q.*

Both Theorems 2.2 and 2.3 offer nondeterministic polynomial-time algorithms to find either
1. a single solution equivalent to the query Q, or
2. a set of solutions whose union is contained in Q and that contains any other solution that is contained in Q.

In each case, one searches "only" an exponential number (as a function of the length of Q) of minimal queries. If we are looking for one solution equivalent to Q, then we may stop if we find one, and we conclude there is none if we have searched all solutions of length up to the bound and found none. If we want all solutions contained in the query, then we search all up to the bound, taking those that are contained in Q.

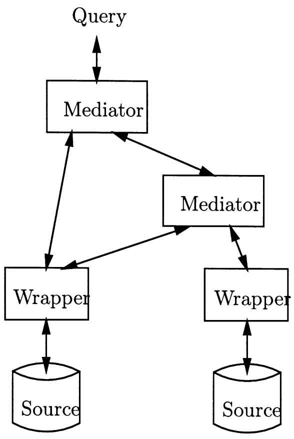

Fig. 3. Common integration architecture.

3. Information-integration systems

Information integration has long been recognized as a central problem of modern database systems. While early databases were self-contained, it is now generally realized that there is great value in taking information from various sources and making them work together as a whole. Yet there are several difficult problems to be faced:

- "Legacy" databases cannot be altered just because we wish to support a new, integrating application above them.
- Databases that ostensibly deal with the same concepts may have different shades of meaning for the same term, or use different terms for the same concept.
- Information sources, such as those on the "web", may have no fixed schema or a time-varying schema.

A common integration architecture is shown in Fig. 3. Several sources are *wrapped* by software that translates between the source's local language, model, and concepts and the global concepts shared by some or all of the sources. System components, here called *mediators* [30], obtain information from one or more components below them, which may be wrapped sources or other mediators. Mediators also provide information to components above them and to external users of the system.

In a sense, a mediator is a view of the data found in one or more sources. Data does not exist at the mediator, but one may query the mediator as if it were stored data; it is the job of the mediator to go to its sources and find the answer to the query.

Today, the components labeled "mediator" in Fig. 3 are unlikely to be true mediators, but rather *data warehouses*. If a mediator is like a view, then a warehouse is like a materialized view. That is, the warehouse holds data that is constructed from the data at the sources. The warehouse is queried directly, with no involvement by the sources. There are numerous problems associated with the design and implementation of warehouses (see e.g. [29]), not the least of which is that it is difficult and/or expensive to keep the warehouse up-to-date, as the underlying data changes.

There are, however, several research projects developing true mediator capabilities, and in this section we shall introduce two of them:

1. *Information Manifold* [9, 12, 13], a project of AT&T Laboratories.
2. *Tsimmis* [7, 17–19, 24], a project at Stanford University.

Both systems use logic-based technology, and while neither is based on Datalog per se, the operation of each can be translated into Datalog.

3.1. Information manifold

Information manifold (*IM*) is based on a dialect of *description logic* called CARIN [15]. Description logic is a fragment of first-order logic that can almost be thought of as nonrecursive Datalog with IDB predicates restricted to be unary, although there are certain capabilities of description logic that are beyond what Datalog provides [2]. Here, we shall use Datalog in examples of the architecture of IM.

The architecture of IM is essentially that described in Section 2. The following points characterize IM in these terms:

- An IM application has a collection of "global" predicates, in terms of which all queries are expressed.
- Each information source is associated with one or more views. Views are also defined in terms of the global predicates.
- However, the definition of a view should not be given the usual interpretation of "this source provides *all* facts derivable from its definition and the global predicates". Rather, the intension is that the view provides *some* of those facts.
- The solution to a query is the union of all minimal CQs (over the views) contained in the query. Note that there could be other solutions to the query in sources not available to this IM application, but the minimal solutions provide all the query answers that are accessible to IM.
- Also associated with a source are zero or more *constraints*. A constraint is a guarantee that certain facts that might be present in the view will in truth *not* appear there. For example, a source might supply a parent-child predicate as its view, and a constraint might state that the only pairs supplied will have female children born after 1970.

Example 3.1. Let us consider an integrated information system about employees of a company. This example too is somewhat contrived for the sake of some technical points. In this system, the global predicates are:

1. *emp*(E), meaning E is an employee.
2. *phone*(E,P), meaning P is E's phone.
3. *office*$((E,O)$, meaning O is E's office.
4. *mgr*(E,M), meaning M is E's manager.
5. *dept*(E,D), meaning D is E's department.

There are three sources, each of which provides one view. The definitions of the views are:

```
v1(E,P,M) :- emp(E) & phone(E,P) & mgr(E,M),
v2(E,O,D) :- emp(E) & office(E,O) & dept(E,D),
v3(E,P)   :- emp(E) & phone(E,P) & dept(E,toy).
```

That is, the first source, which supports view v_1, gives information about employees, their phones and managers. The second source supports view v_2 and gives information about the offices and departments of employees. The third source supports view v_3 and provides the phones of employees, but only for employees in the Toy Department. Notice that the constraint department = "Toy" is enforced by the subgoal $dept(E, toy)$ in the definition of v_3. This constraint would be important if we asked a query about employees known not to be in the Toy Department; then we would know that v_3 does not appear in any minimal solution.

Also note that there is no reason to believe the phone information provided by v_1 and v_3 is consistent. Further, it is entirely possible that the information is *incomplete*; only one of these sources provides phone information, even though the employee is in the Toy Department. In fact, perhaps neither source tells us Sally's phone, even though she has a phone.

Suppose this system is asked a query: "what are Sally's phone and office?" We can express this query in terms of the global predicates as

```
q1(P,O) :- phone(sally,P) & office(sally,O).
```

There are two minimal solutions to this query. Both use v_2 to get Sally's office, while the two solutions differ on whether v_1 or v_3 is used to get the phone. That is, the full answer to query q_1 is the union of the CQs:

```
answer(P,O) :- v1(sally,P,M) & v2(sally,O,D),
answer(P,O) :- v3(sally,P) & v2(sally,O,D).
```

Note that the expansions of these solutions:

```
answer(P,O) :- emp(sally) & phone(sally,P) & mgr(sally,M) &
               emp(sally) & office(sally,O) & dept(sally,D),
answer(P,O) :- emp(sally) & phone(sally,P) & dept(sally,toy) &
               emp(sally) & office(sally,O) & dept(sally,D)
```

are not equivalent to q_1; they are the CQs that come closest to q_1 while still being contained in q_1.

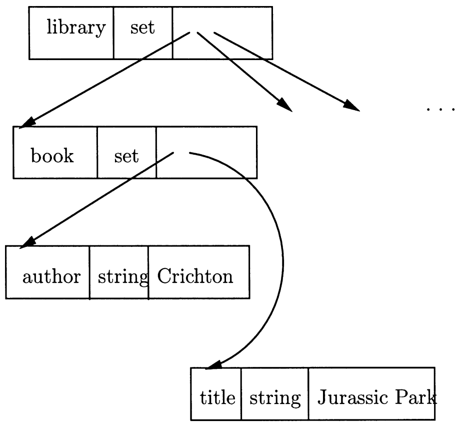

Fig. 4. An OEM object.

3.2. Tsimmis

Tsimmis, which stands for “The Stanford-IBM Manager of Multiple Information Sources”, was a DARPA-funded, joint project of the Stanford database group and the IBM/Almaden database research group, although the IBM contingent later began work on their own information integration project called the *Garlic* [8]. Tsimmis follows the mediator architecture of Fig. 3, allowing us to create a hierarchy of wrappers and mediators that talk to one another. Tsimmis components talk among themselves using a data model called the *object-exchange model* (OEM) and a query language called *mediator specification language* (MSL). MSL is also used to describe mediators and wrappers at a high level, and these components can be generated automatically from the MSL specification.

3.2.1. OEM

The OEM model [18] is “object-oriented”, and data is assumed to be organized into objects. An *OEM object* consists of:

1. A *label*, roughly the name of the object’s class.
2. A *type* for the value of the object. The type is either an atomic type: integer, string, Java script, and so on, or it is the type “set of OEM objects”.
3. A *value*, either an actual value if the object is atomic, or a set of OEM objects.
4. An (optional) object-ID.

Example 3.2. Fig. 4 suggests an OEM object with label `library`, whose value is a set of objects representing the documents in the library. We also see one member object, with label `book`. The value of this object is a set, and we have shown two members of that set. Both are atomic objects, one labeled `title` and having value `Jurassic Park`, and the other labeled `author` with value `Crichton`.

```
(1) <f(E) epo {<name E> <phone P>}> @med :-
              <emp {<name E> <phone P>} > @source1
(2) <f(E) epo {<name E> <phone P>}> @med :-
              <emp {<name E> <phone P>}> @source3
(3) <f(E) epo {<name E> <office O>}> @med :-
              <emp {<name E> <office O>}> @source2
(4) <edm {<name E> <dept D> <mgr M>}> @med :-
              <emp {<name E> <mgr M>}> @source1 AND
              <emp {<name E> <dept D>}> @source2
```

Fig. 5. An MSL mediator-description.

3.2.2. MSL

MSL statements are logical rules, but the rules are not exactly Datalog. Rather, MSL uses a form of object-logic, in which

- Labels and values are connected using triangular brackets, `<...>`.
- It is also possible to include an object-ID inside triangular brackets as an optional first component.
- Object-IDs may be constructed using function symbols, as in HiLog [6]
- Some (not necessarily all) members of a set of objects may be described by enclosing them in curly braces `{...}`.

Example 3.3. Let us reconsider Example 3.1, where we had three sources. Source 1 produces employee–phone–manager information, Source 2 produces employee–office–department information, and Source 3 produces employee–phone information for members of the Toy Department. Each of these sources will be assumed to export appropriate OEM objects. For example, Source 1 exports objects with atomic subobjects labeled `name`, `phone`, and `mgr`. We wish to describe, using MSL rules, a mediator named `med` that uses these three sources and exports two types of objects:

- Employee–phone–office objects with label `epo`.
- Employee–department–manager objects with label `edm`.

Each object of these types will have subobjects with the appropriate labels. Fig. 5 shows the MSL rules that describe these objects exported by `med`.

In this example, we have made the (unrealistic) assumption that employee names are unique. Thus, as we assemble `epo` objects for an employee named E, we use the object-ID $f(E)$, expecting that this ID is unique. Rule (1) says that whenever there is an `emp` object at Source 1 with a `name` subobject having value E and a `phone` subobject with value P, we "create" at the mediator `med` an object whose ID is $f(E)$ and whose label is `epo`. This object has a subobject with label `name` and value E and a second subobject with label `phone` and value P. Rules (2) and (3) are similar; rule (2) takes employee/phone information from Source 3, while rule (3) takes employee/office information from Source 2. Three important points are:

- Because the object-ID is specified in rules (1)–(3), whenever information about the same employee E is found in two or more sources, the subobjects implied by the heads of these rules will be combined into the value of the same object – the one with ID $f(E)$. Thus, it will be typical that employee objects will have three subobjects, with labels `name`, `phone`, and `office`. They could even have more than three subobjects. For example, Sources 1 and 3 could give different phones, so two subobjects labeled `phone` would appear. A single source could also have several phones or offices for employee E, and all of these would appear as subobjects at the mediator.
- The fact that rule (1) only mentions `name` and `phone` subobjects at Source 1 does not mean it will fail if there are more subobjects, e.g., a `manager` subobject. MSL only mentions subobjects it needs, allowing any other subobjects to be present. There is even a way (the *rest-variable*) to refer to “whatever other subobjects are present”.
- There is no assumption that variables like E or P are atomic. They might turn out to have sets of objects as values, and in fact different objects at the sources may have different types for values having the same label. For instance, some employees may have strings for names, while others have objects with first- and last-name subobjects.

Rule (4) in Fig. 5 follows a somewhat different philosophy in constructing the `edm` objects at `med`. Here, an object is produced only if we are successful in finding, for employee E, a department at Source 2 and a manager at Source 1. If either is missing, then there is no object for employee E at `med`. In contrast, rules (1)–(3) allow there to be an `epo` object for E if any one of the three sources mentions E. Note also that the object-ID component in the constructed sources is optional, and in rule (4) there is no need to specify an ID. Thus, the head of rule (4) has only label and value components, while the other rules have 3-component heads.

3.2.3. Converting MSL to Datalog

There is a way to convert MSL into completely equivalent Datalog [17]. We shall not go into this process, but rather give a simplification that will help us compare IM and Tsimmis.

Example 3.4. The following rules capture much of the content of the MSL rules in Fig. 5:

```
epo(E,P,O) :- v1(E,P,M) & v2(E,O,D)

epo(E,P,O) :- v3(E,P) & v2(E,O,D)

edm(E,D,M) :- v1(E,P,M) & v2(E,O,D)
```

Recall that v_1, v_2, and v_3 are the three views that we introduced in Example 3.1. They correspond to Sources 1–3 in Example 3.3.

There is one important way that the rules above differ from the MSL rules in Fig. 5. We only get *epo* facts for employees such that among the three views we find both a phone and office for that employee. In contrast, as we mentioned in Example 3.3, the MSL rules can yield a phone without an office or vice versa. This capability of MSL is an essential contribution to dealing with heterogeneous, often incomplete information sources.

3.2.4. Querying Tsimmis mediators

When we query an MSL mediator, we are effectively querying the objects exported by the mediator. There is no notion of "global" predicates as there is in IM. Rather, we must refer to the labels (equivalent to predicates) that the mediator exports. Completion of our running example will illustrate the distinction between the Tsimmis and IM approaches.

Example 3.5. Again let us ask "what are Sally's phone and office"? This time, however, we ask it of the mediator `med`, whose exported objects we have represented in Datalog by the rules of Example 3.4. The appropriate query is thus:

```
answer(P,O) :- epo(sally,P.O).
```

MSL-generated mediators answer their queries by expanding the rules by which the mediator is defined, in order to get the same query in terms of information at the sources. In our simple example, we would replace the *epo* subgoal in the query by the bodies of the two rules that define *epo* at `med`, thus obtaining:

```
answer(P,O) :- v1(sally,P,M) & v2(sally,O,D),
answer(P,O) :- v3(sally,P) & v2(sally,O,D).
```

Notice that this expansion is identical to what IM obtained for the same query.

3.3. Comparing the IM and Tsimmis query processors

We should not suppose from Example 3.5 that the result of "equivalent" IM and Tsimmis queries are always the same, even after accounting for the difference in the underlying logics. The processes of query translation are rather different.

- IM uses the query synthesis strategy outlined in Section 2.
- IM queries are in terms of global predicates, which are translated into views.
- Tsimmis queries are in terms of predicates synthesized at a mediator. These concepts, in turn, are built from views in the IM sense, exported by the sources.
- Tsimmis uses a strategy of rule expansion to answer queries. Although the expansion can result in an exponential number of terms, the flavor of the search is different from IMs. In Tsimmis we can expand each subgoal of the query independently, using every rule whose head unifies with the subgoal.

Example 3.6. The following is an example of how the two systems can differ. In this example, Tsimmis appears to flounder, but we should emphasize that it is an atypical example, contrived for the sake of illustration.

Suppose we wanted to know Sally's office and department. That is

```
q2(O,D):- office(sally,O) & dept(sally,D).
```

Using the views of Example 3.1, IM would find that the only minimal solution to the query q_2 is

```
answer(O,D):- v2(sally,O,D).
```

However, using the Tsimmis mediator `med` of Example 3.3, we can only express our query as

```
q3(O,D):- epo(sally,P,O) & edm(sally,D,M).
```

The reason for this awkwardness is that each mediator exports a specific collection of objects. We do not have the freedom to penetrate, in our query, to the terms used by the mediator's sources.

The mediator `med` would process query q_3 by expanding each subgoal. The result would be the pair of rules:

```
answer(O,D):- v1(sally,P1,M1) & v2(sally,O,D) &
                   v1(sally,P2,M2) & v2(sally,O,D)
answer(O,D):- v3(sally,P1) & v2(sally,O,D) &
                   v1(sally,P2,M) & v2(sally,O,D)
```

Of course, the MSL optimizer will eliminate redundant terms and simplify this solution. However, it cannot completely eliminate the subgoals using the irrelevant views v_1 and v_3. As a result, it produces an empty answer in the case that we do not know a phone or manager for Sally.

Let us again emphasize that the apparent failure of Tsimmis in Example 3.6 is due only to the fact that we contrived the mediator to export inconvenient objects. The motivation for the design of Tsimmis is that the mediators it creates may perform some very complex processing of source data to produce its exported objects. It may not be feasible to define or create objects for every conceivable query. In comparison, IM is limited in the way it can combine its sources, since it must rely on the particular search algorithm of Section 2 to combine sources.

3.4. Further comparisons of IM and Tsimmis

In addition to the differences in query processing discussed in Section 3.3, there are a number of other ways in which IM and Tsimmis differ.

3.4.1. Levels of mediation

IM is designed to have two levels: the sources and the "global mediator". In contrast, Tsimmis assumes that there is an indefinite number of levels, as the output of one mediator can be a source for a higher-level mediator. Of course, it would in principle be possible for one IM application to be a source for another. However, then we would have to wrap the first application, defining for it a fixed set of views that it exported. We thus might face the same sort of awkwardness that we explored in Example 3.6 in the context of Tsimmis.

3.4.2. Adding sources

IM makes it quite convenient to add new sources. One must write a wrapper for the sources and define its views and constraints in terms of the global concepts. However, no change to the query-processing algorithm is needed. The new views will be used whenever they are appropriate for the query. In contrast, new Tsimmis sources not only must be wrapped, but the mediators that use them have to be redefined and their MSL definitions recompiled. The administrator of the system must figure out whether and how to use the new sources.

3.4.3. Semistructured data

As we have mentioned, Tsimmis supports the notion that data does not have a fixed or uniform schema; we call such data *semistructured*. Objects with the same label, say `employee`, may have different sets of information available, and even the same information may appear with different structures in different objects. For example, some employees may be retired and have no salary subobject. Others may have an integer salary. Others may have a structured salary including base, weekly commissions, and so on. The MSL language has been designed to allow the mediator-implementor to deal with the lack of schema. The reader will find more on the important issue of handling semistructured data in [1].

3.4.4. Constraints

Only IM has an explicit mechanism for describing special properties of the information that a particular source will supply and using that information in its query-processing algorithm.

3.4.5. Automatic generation of components

Tsimmis has stressed the automatic generation of both wrappers [19] and mediators [17]. In a sense, IM has no need for automatic generation of mediators, since each application has one "mediator" and the query-processing algorithm it uses is the same as that of any other IM application. Tsimmis wrapper-generation technology could be used to wrap IM sources, although the difference in the models and languages (OEM/MSL versus Description Logic) makes direct adaptation impossible.

3.5. Extensions of the query/view model of mediation

Both IM and Tsimmis have concentrated on conjunctive queries as the principal model of both queries and views. However, there has been some exploration in both

```
answer(X) :- book(X) and QUALS(X).

QUALS(X):- QUALS(X) & Q(X).

QUALS(X):- Q(X).

Q(X) :- property(X,$pname,$value).
```

Fig. 6. A recursive program generating views.

projects of the possibility of using more powerful languages for defining views. The natural "next step" is to use recursive Datalog programs to generate infinite families of views. While describing a simple source by a finite set of views or rules is adequate, sources that support a rich query language (e.g., an SQL database) are better described by infinite families of queries.

Example 3.7. Suppose the source is an on-line bibliography that allows queries in which one or more properties are specified. We might describe the source by the recursive program of Fig. 6.

There are several things we must understand about the notation in Fig. 6. First, predicates *QUALS* and *Q* are expected to be expanded in all possible ways, generating an infinite set of conjunctive queries, each of the form

```
answer(X) :- book(X) & property() &
                 property() &...& property()
```

That is, each query asks for books X that satisfy certain properties.

The variables \$*pname* and \$*value* are parameters that are intended to be filled in for each property, allowing the CQ to match queries in which particular properties are required to have specific values. A typical query is

```
query(X) :- book(X) & property(X, author, crichton) &
                      property(X, subject, dinosaurs).
```

The idea has been explored in the context of Tsimmis in [19]. It also has been proposed as an extension to IM in [14]. In each case the satisfactory incorporation of recursively generated, infinite view sets requires extending the previously known algorithms for containment of conjunctive queries and Datalog programs.

4. Conclusions

Both IM and Tsimmis offer interesting approaches to the difficult problems of information integration. Moreover, they both draw upon similar, fairly ancient ideas from database logic, such as conjunctive query containment, as well as new ideas in database theory. They differ in a number of ways, including the underlying logic, the approach to semistructured data, and the query processing algorithm. Each represents an exciting direction for further research in database systems and for the creation of a new class of information-processing tools.

Acknowledgements

Comments by Pierre Huyn, Phokion Kolaitis, Alon Levy, Yannis Papakonstantinou, Anand Rajaraman, and Shuky Sagiv are greatly appreciated.

References

[1] S. Abiteboul, Querying semistructured data, Theoret. Comput. Sci., These Proceedings.

[2] A. Borgida, On the relative expressiveness of description logics and predicate logics, Artif. Intell. 82 (1996) 353–367.

[3] A.K. Chandra, P.M. Merlin, Optimal implementation of conjunctive queries in relational databases, Proc. 9th Ann. ACM Symp. on the Theory of Computing, 1977, pp. 77–90.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing queries with materialized views, Internat. Conf. on Data Engineering, 1995.

[5] S. Chaudhuri, M.Y. Vardi, On the equivalence of datalog programs, Proc. 11th ACM Symp. on Principles of Database Systems, 1992, pp. 55–66.

[6] W. Chen, M. Kifer, D.S. Warren, HiLog: a first order semantics for higher order programming constructs, 2nd Internat. Workshop on Database Programming Languages, Morgan-Kaufmann, San Francisco, 1989.

[7] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, J. Widom, The TSIMMIS approach to mediation: data models and languages, 2nd Workshop on Next-Generation Information Technologies and Systems, Naharia, Israel, June, 1995.

[8] http://www-i.almaden.ibm.com/cs/showtell/ garlic/Initpage.html.

[9] T. Kirk, A.Y. Levy, Y. Sagiv, D. Srivastava, The Information Manifold, AAAI Spring Symp. on Information Gathering, 1995.

[10] A. Klug, On conjunctive queries containing inequalities, J. ACM 35(1) (1988) 146–160.

[11] A. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava. Answering queries using views, Proc. 14th ACM Symp. on Principles of Database Systems, 1995, pp. 113–124.

[12] A.Y. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, Internat. Conf. on Very Large Databases, September 1996.

[13] A.Y. Levy, A. Rajaraman, J.J. Ordille, Query answering algorithms for information agents, 13th Natl. Conf. on AI, August 1996.

[14] A.Y. Levy, A. Rajaraman, J.D. Ullman, Answering queries using limited external processors, Proc. 15th ACM Symp. on Principles of Database Systems, 1996, pp. 227–237.

[15] A.Y. Levy, M.-C. Rousset, CARIN: A representation language combining Horn rules and description logics, 13th European Conf. on AI, Budapest, August 1996.

[16] A.Y. Levy, Y. Sagiv, Queries independent of update, Proc. Internat. Conf. on Very Large Data Bases, 1993, 171–181.

[17] Y. Papakonstantinou, Query processing in heterogeneous information sources, Ph.D. Thesis, Dept. of CS, Stanford University, 1996.

[18] Y. Papakonstantinou, H. Garcia-Molina, J. Widom, Object exchange across heterogeneous information sources, Internat. Conf. on Data Engineering, March, 1995.

[19] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J.D. Ullman, A query translation scheme for rapid implementation of wrappers, 4th DOOD, Singapore, December 1995.

[20] A. Rajaraman, Y. Sagiv, J.D. Ullman, Answering queries using templates with binding patterns, Proc. 14th ACM Symp. on Principles of Database Systems, 1995, pp. 105–112.

[21] R. Ramakrishnan, Y. Sagiv, J.D. Ullman, M.Y. Vardi, Proof tree transformation theorems and their applications, Proc. 8th ACM Symp. on Principles of Database Systems, 1989, pp. 172–181.
[22] Y. Saraiya, Subtree elimination algorithms in deductive databases, Doctoral Thesis, Dept. of CS, Stanford University, January 1991.
[23] O. Shmueli, Decidability and expressiveness aspects of logic queries, Proc. 6th ACM Symp. on Principles of Database Systems, 1987, pp. 237–249.
[24] http://www-db.stanford.edu/tsimmis.
[25] J.D. Ullman, Principles of Database and Knowledge-Base Systems, vol. I, Computer Science Press, New York, 1988.
[26] J.D. Ullman, Principles of Database and Knowledge-Base Systems, vol. II, Computer Science Press, New York, 1989.
[27] J.D. Ullman, Assigning an appropriate meaning to database logic with negation, in: H. Yamada, Y. Kambayashi, S. Ohta (Eds.), Computers as Our Better Partners, World Scientific Press, Singapore, 1994, pp. 216–225.
[28] R. van der Meyden, The complexity of querying indefinite data about linearly ordered domains, Proc. 11th ACM Symp. on Principles of Database Systems, 1992, pp. 331–345.
[29] J. Widom, Research problems in data warehousing, 4th Internat. Conf. on Information and Knowledge Management, 1995, pp. 25–30.
[30] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Comput. 25(3) (1992) 38–49.
[31] H.Z. Yang, P.A. Larson, Query transformation for PSJ queries, Proc. Internat. Conf. on Very Large Data Bases, 1987, pp. 245–254.
[32] X. Zhang, M.Z. Ozsoyoglu, On efficient reasoning with implication constraints, Proc. 3rd DOOD Conf., pp. 236–252.