Towards a Theory of Interestingness

V.W. Marek
Computer Science Department
University of Kentucky
Lexington, KY 40506.
marek@cs.uky.edu

V.S. Subrahmanian
Department of Computer Science
University of Maryland
College Park, MD 29742
vs@cs.umd.edu

Abstract

There are a wide variety of applications that either require or assume the existence of some underlying definition of "interestingness." However, interests vary from user to user, from situation to situation, and from one time to another. This diversity of interests cannot be captured through a single definition. In this paper, we propose a framework called *Full Interestingness Programs* (FIPs) that form a subclass of the Hybrid Knowledge Base Paradigm of Lu, Nerode and Subrahmanian. FIPs may be built "on top" of any query language whatsoever. Using FIPs, interests may be easily expressed and captured, and used on an application-specific basis using an application-independent FIP-evaluator. In this paper, we provide a formal semantics for FIPs, as well as techniques for processing requests (queries) to FIPs.

University of Maryland Technical Report CS-TR-3817 and UMIACS-TR-97-57, July 25, 1997.

1 Introduction

There are a wide variety of applications that either require or assume the existence of some underlying definition of “interestingness.” For example, data mining and knowledge discovery tools [5, 12, 13] seek to find “interesting” phenomena lurking in a body of data – yet, by and large, there are no principled definitions of interestingness. Similarly, in the so-called intelligent agent systems [10, 11, 17], agents supposedly identify “interesting” items. For example, an e-mail filtering agent usually tries to identify e-mails that its owner might find “interesting.” Likewise, in situations where an intelligent server is shipping data to multiple clients, the server may only wish to ship that part of the data that it deems interesting to the relevant client.

All the above examples use an underlying notion of “interestingness.” The “intuitive” notion of interestingness must exhibit some common properties across the above applications. However, these common properties have proved rather hard to pin down. The reason for this is that different users find different things interesting. Even worse, the same user may find something interesting under one set of circumstances, but not under another set of differences. Furthermore, interests may (and usually do) change with time. Last, but not least, users associate different degrees of interests with various topics or phenomena. Thus, we believe that any attempt to legislate a definition of interestingness is doomed to failure.

What we propose in this paper is a single theoretical framework within which users may specify their interests. This framework is rich enough to capture all the desiderata described above. As the framework is application independent, an implementation of the framework may be used across the board for a variety of applications. In other words, one does not need to reinvent the wheel each time a new notion of interestingness comes along. Rather, one can merely articulate this new notion of interestingness within the existing framework, and be guaranteed that the framework will support a range of user requests based on this notion.

Examples of scenarios that our framework supports include the following:

Scenario 1 We are interested in all Shoe salesmen whose daily sales exceed the average shoe salesman’s sales by 50% or more.

Scenario 2 We are interested in all salesmen making sales with maximal value, provided that these sales exceed $ 1000.00 per day.

Scenario 3 We are interested in identifying extremal weather conditions, but only those that affect us (i.e. we may have no interest in identifying storms in Antarctica in Jan 1999, if we have no plans of being there).

Scenario 4 We are interested in identifying groups G_1 of American citizens whose average credit card bills significantly exceed that of the average American citizen.

Scenario 5 We are interested in identifying subgroups of G_1 whose average credit card bills are significantly below the average credit card bill of members of G_1.

Scenario 6 : We are interested in broker’s b transactions on date d, providing that on date $d-1$, b received from client c at least $10,000.00.

Scenario 7 : We are interested in broker's b transactions on date d, providing that on date $d-1$, b received from client c at least \$10,000.00. Moreover, if we are interested in b's transaction on date d and b traded on d in commodity x then we are interested in the price of d on the date $d+1$.

Scenario 8 : We are interested in broker's b transactions on date d, providing that on date $d-1$, b received from client c at least \$10,000.00. We are interested in broker's b transactions on date d, providing that on date $d-1$, b received from client c_1 at least \$5,000.00.

Scenario 9 : We are interested with the degree k in broker's b transactions on date d, providing that on date $d-1$, b received from client c at least \$10,000.00. We are interested with a degree $2k$ in broker's b transactions on date d, providing that on date $d-1$, b received from client c_1 at least \$5,000.00.

The framework we use is a very small subset of Hybrid Knowledge Bases [22] which in turn builds upon work on Constraint Logic Programming [14]. *Our aim is to find a compact, but powerful language within which interests may be expressed because the more specialized the language, the easier it is to implement, and the more efficient are such implementations.*

The outline of this paper is now as follows: in Section 2, we will introduce a language called a "Basic Interestingness Language" within which we can declaratively specify the interests of a *single* user at a *single* point in time, with a *single* level of interest. We will prove various that basic interestingness programs (BIPs, for short) possess many elegant properties, the first of which is that they have a clean logical semantics. We study a sound and complete algorithm for computing of interestingness of queries. Later, in Section 3, we will extend the syntax of BIPs to accommodate multiple users, changes in time, and degrees of interest. The resulting framework, called "Full Interestingness Programs" (FIPs for short) also have a clean declarative semantics, and several elegant properties. When building an application requiring the use of interestingness, we need to specify an FIP. Once such an FIP is specified for an application, users of the application may interact with it through a range of queries. We will provide a query language for this purpose in Section 4.

2 Basic Interestingness Programs

In this section, we introduce the concept of a basic interestingness program (BIP). A BIP expresses the interests of a user at a fixed point in time. Intuitively, a BIP specifies that a user is interested in one or more queries to a database (or a heterogeneous collection of databases). In other words, given a database query language QL, BIPs specify which queries in QL are of interest to the user, and which ones are not.

Without loss of generality, we will assume that each query in QL returns as output, a *set* of objects of a given *type* (e.g. a selection operation over a relation R in a relational database returns a set of tuples obeying the schema of R). If a query returns an atomic object (e.g. an aggregate query such as `COUNT` or `SUM` of `AVG`), then we will coerce this object into a set with no loss of generality. Examples of query language that fit into this framework include the relational algebra and calculus and SQL[16], temporal query languages like TSQL[26], logic programming languages [19], heterogeneous database query languages [22, 1, 20, 2], image query languages such as PSQL[25], and multimedia query languages such as VideoSQL[24].

We will assume that QL has an associated implemented equivalence relation $\sim$. There are many possible candidates for query equivalence, and our framework can work with any of these. Some examples of such equivalence relations include:

- Syntactic Equivalence: q_1, q_2 are syntactically equivalent iff they are identical.
- Renaming Equivalence: q_1, q_2 are renaming-equivalent iff they are renamings of each other.
- D-equivalence: Given a database instance $\mathcal{D}$, q_1, q_2 are $\mathcal{D}$-equivalent if they return the same answer when evaluated over database instance $\mathcal{D}$.
- Semantic Equivalence: q_1, q_2 are semantically equivalent if for every database instance $\mathcal{D}$, it is the case that q_1, q_2 are $\mathcal{D}$-equivalent.

In most of our examples, we will $\mathcal{D}$-equivalence for illustrative purposes, though it should be clear that any equivalence relation on queries fits into our framework.

In addition, we will assume that given any query language QL, it is possible to create a new *Boolean Query Language BQL.* Queries in BQL are built "on top" of queries in QL and are defined as follows:

1. If $q_1, q_2 \in QL$ and o is an object of the same type as the output type of q, then $o \in q$, $q_1 \subseteq q_2$,and $q_1 = q_2$ are Boolean queries.
2. If b_1, b_2 are Boolean queries, then so are $b_1 \wedge b_2$, $b_1 \vee b_2$ and $\neg B_1$. We will use the letter b (possibly with indices) to range over the queries of the language BQL.

The framework in this paper will assume that a query language QL has been arbitrarily chosen but is fixed. Hence, BQL is also fixed. *We will assume that implementations exist for QL and BQL.* This is certainly reasonable as all the query languages we expect to work with are in fact implemented. Figure 1 shows the relationship between query languages, boolean query languages, data sources and BIPs/FIPs (to be introduced later) in our architecture.

2.1 Basic Interestingness Programs

We will now define the basic query interestingness language $BQIL$ as follows. First, we will have an additional metapredicate intr that will be applied to queries of the language QL. Intuitively, $\mathsf{intr}(q)$ will denote the fact that the query q is of interest. The interestingness language over the language QL consists of interestingness clauses (or clauses for short). Such clauses, C, are of the following form:

$$\mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q_1), \ldots, \mathsf{intr}(q_n). \qquad (1)$$

Here $q, q_1, \ldots, q_n$ are arbitrary queries of QL and b is an arbitrary *boolean* query, that is a query of the language BQL. The expression $\mathsf{intr}(q)$ is called the *head* of the clause C. The boolean

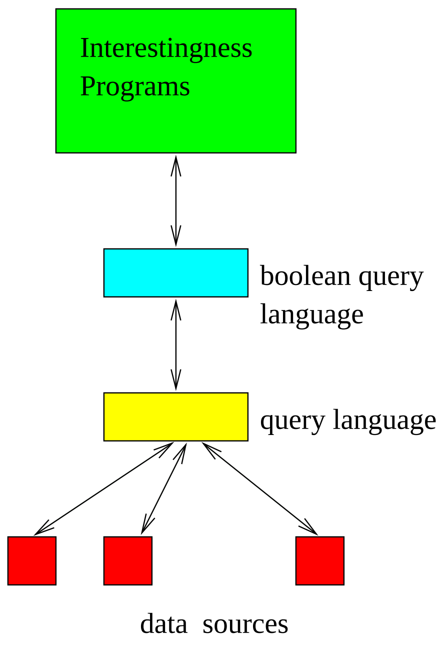

Figure 1: Interestingness Architecture

query b is called *guard* of the clause C and the expressions $\mathsf{intr}(q_1), \ldots \mathsf{intr}(q_n)$ form the *body* of C. Notice that interestingness clauses are similar to clauses of constraint logic programming. However, a major difference is that as the “guard” b is evaluated w.r.t. an underlying database $\mathcal{D}$, the truth of guards changes with changes in the database $\mathcal{D}$. In contrast, in the case of constraint logic programs, the constraint domain is always fixed (which is appropriate for the domain of the real numbers as in CLP($\mathbf{R}$), but not for a database). In addition, notice that basic interestingness clauses have a very tightly restricted syntactic form, making them amenable to efficient query evaluation and update mechanisms (to be described in Section 4) that do not apply in the case of constraint logic programs.

A (ground) clause C of the form (1) has this intuitive meaning: “Query q is interesting over $\mathcal{D}$, if the database $\mathcal{D}$ satisfies the boolean query b and the queries $q_1, \ldots, q_n$ are interesting over $\mathcal{D}$.

A *basic query interestingness program* is a collection P of basic query interestingness clauses. We will now be interested in providing a meaning for such programs. It should be clear, though, that a program itself is not sufficient to tell us what queries are important — this depends as well on the *context*, the current database. This database is used to test the guards of clauses for their applicability. Thus we are going to speak about queries that are interesting according to program P over the database $\mathcal{D}$.

In this paper we will be making an assumption that equivalent queries either are both interesting or both are not interesting. It is not obvious that this assumption is universally valid. We will assume that the implementation of the language QL provides a mechanism for testing equivalence of queries over a given database.

The process of finding an instantiation of the guard (boolean query) b in the clause C of the form (1) must be provided by *BQL*. It is akin to finding the values of variables which satisfy a formula (if the boolean queries are formulas of first-order language over the database) or finding a ground answer substitution (in the case of Horn programs). By executing the instantiation we may assume that the guard b always takes the boolean values *true* or *false*.

Let us look a the scenarios described in our introduction. Scenario 1 can be described using the basic interestingness language, as

$$\mathsf{intr}(q_1) \leftarrow$$

where q_1 is the following query:
SELECT name
FROM employee
WHERE sales > 1.5 **avg** (**SELECT** sales
FROM employee)

The scenario 2 is expressed as

$$\mathsf{intr}(q_2) \leftarrow b_2$$

Here q_2 is the query:
SELECT name
FROM employee
WHERE sales = **max** (**SELECT** sales
FROM employee)
The boolean query b_2 is a Boolean query asserting that the maximum value of the attribute sales is greater than \$1000.00. This may be expressed as:
1000 <
(**max** (**SELECT** sales
FROM employee)).

Thus, interestingness of query q_1 is qualified by the current state of our database. That is, if maximal sales are not bigger than \$1000.00, the query will not be interesting for us.

Scenario 4 requires specification of "significance level". Once this is specified, it is formalized similarly to Scenario 1. Similar effort is needed to specify Scenario 5.

The other scenarios from the introduction may be similarly expressed.

Given a pair $\langle P, \mathcal{D} \rangle$ we assign to it a collection of queries interesting for $\mathcal{D}$ under P as follows

Definition 2.1 $\mathsf{intr}_{P,\mathcal{D}}$ *is the least set V of queries satisfying the the following conditions:*

1. *V is closed under $\sim$, that is $q \in V$ and $q \sim q'$ implies $q' \in V$*
2. *whenever a clause C of the form (1) belongs to P, $q_1, \ldots, q_n \in V$ and the query b is evaluated over $\mathcal{D}$ as* true, *then $q \in V$.*

The following result shows that for any basic query interestingness program P, and any database $\mathcal{D}$, $\mathsf{intr}_{P,\mathcal{D}}$ is well-defined.

Proposition 2.1 *The set* $\mathsf{intr}_{P,\mathcal{D}}$ *is well-defined.*

Proof: The set of all queries satisfies both conditions (1) and (2). Now, the intersection of any family of sets satisfying the conditions (1) and (2) also satisfies these conditions. Thus the least set satisfying these conditions exists. □

We will now describe two different ways to describe the set of interesting queries, $\mathsf{intr}_{P,\mathcal{D}}$ as a least fixpoint of a monotonic and compact operator. These characterizations are similar to those of [19] for logic programming. Notice, that in our case we have the universe over which the guards are tested as one input to the operator.

Thus, let P be a basic interestingness language program, and $\mathcal{D}$ a database.

Definition 2.2 *The operator $T_{P,\mathcal{D}}$ maps the powerset of the set of queries of QL into itself and is defined as follows. Given a set $A \subseteq QL$,*

$$\begin{array}{rcl} T_{P,\mathcal{D}}(A) & = & \{q : \textit{ There is a clause } C = \mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q'_1), \ldots, \mathsf{intr}(q'_n) \textit{ in } P \textit{ such that} \\ & & q_1, \ldots q_n \in A, q_1 \sim q'_1, \ldots q_n \sim q'_n \textit{ and } b \textit{ is true in } \mathcal{D}\} \end{array}$$

The *iterations* of $T_{P,\mathcal{D}}$ are defined as follows:

$$\begin{array}{rcl} T^0_{P,\mathcal{D}} & = & \emptyset \\ T^{i+1}_{P,\mathcal{D}} & = & T_{P,\mathcal{D}}(T^i_{P,\mathcal{D}}) \\ T^\omega_{P,\mathcal{D}} & = & \bigcup_{i<\omega} T^i_{P,\mathcal{D}} \end{array}$$

$T_{P,\mathcal{D}}$ is said to be *monotonic* iff $A \subseteq A' \to T_{P,\mathcal{D}}(A) \subseteq T_{P,\mathcal{D}}(A')$. An operator T is said to be *compact* if it preserves unions of increasing families of sets, that is if $T(\bigcup_n X_n) = \bigcup_n T(X_n)$ for every increasing family $\langle X_n \rangle_{n\in\omega}\rangle$.

We have the following

Proposition 2.2 *Let P be a basic query interestingness program, and let $\mathcal{D}$ be a database. Then $T_{P,\mathcal{D}}$ is a compact and monotonic operator in QL. Thus $T_{P,\mathcal{D}}$ possesses a least fixpoint $F_{P,\mathcal{D}}$, reachable in at most ω steps.*

Operator $T_{P,\mathcal{D}}$ and its least fixpoint can be used to characterize $\mathsf{intr}_{P,\mathcal{D}}$.

Proposition 2.3 *The set* $\mathsf{intr}_{P,\mathcal{D}}$ *coincides with* $\{\mathsf{intr}(q) : q \in F_{P,\mathcal{D}}\}$.

Proof: It is easy to see that the set $\{\mathsf{intr}(q) : q \in F_{P,\mathcal{D}}\}$ satisfies conditions (1) and (2) above and hence $\mathsf{intr}_{P,\mathcal{D}} \subseteq \{\mathsf{intr}(q) : q \in F_{P,\mathcal{D}}\}$. On the other hand it is immediate that $T_{P,\mathcal{D}}(\{q : \mathsf{intr}(q) \in \mathsf{intr}_{P,\mathcal{D}}\}) \subseteq \{q : \mathsf{intr}(q) \in \mathsf{intr}_{P,\mathcal{D}}\}$. Consequently, $\{q : \mathsf{intr}(q) \in \mathsf{intr}_{P,\mathcal{D}}\}$ is a prefixpoint of $T_{P,\mathcal{D}}$. Thus the least fixpoint of $T_{P,\mathcal{D}}$ is included in $\{q : \mathsf{intr}(q) \in \mathsf{intr}_{P,\mathcal{D}}\}$. □

Next, we show another characterization of the set $\mathsf{intr}_{P,\mathcal{D}}$.

Definition 2.3 *Let P be a basic query interestingness program and $\mathcal{P}$ be a database. The reduct of P by $\mathcal{D}$, $P_{\mathcal{D}}$ is the program in which clauses have no guards, and which is obtained in the following two steps:*

1. *For every clause C of the form (1), if the guard b of C is false in $\mathcal{D}$, eliminate C altogether.*
2. *In the remaining clauses, eliminate the guards from the body.*

Assign the following operator $T^P_{\mathcal{D}}$ (acting on subsets of QL) to the program $P_{\mathcal{D}}$:

$T^P_{\mathcal{D}}(A) = \{q :$ There is a clause $C = \mathsf{intr}(q) \leftarrow \mathsf{intr}(q'_1), \ldots, \mathsf{intr}(q'_n)$ in $P_{\mathcal{D}}$ such that there exist $q_1, \ldots q_n \in A$, $q_1 \sim q'_1, \ldots q_n \sim q'_n$ $\}$

We then have the following proposition.

Proposition 2.4 *Let P be a basic query interestingness program, and $\mathcal{D}$ a database. Then $T^P_{\mathcal{D}}$ is a compact and monotonic operator in QL. Thus $T^P_{\mathcal{D}}$ possesses a least fixpoint $G_{P,\mathcal{D}}$. That fixpoint is reachable in at most ω steps.*

The fixpoint $G_{P,\mathcal{D}}$ provides another characterization of $\mathsf{intr}_{P,\mathcal{D}}$.

Proposition 2.5 *The set $\mathsf{intr}_{P,\mathcal{D}}$ coincides with $\{\mathsf{intr}(q) : q \in G_{P,\mathcal{D}}\}$.*

In principle, the equivalence relation $\sim$ may have infinite equivalence classes. That is, given a query q there may be infinitely many queries q' equivalent to q. This may seem, at the first glance, make the use of the set $\mathsf{intr}_{P,\mathcal{D}}$ awkward. If the language QL provides a fast mechanism for testing equivalence of queries, we can avoid the problem of infinite sets of interesting queries by restricting our attention to their *bases*.

Definition 2.4 *A basis for the set $\mathsf{intr}_{P,\mathcal{D}}$ is any set B satisfying these conditions:*

1. *$B \subseteq \mathsf{intr}_{P,\mathcal{D}}$*
2. *whenever $q \in \mathsf{intr}_{P,\mathcal{D}}$ then there is $q' \in B$ such that $q \sim q'$.*

We then have the following program

Proposition 2.6 *Let P be a finite basic query interestingness program and $\mathcal{D}$ be a database. Then a basis for $\mathsf{intr}_{P,\mathcal{D}}$ can be found in time polynomial in P with respect to an oracle $O^{\mathcal{D}}$ that provides the answers for equivalence of queries over $\mathcal{D}$.*

Proof: By Proposition 2.3, the set of interesting queries can be computed as the least fixpoint of the operator $S_{P_{\mathcal{D}}}$.

First, observe that the program $P_{\mathcal{D}}$ can be computed in time linear in the number of clauses in P (thus linear in the size of P) as follows. Namely, we inspect the clauses of P and in each of these clauses, C, ask if the guard b of C is true under $\mathcal{D}$. If this is the case, the clause is C is kept, and the guard b is eliminated. Otherwise the entire clause C is eliminated.

Once the program $P_{\mathcal{D}}$ is computed we use a modified algorithm of computing the least model of a Horn program ([8]). Roughly, that algorithm works as follows: to each atom we attach the list of clauses of which it is the head, and for each atom, we set up the list of clauses to whose bodies that atom belong. Moreover, with each clause we associate a counter counting the atoms in the body that still need to be proven. Originally, this counter gets the value equal to the number of atoms in the body. An atom is proven when it is a head of a clause with the associated counter value equal to 0. Once an atom is proven, we lower the counter values associated with all clauses that have that atom in the body. Dowling and Gallier prove that this algorithm computes the least model of a Horn program in linear time. Notice that the algorithm of [8] requires modification because when we reduce a list of atoms in the body of a clause because one of these atoms have been proven, we may need to consult the oracle $O^{\mathcal{D}}$. Indeed, we may have just proven that $\mathsf{intr}(q)$ holds, but the body of some clause contains $\mathsf{intr}(q')$ and $q' \sim q$. This is the reason why the modified algorithm is no longer linear, but only quadratic in the size of the program P. Yet, it is easy to see that this algorithm correctly computes a least model of $P_{\mathcal{D}}$. □

The procedure implied by the proof of Proposition 2.6 will be denoted by *Base* and will be used in Section 2.4.

2.2 Basic Interestingness Programs with Integrity Constraints

We will now discuss the concept of interestingness in the presence of integrity constraints. First, we will assume that the integrity constraints that we consider are boolean queries. That is the set of integrity constraints IC consists of boolean queries that are supposed to be true.

Notice that there is always a semantic consequence operation associated with any set of integrity constraints of this sort. Namely

$$IC \models c \text{ if for every } \mathcal{D} \text{ whenever all constraints in } IC \text{ are true, then also } c \text{ is true}$$

Given a program P and a set, IC, of integrity constraints, we can reduce the program P in the manner similar to that used to reduce P by a database $\mathcal{D}$. Namely

1. If C is a clause in P and IC does not entail the guard b of C then eliminate C altogether.

2. In the remaining clauses eliminate guards.

The resulting program is denoted by P_{IC}. The following fact is proven in a manner similar to Proposition 2.5.

Proposition 2.7 *Let IC be a set of integrity constraints, and let P_{IC} be the reduct of P described above. Then for every query q that belongs to the least fixpoint of the operator associated with P_{IC}, for every database $\mathcal{D}$ satisfying all the constraints in IC,* $\mathsf{intr}(q) \in \mathsf{intr}_{P,\mathcal{D}}$.

Thus, with a set, IC, of integrity constraints, we can associate its P-companion, that is, the set of interestingness atoms that P implies whenever IC are true. That is, given P and a database $\mathcal{D}$ satisfying IC, the atoms in P-companion of IC are always interesting. The operator assigning to IC its P-companion is monotonic. That is, the larger IC is, the bigger its P-companion.

2.3 Properties of Basic Interestingness Programs

Definition 2.5 *An interestingness clause c is called* simple *if it is of the form*

$$\mathsf{intr}(q) \leftarrow b$$

where b is a boolean query and q is a query of QL.

Proposition 2.8 *For every interestingness program P there is an interestingness program P' such that:*

1. *P' consists of simple clauses*
2. *For every database $\mathcal{D}$, $\mathsf{intr}_{P,\mathcal{D}} = \mathsf{intr}_{P',\mathcal{D}}$.*

Proof: We describe an unfolding procedure that allows us to get P'. The program P' is constructed as the union of programs Q_n. Q_0 is defined as the part of P consisting of simple clauses. Assume Q_n is already defined. For every clause $\mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q'_1), \ldots, \mathsf{intr}(q'_m)$ of P and for every choice of clauses in Q_n, $\mathsf{intr}(q''_1) \leftarrow b_1, \ldots \mathsf{intr}(q''_m) \leftarrow b_m$, form this clause C:

$$\mathsf{intr}(q) \leftarrow b \wedge b_1 \ldots \wedge b_m \wedge q'_1 = q''_1 \wedge \ldots \wedge q'_m = q''_m.$$

and put C in Q_{m+1}. The resulting program $P' = \bigcup_{n \in \omega} Q_n$ has the desired property. $\Box$

The construction used above is, essentially, that of [9].

We will now be investigating the preservation properties for interestingness programs.

First, let us notice that increasing the *program* (but preserving the database) always increases the collection of interesting queries. That is, the operator that assigns to the pair $\langle P, \mathcal{D} \rangle$, the set $\mathsf{intr}_{P,\mathcal{D}}$ is monotone in the first argument. Formally we have the following:

Proposition 2.9 *Let P_1, P_2 be two basic interestingness programs and $\mathcal{D}$ a database. If $P_1 \subseteq P_2$ then $\mathsf{intr}_{P_1,\mathcal{D}} \subseteq \mathsf{intr}_{P_2,\mathcal{D}}$.*

The operation intr is not, in general, monotonic in the second argument. That is a larger database does not necessarily yield more interesting queries. Nevertheless, some interesting (no pun intended !) properties still can be proved.

Let $\mathcal{D}_1, \mathcal{D}_2$ be two databases. We say that the the transition from $\mathcal{D}_1$ to $\mathcal{D}_2$ *preserves constraints* of the program P, if for every clause of the form (1), whenever c is true in $\mathcal{D}_1$, then c is also true in $\mathcal{D}_2$.

Lemma 2.1 *If P is a basic interestingness program, and the transition from $\mathcal{D}_1$ to $\mathcal{D}_2$ preserves constraints of P, then $\mathsf{intr}_{P,\mathcal{D}_1} \subseteq \mathsf{intr}_{P,\mathcal{D}_2}$.*

Lemma 2.2 *If P is a basic interestingness program, and the transition from $\mathcal{D}_1$ to $\mathcal{D}_\in$ preserves constraints of P, then every basis of $\mathsf{intr}_{P,\mathcal{D}_1}$ extends to a basis of $\mathsf{intr}_{P,\mathcal{D}_2}$.*

Fortunately, there are classes of programs for which inclusion is a transition that preserves constraints. Specifically, assume that the query language QL is a first order many-sorted language of predicate calculus over a fixed set of constants. The following property, due essentially to Tarski, (see Chang and Keisler [6])

Lemma 2.3 *The positive sentences of QL are preserved upwards. That is, if c is a positive sentence of QL, $\mathcal{D}_1 \subseteq \mathcal{D}_2$, and $\mathcal{D}_1 \models c$ then also $\mathcal{D}_2 \models c$.*

Lemma 2.3 implies the following property of interestingness programs.

Proposition 2.10 *Let P be a program such that all its constraints are positive. Then whenever $\mathcal{D}_1 \subseteq \mathcal{D}_2$, then $\mathsf{intr}_{P,\mathcal{D}_1} \subseteq \mathsf{intr}_{P,\mathcal{D}_2}$.*

When all constraints are negative, the opposite inclusion holds. Specifically, we have

Proposition 2.11 *Let P be a program such that all its constraints are negative. Then whenever $\mathcal{D}_1 \subseteq \mathcal{D}_2$, then $\mathsf{intr}_{P,\mathcal{D}_2} \subseteq \mathsf{intr}_{P,\mathcal{D}_1}$.*

The use of boolean constraints and the fact that the database may change implies that an interestingness program really encodes a whole collection of programs, namely, for each database $\mathcal{D}$ the program $P_\mathcal{D}$ is encoded in P.

We conclude this section with remarks on the difference with modal formalisms. Although on the first glance it may seem that the metapredicate intr acts like a modality, the similarity is purely superficial. First, there is a difference of types. Whereas the formulas φ and $\Box\varphi$ are of the same type (i.e. formulas), q and $\mathsf{intr}(q)$ are not, in general, of the same type. Specifically, q is *not* a formula and $\mathsf{intr}(q)$ is a formula. However, even if we allow for queries to be formulas, there is no reason that any modal axiom could represent user's interests. In fact, it is immediate that in this case usual modal axioms do not make sense. For instance, adopting the modal scheme **K** (cf. Chellas[7]) would imply that tautologies are interesting, which certainly is not the case!

2.4 Processing Queries to Propositional BIPs

In this section we will investigate several algorithms for testing and computing interestingness with respect to BIPs that contain no variables. Later, in Section 4, we will extend this to the first order case.

First of all, let us describe basic questions related to interestingness. The main questions are

1. Given an interestingness program P, a database $\mathcal{D}$ and a query q, is query q interesting? That is, does $\mathsf{intr}(q)$ belong to $\mathsf{intr}_{P,\mathcal{D}}$?

2. Find a basis for $\mathsf{intr}_{P,\mathcal{D}}$

3. Under appropriate restrictions, can we find such basis in an incremental fashion?

We describe a modification of the basic backward chaining algorithm to process interestingness queries. It is a straightforward modification of resolution for Horn programs. However, it is important to note that these modifications are in fact needed.

An interestingness goal is a finite list

$$G = [\mathsf{intr}(q_1), \ldots \mathsf{intr}(q_n)]$$

Given a clause

$$C = \mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q'_1), \ldots, \mathsf{intr}(q'_m)$$

we say that C can be used for expansion of G at i over $\mathcal{D}$ if

- $\mathcal{D} \models b$
- $q \sim q_i$

The result of an expansion is then

$$[\mathsf{intr}(q_1), \ldots \mathsf{intr}(q_{i-1}), \mathsf{intr}(q'_1), \ldots, \mathsf{intr}(q'_m), \mathsf{intr}(q_{i+1}), \ldots \mathsf{intr}(q_n)]$$

With this definition, we define the refutation of an interestingness atom $\mathsf{intr}(p)$ in an usual fashion, namely as a sequence of lists starting at $[\mathsf{intr}(p)]$ and ending with an empty list, and each next list is an expansion of the previous one by a clause that can be used. We now have the following proposition:

Proposition 2.12 *The set* $\{\mathsf{intr}(p) :\ [\mathsf{intr}(P)]$ *possesses a refutation*$\}$ *coincides with* $\mathsf{intr}_{P,\mathcal{D}}$.

Proposition 2.12 is proved by a version of the usual argument showing that the least model of a Horn program coincides with the set of atoms for which resolution refutation exists. It is quite clear the the construction given above can easily be transformed into an algorithm.

We will now formulate a result that implies an algorithm for incremental computation of interestingness queries for a class of programs.

Recall, that we say that a boolean query b is preserved under the transition from database $\mathcal{D}_1$ to database $\mathcal{D}_2$ if $\mathcal{D}_1 \models b$ implies that $\mathcal{D}_2 \models b$.

When the boolean queries serving as constraints in interestingness clauses of program P are preserved under the transition form $\mathcal{D}_1$ to $\mathcal{D}_2$ then we have the following incremental algorithm for computation of $\mathsf{intr}_{P,\mathcal{D}_2}$.

Definition 2.6 *1. Given a set $\mathcal{X}$ of queries of the language QL, and an interestingness clause C of the form (1),*

$$\mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q_1), \ldots, \mathsf{intr}(q_n).$$

an input reduct *of C by $\mathcal{X}$, $ir(C, \mathcal{X})$ is* **nil** *if $q \in \mathcal{X}$ and the clause*

$$\mathsf{intr}(q) \leftarrow b, \mathsf{intr}(q_{r_1}), \ldots, \mathsf{intr}(q_{r_m}).$$

where $q_{r_1}, \ldots, q_{r_m}$ are all queries in the body of C that do not belong to $\mathcal{X}$, if $q \notin \mathcal{X}$.

2. $ir(P, \mathcal{X}) = \{ir(C, \mathcal{X}) : C \in P\}$.

The notion of input reduct by $\mathcal{X}$ has the following meaning. Once the interestingness of queries from $\mathcal{X}$ has been established, we do not need to reestablish them again, and we can use the fact that those are computed as interesting in our further computations. Moreover, the clauses that have an occurrence of a query from X in the head can be safely eliminated. It is easy to devise a procedure for reduction of clauses by a set of queries. We will denote this procedure by *Reduce* and use it below.

The following result forms a basis for an algorithm for an incremental computation of interestingness of queries provided that all the constraints of clauses in P are preserved under the transition.

Theorem 2.1 *Let $\mathcal{D}_1$, $\mathcal{D}_2$ be two databases, P is an interestingness program and for all clauses C of P, the constraint in C is preserved under the transition form $\mathcal{D}_1$ to $\mathcal{D}_2$. then*

$$\mathsf{intr}_{P,\mathcal{D}_2} = \mathsf{intr}_{P,\mathcal{D}_1} \cup \mathsf{intr}_{ir(P,\mathsf{intr}_{P,\mathcal{D}_1}),D_2}.$$

Theorem 2.1 says that if all the constraints of clauses are preserved under the transition from $\mathcal{D}_1$ to $\mathcal{D}_2$ then we can compute the set of interesting queries incrementally, as follows. First, we compute the set of interesting queries over $\mathcal{D}_1$ with respect to P. Then we reduce the program P by the set thus computed. Subsequently, we compute the set of interesting queries over the reduced program (and $\mathcal{D}_2$). These sets are necessarily disjoint by our construction. The union of the two computed sets is the set of interesting queries over $\mathcal{D}_2$.

Notice that if we precompiled the set $\mathsf{intr}_{P,\mathcal{D}_1}$ then we are getting precisely a technique for incremental computation of the set $\mathsf{intr}_{P,\mathcal{D}_2}$.

Procedure *Base* computes a basis for the set of interesting queries from a given interestingness program P and a database $\mathcal{D}$. Procedure *Base* was entailed by the Proposition 2.6. We will also assume that we have a procedure to test $\sim$ (cf line (3) of Figure 2).

3 Extended Interestingness Programs

In the preceding section, we introduced the concept of a basic query interestingness program. Basic query interestingness programs allowed a *single* user to specify a *single* level of interest at a *single* point in time.

Algorithm for testing if, given an interestingness program P, database $\mathcal{D}$, and a query q, q is interesting.
Input: a finite interestingness program P, a database $\mathcal{D}$ and a query q.
Output: The decision if q is interesting.

```
call procedure $Base(B)$
    for $q' \in B$
    if $q' \sim q$
    return $true$
    rof
return $false$
```

Figure 2: Algorithm for testing if q belongs to $\mathsf{intr}_{P,\mathcal{D}}$

Algorithm for incremental computation of $\mathsf{intr}_{P,\cdot}$ providing the constraints are preserved under the transition form $\mathcal{D}_1$ to $\mathcal{D}_2$.
Input: a finite interestingness program P, and two databases $\mathcal{D}_1$ and $\mathcal{D}_2$ such that all the constraints of clauses in C are preserved under the transition.
Output: A basis B'' for $\mathsf{intr}_{P,\mathcal{D}_2}$

```
call procedure $Base(P, \mathcal{D}_1, B)$
call procedure $Reduce(P, B, P')$
call procedure $Base(P', \mathcal{D}_2, B')$
$B'' := B \cup B'$
return $B''$.
```

Figure 3: Algorithm for incremental computation of a basis for $\mathsf{intr}_{P,\mathcal{D}_2}$ providing the constraints are preserved under the transition from $\mathcal{D}_1$ to $\mathcal{D}_2$.

However, as already mentioned in Section 1, different users may have different interests. Even a single user may have different interests at different points in time. Last, but not least, a single user may have multiple levels of interest, and may ascribe different levels of interest to different phenomena. In this section, we will enhance the language of basic query interestingness so that such desiderata may be expressed.

3.1 Adding Users and Time

Suppose $\mathcal{U}$ is a finite set of user names. Let $\mathcal{T}$ denote the set of all natural numbers (time points). First, we define a *ternary* interestingness predicate, $\mathsf{intr3}$ that (informally) takes three arguments:

- The first argument is a query;
- The second argument is a user;
- The third argument is a time point.

Intuitively, the *ground atom*, $\mathsf{intr3}(q, u, t)$ may be read as: "User u is interested in query q at time t."

Returning to the motivating examples in the Introductory section of this paper, the reader will note that the formalization of Scenario 2 requires the ability to specifically refer to/denote users. Similarly, scenarios 6, 7, and 8 involved time. In each of those, our interest is predicated on the evaluation of queries in temporal databases. In scenario 6, the interestingness of a certain query on a date d is dependent on the state of the (temporal) database on the previous date. In scenario 8, the interest in two different queries depended on the state of the database on a different date. Scenario 7 was different from scenario 8 in that interestingness of a query depended not only on the state of the database, but also on the fact that another query was already interesting.

Formally, we assume we have 2 sets of variables – a set $V_{\mathcal{U}}$ of variables ranging over $\mathcal{U}$, and a set $V_{\mathcal{T}}$ of variables ranging over $\mathcal{T}$. In addition, $\mathcal{F}_{\mathcal{T}}$ is a finite set of function symbols (each with an associated arity). Without any loss of generality, we will assume that each of the function symbols in $\mathcal{F}_{\mathcal{T}}$ has a pre-interpreted meaning.

We are now ready to define $\mathcal{U}$-terms and $\mathcal{T}$-terms.

Definition 3.1 *$\mathcal{U}$-terms and $\mathcal{T}$-terms are defined as follows:*

- *A $\mathcal{U}$-*term *is a member of $\mathcal{U} \cup V_{\mathcal{U}}$.*
- *A $\mathcal{T}$-*term *is inductively defined as follows:*
  - *Every member of $\mathcal{T} \cup V_{\mathcal{T}}$ is a $\mathcal{T}$-*term.
  - *If $\tau_1, \ldots, \tau_m$ are $\mathcal{T}$-terms, and f is an m-ary function symbol in $\mathcal{F}_{\mathcal{T}}$, then $f(\tau_1, \ldots, \tau_m)$ is a $\mathcal{T}$-term.*

For example, if John is a user, then "john" is a $\mathcal{U}$-term. If U is a variable ranging over users, then U is a $\mathcal{U}$-term. Similarly, 5, T and $(+(T,5))$ are all $\mathcal{U}$-terms.

Definition 3.2 *Suppose* $\mathsf{intr}(q)$ *is an interestingness atom, and* ν *is a* $\mathcal{U}$*-term, and* τ *is a* $\mathcal{T}$*-term. Then* $\mathsf{intr3}(q,\nu,\tau)$ *is an* intr3-atom.

In order to present some examples of intr3-atoms, we present two simple SQL queries q_1, q_2 below which will later be used to construct some interestingness atoms:

q_1	q_2

SELECT name, salary
FROM employee
WHERE salary > 100,000 **AND**
expenses > salary.
	SELECT name, salary
	FROM employee
	WHERE title = secretary **AND**
	salary > 100,000

Based on the above two SQL queries, we may now construct some intr3-atoms:

- $\mathsf{intr3}(q_1, mr\_auditor, 5)$
  This interestingness atom says that a user, called `Mr. Auditor`, is interested at time 5, in all people (and their salaries) who make over 100K and whose expenses are very high. He may suspect that such people are mis-using their expense accounts.
- $\mathsf{intr3}(q_2, mr\_auditor, 5)$
  This interestingness atom says that a user, called `Mr. Auditor`, is interested at time 5, in all secretaries who make an exorbitant salary. Again, there may be reasons for him to suspect something inappropriate.

Definition 3.3 *If* b *is a Boolean query in QL and* $\mathsf{intr3}(q_0,\nu_0,\tau_0),\ldots,\mathsf{intr3}(q_m,\nu_m,\tau_m)$ *are* intr3*-atoms, then*

$$\mathsf{intr3}(q_0,\nu_0,\tau_0) \quad\leftarrow\quad b, \mathsf{intr3}(q_1,\nu_1,\tau_1),\ldots,\mathsf{intr3}(q_m,\nu_m,\tau_m) \tag{2}$$

is called an extended interestingness clause. *An* extended interestingness program *(EIP) is a finite set of extended interestingness clauses.*

Returning to the preceding example, we may easily construct the following extended interestingness clauses. First, we construct the following query $q_3(T)$:

SELECT E1.Name
FROM employee E1, E2
WHERE E1.Time=T **AND** E1.Name=E2.Name **AND**
E2.Time=(T-6) **AND** E2.Title=E1.Title **AND**
E2.Title = secretary **AND** E1.Salary > 100,000**AND**
E2.Salary < 40,000.

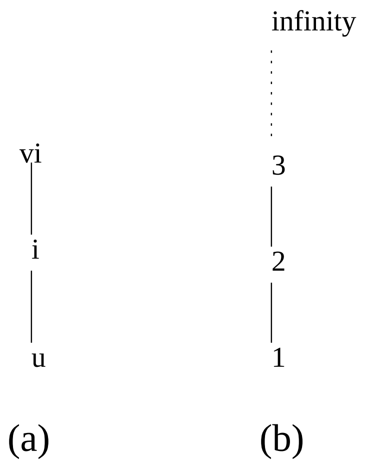

Figure 4: Two example interestingness lattices

Let $q_4(T)$ be the same query as the one above *except* that the first line replaces "E1.Name" by "E1.boss." Then the following is an extended interestingness clause:

$$\mathsf{intr3}(q_4(T), mr\_auditor, T) \quad \leftarrow \quad q_3(T).$$

This extended interestingness clause says that Mr. Auditor is interested in finding the names of all bosses whose secretaries jumped from relatively low salaries (below 40K) to very high salaries (over 100K) in less than 6 months. It is very important for the reader to note that the *variable* T ranging over time points occurs both in the head of this clause as well as in the body, i.e. it occurs in both $\mathsf{intr3}(q_4, mr\_auditor, T)$ and in the query q_3.

3.2 Adding Levels of Interest

Suppose $(\mathsf{IL}, \sqsubseteq)$ is a complete lattice whose elements denote *levels* of interest. Examples of such complete lattices include both *qualitative* and *quantitative levels of interest.* Here are some example interestingness lattices:

- The set of all real numbers between 0 and 1 (inclusive), ordered by the usual $\leq$ ordering. In this lattice, 0 denotes absolute lack of interest, while 1 denotes complete interest. 0.6 for instance denotes a greater level of interest than 0.5.
- Figure 4(a) shows another interestingness lattice with three levels of interest u (uninterested), i (interested), and vi (very interested).
- Figure 4(b) shows another interestingness lattice consisting of the positive integers, together with the symbol "infinity" which denotes maximal interest.

In any application requiring the use of interestingness, we assume that the application developer will pick an interestingness lattice $(\mathsf{IL}, \sqsubseteq)$. S/he may pick any complete lattice whatsoever.

Given an interestingness lattice $(\mathsf{IL}, \sqsubseteq)$, we associate a set V_{IL} of interestingness variables , and a set $\mathcal{F}_{\mathsf{IL}}$ of interpreted function symbols (each having an associated arity). We may now define interestingness terms.

Definition 3.4 $(\mathsf{IL}, \sqsubseteq)$*-terms are defined as follows:*

- *Any member of* $V_{\mathsf{IL}} \cup \mathsf{IL}$ *is an* $(\mathsf{IL}, \sqsubseteq)$*-term.*
- *If* $\xi_1, \ldots, \xi_m$ *are* $(\mathsf{IL}, \sqsubseteq)$*-terms and* f *is an* m*-ary function symbol in* $\mathcal{F}_{\mathsf{IL}}$*, then* $f(\xi_1, \ldots, \xi_m)$ *is an* $(\mathsf{IL}, \sqsubseteq)$*-term.*

Definition 3.5 *If* $\mathsf{intr3}(q, \nu, \tau)$ *is an* $\mathsf{intr3}$*-atom, and* ξ *is an* $(\mathsf{IL}, \sqsubseteq)$*-term, then* $\mathsf{intr3}(q, \nu, \tau) : \xi$ *is an* annotated interestingness atom*.*

Intuitively, if $\mathsf{intr3}(q, \nu, \tau) : \xi$ is variable free, then this annotated interestingness atom may be read as saying: "At time τ, user ν is interested in query q with *at least* level ξ of interest.

We show below, a few simple annotated interestingness atoms using the $[0, 1]$ lattice of reals.

- $\mathsf{intr3}(q_1, mr\_auditor, 5) : 0.8$
  This annotated interestingness atom says that a user, called `Mr. Auditor`, is interested, with degree of interest 0.8, at time 5, in all people (and their salaries) who make over 100K and whose expenses are very high.
- $\mathsf{intr3}(q_2, mr\_auditor, 5) : 0.6$
  This annotated interestingness atom says that a user, called `Mr. Auditor`, is interested, with degree 0.6, at time 5, in all secretaries who make an exorbitant salary.
- Notice that Mr. Auditor's interest in the first annotated interestingness atom above is higher than his interest in the second.

Definition 3.6 *If* $\mathsf{intr3}(q_0, \nu_0, \tau_0), \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m)$ *are* $\mathsf{intr3}$*-atoms,* b *a boolean query, and* $\xi_0, \ldots, \xi_m$ *are* $(\mathsf{IL}, \sqsubseteq)$*-terms, then*

$$\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0 \quad \leftarrow \quad b, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m \tag{3}$$

is called a full interestingness clause*. A* full interestingness program *(FIP) is a finite set of full interestingness clauses.*

To see an example of an FIP that uses our running example, consider the following set of rules:

$$\begin{array}{rcl} \mathsf{intr3}(q_1, mr\_auditor, T) : 0.5 & \leftarrow & q_1. \\ \mathsf{intr3}(q_4, mr\_auditor, T) : 0.7 & \leftarrow & q_3. \\ \mathsf{intr3}(q_4, mr\_auditor, T) : min(1, 0.7 + \dfrac{V}{2}) & \leftarrow & q_3, \mathsf{intr3}(q_1, mr\_auditor, T) : V. \end{array}$$

The above FIP may now be informally read as follows:

- Mr. Auditor is interested, with degree 0.5, in all employees who make over 100K and who have high expense accounts.
- Mr. Auditor is interested, with degree 0.7, in all employees who are bosses of secretaries whose salaries have jumped enormously during the past 6 months.
- Mr. Auditor is interested, with degree $min(1, 0.7 + \frac{V}{2})$ (which is a higher degree than 0.7), in all employees who;
  - make over 100K and
  - who have high expense accounts and
  - whose secretaries' salary has jumped enormously during the past 6 months.

The above situation may arise in an application where an auditor is looking for "patterns" in the employee database that might point him towards fruitful avenues of investigation. The above queries are examples of "patterns" that interest the auditor. Based on the results obtained, he might identify potential problems that are worth further investigation.

3.3 Semantics of FIPs

At this point, we have provided a formal definition of FIPs. In this section, we define the semantics of FIPs.

Let us use the symbol INT^3 to denote the set of all *ground* intr3-atoms in our language.

Definition 3.7 *An* interestingness interpretation*,* II*, is a mapping from* INT^3 *to* $(\mathsf{IL}, \sqsubseteq)$*.*

Intuitively, suppose we have an interestingness interpretation II such that

$$\begin{aligned} \mathsf{II}(\mathsf{intr3}(q_0, john, 5)) &= 0.9. \\ \mathsf{II}(\mathsf{intr3}(q_0, mary, 5)) &= 0.15. \end{aligned}$$

This says that *according to interestingness interpretation* II, John has a 0.9 level of interest in query q_0 at time 5. In contrast, Mary has level of interest 0.15 in query q_0 at time 5. In this example, of course, we are considering as our interestingness lattice, the unit interval $[0, 1]$ of reals, under the usual $\leq$ ordering.

Given two interestingness interpretations II and II', we may extend the $\sqsubseteq$ ordering on $(\mathsf{IL}, \sqsubseteq)$ to interestingness interpretations as follows: $\mathsf{II} \sqsubseteq \mathsf{II}'$ iff for all queries q, all users u and all times t, $\mathsf{II}(\mathsf{intr3}(q, u, t)) \sqsubseteq \mathsf{II}'(\mathsf{intr3}(q, u, t))$. The reader may easily verify that INT^3 is a complete lattice under this ordering.

Based on this intuition, we may now extend the concept of $\mathsf{intr}_{P,\mathcal{D}}$ introduced in Definition 2.1, to handle the case of multiple users, time and levels of interest, through the following definition.

Definition 3.8 $\mathsf{intr}^3_{P,\mathcal{D}}$ *is the smallest (with respect to the* $\sqsubseteq$*-ordering) interestingness interpretation* II *satisfying the the following conditions:*

1. II *is closed under* $\sim$*, that is if* $q \sim q'$*, then* $\mathsf{II}(\mathsf{intr3}(q,u,t)) = \mathsf{II}(\mathsf{intr3}(q',u,t))$ *for all users* u *and all times* t*..*

2. *whenever a clause* C *of the form (3) belongs to* P*, and for all* $1 \leq i \leq n$*,* $\xi_i \sqsubseteq \mathsf{II}(q_i)$ *and the query* b *is evaluated over* $\mathcal{D}$ *as* true*, then* $\xi_0 \sqsubseteq \mathsf{II}(q_0)$*.*

The following result shows that for any FIP P, and any database $\mathcal{D}$, $\mathsf{intr}^3_{P,\mathcal{D}}$ is well-defined.

Proposition 3.1 *The interesting interpretation* $\mathsf{intr}^3_{P.\mathcal{D}}$ *is well-defined.*

We now extend the fixpoint and model theoretic characterization described in Section 2 to handle the introduction of users, uncertainty, and time. First, we extend the operation $T_{P,\mathcal{D}}$ as follows.

Definition 3.9 *The operator* $T3_{P,\mathcal{D}}$ *maps* INT^3 *to* INT^3 *and is defined as follows. Given an interestingness interpretation* II*, and an interestingness atom* $\mathsf{intr3}(q,u,t)$*:* $T3_{P,\mathcal{D}}(\mathsf{II})(\mathsf{intr3}(q,u,t)) = \sqcup\{\xi_0 \mid$ *There is a clause* $C = \mathsf{intr3}(q_0,u_0,t_0) : \xi_0 \leftarrow b, \mathsf{intr3}(q_1,u_1,t_1) : \xi_1 \,\&\ldots\&\, \mathsf{intr3}(q_n,u_n,t_n) : \xi_n$ *in* P *such that for all* $1 \leq i \leq n$*. there exists a query* $q'_i \sim q_i$ *such that* $\xi_i \sqsubseteq II(\mathsf{intr3}(q'_i,u_i,t_i))$ *and* b *is true in* $\mathcal{D}\}$*.*

The following result extends the result in Proposition 3.2 to the case of FIPs.

Proposition 3.2 *Let* P *be a FIP, and* $\mathcal{D}$ *a database. Then* $T3_{P,\mathcal{D}}$ *is a monotonic operator in* QL*. Thus* $T_{P,\mathcal{D}}$ *possesses a least fixpoint* $F3_{P,\mathcal{D}}$*. Furthermore, if either our lattice* $(\mathsf{IL},\sqsubseteq)$ *is finite, or if* P *contains no lattice functions (i.e.* $\mathcal{F}_{\mathsf{IL}} = \emptyset$*), then* $T_{P,\mathcal{D}}$ *is compact, and hence,its least fixpoint* $F3_{P,\mathcal{D}}$ *may be reached in* ω *steps.*

Just as the operator $T_{P,\mathcal{D}}$ and its least fixpoint were used to characterize the set $\mathsf{intr}_{P.\mathcal{D}}$, we may use the extended operator, $T3_{P,\mathcal{D}}$'s least fixpoint to characterize the interestingness interpretation $\mathsf{intr}^3_{P,\mathcal{D}}$.

Proposition 3.3 *The interesting interpretation* $\mathsf{intr}^3_{P,\mathcal{D}}$ *coincides with* $F3_{P,\mathcal{D}}$*.*

We may likewise extend the concept of *reduct* of a basic query interestingness program to an analogous notion for FIPs as follows. Note that the definition of reduct provided earlier (Definition 2.3) does not take the interestingness atoms into account (i.e. it only considers guards), and hence, we can apply the same definition of reduct given earlier to FIPs as well. If P is a FIP, then as before, we use the notation $P3_{\mathcal{D}}$ to denote the FIP obtained by applying the transformation in Definition 2.3 to P.

It is now easy to see that we may define an operator, $T^{P3}_{\mathcal{D}}$ which maps interestingness interpretations to interestingness interpretations, as follows: $T^{P3}_{\mathcal{D}}(\mathsf{II})(\mathsf{intr3}(q,u,t)) = \sqcup\{\xi_0 \mid$

There is a clause $C = \mathsf{intr3}(q_0, u_0, t_0) : \xi_0 \leftarrow \mathsf{intr3}(q_1, u_1, t_1) : \xi_1 \,\&\, \ldots \,\&\, \mathsf{intr3}(q_n, u_n, t_n) : \xi_n$ in $P3_{\mathcal{D}}$ such that for all $1 \leq i \leq n$. there exists a query $q'_i \sim q_i$ such that $\xi_i \sqsubseteq \mathsf{intr3}(q'_i, u_i, t_i)\}$.

We then have the following result:

Proposition 3.4 *Let P be an FIP, and $\mathcal{D}$ a database. Then $T_{\mathcal{D}}^{P3}$ is a monotonic operator. Thus $T_{\mathcal{D}}^{P3}$ possesses a least fixpoint $G3_{P,\mathcal{D}}$. Furthermore, if either our lattice $(\mathsf{IL}, \sqsubseteq)$ is finite, or if P contains no lattice functions (i.e. $\mathcal{F}_{\mathsf{IL}} = \emptyset$), then $T_{\mathcal{D}}^{P3}$ is compact, and hence,its least fixpoint $G3_{P,\mathcal{D}}$ may be reached in ω steps.*

The fixpoint $G3_{P,\mathcal{D}}$ provides another characterization of $\mathsf{intr}^3_{P,\mathcal{D}}$.

Proposition 3.5 *The interestingness interpretation $\mathsf{intr}^3_{P,\mathcal{D}}$ coincides with $\{\mathsf{intr}(q) : q \in G3_{P,\mathcal{D}}\}$.*

3.4 A Model-Theoretic Characterization of FIPs

In the preceding sections, we have already provided some alternative definitions of the semantics of FIPs. In this section, we provide a model-theoretic definition of the semantics of a FIP.

Definition 3.10 *Suppose II is an interestingness interpretation. We define a* satisfaction relation*, denoted $\models$, between interestingness interpretations and annotated structures as follows:*

- $\mathsf{II} \models \mathsf{intr3}(q, \nu, \tau) : \xi$ *where* $\mathsf{intr3}(q, \nu, \tau) : \xi$ *is ground iff* $\xi \sqsubseteq \mathsf{II}(\mathsf{intr3}(q, \nu, tau)$.
- $\mathsf{II} \models (F \,\&\, G)$ *iff* $\mathsf{II} \models F$ *and* $\mathsf{II} \models G$.
- $\mathsf{II} \models (F \vee G)$ *iff* $\mathsf{II} \models F$ *or* $\mathsf{II} \models G$.
- $\mathsf{II} \models (F \langle b, G)$ *iff either* $\mathsf{II} \models F$ *or the boolean query b is false or* $\mathsf{II} \not\models G$.
- $\mathsf{II} \models (\forall x)F$ *iff for every object a in the domain over which variable x ranges,* $\mathsf{II} \models F[x/a]$. *(Here $F[x/a]$ denotes the simultaneous replacement of all free occurrences of x in F by a.*
- $\mathsf{II} \models (\exists x)F$ *iff there is some object a in the domain over which variable x ranges such that* $\mathsf{II} \models F[x/a]$.

Given a set S of formulas of the above sort, we say that II satisfies S iff $\mathsf{II} \models \psi$ for all $\psi \in S$.

Definition 3.11 *Suppose P is a FIP, $\mathcal{D}$ is a database, q is a query in QL, u is a user, and t is a time point. Define the* level of interest, *$\mathsf{LI}(q, u, t)$ as follows:*

$$\begin{aligned}\mathsf{LI}(q,u,t) \;=\; & \sqcap\{\mathsf{II}(\mathsf{intr3}(q,u,t)) \mid \textit{there exists an interestingness interpretation } \mathsf{II} \\ & \textit{such that } \mathsf{II} \models \mathsf{FP}\}.\end{aligned}$$

The following important theorem tells us that all the concepts we have used to characterize the semantics of a FIP coincide.

Theorem 3.1 *Suppose P is a FIP, $\mathcal{D}$ is a database, q is a query in QL, u is a user, and t is a time point. Then the following quantities are all equal:*

- $\mathsf{LI}(q, u, t)$
- $\mathsf{intr}^3_{P,\mathcal{D}}(q, u, t)$
- $F3_{P,\mathcal{D}}(q, u, t)$
- $G3_{P,\mathcal{D}}(q, u, t)$

4 Full-Fledged Query Processing

In Section 2.4, we provided an initial attempt at query processing for propositional BIPs. In this section, we provide techniques to process queries to first order FIPs, i.e. the earlier restriction to *propositionality* is removed and the restriction to BIPs is also removed with FIPs being allowed. *However, throughout this section, we will assume that either our lattice* $(\mathsf{IL}, \sqsubseteq)$ *is finite, or that FIP P contains no lattice functions.* The rest of this section proceeds under this assumption.

This section will develop algorithms to answer the following questions:

(Elementary Query Processing) Given a query q, a user u, a point t in time, and an interestingness level ξ, check if the user is interested in query q with at least ξ level of interest.

(Interest Materialization) Given a user u, a point t in time, and an interestingness level ξ, find all queries q that user u is interested at time t with interest ξ or higher.

In the rest of this section, we will develop algorithms to solve each of the above problems.

4.1 Elementary Query Processing

Suppose we have a scenario where there are multiple agents, and agent A has just determined (for whatever reason) that it is going to evaluate query q. It wants to know which other agents have a "significant" interest in query q. To determine this, it sets a significance level by selecting a lattice value ξ from the interestingness lattice $(\mathsf{IL}, \sqsubseteq)$. When considering a single user u, and time t (now), it then evaluates the query $\mathsf{intr3}(q, u, t) : \xi$ against the FIP P that specifies interests of different users/agents u.

To handle such queries, we introduce the concept of a *lattice constraint*. Given an interestingness lattice $(\mathsf{IL}, \sqsubseteq)$, as well as accompanying notion of $(\mathsf{IL}, \sqsubseteq)$-terms introduced earlier, $(\mathsf{IL}, \sqsubseteq)$-constraints are defined inductively as follows:

- If ξ_1, ξ_2 are $(\mathsf{IL}, \sqsubseteq)$-terms, then $\xi_1 \sqsubseteq \xi_2$ is an $(\mathsf{IL}, \sqsubseteq)$-constraint.
- If $\mathcal{I}_1, \mathcal{I}_2$ are $(\mathsf{IL}, \sqsubseteq)$-constraints, then so is $(\mathcal{I}_1 \,\&\, \mathcal{I}_2$.

Note that $(\mathsf{IL}, \sqsubseteq)$-constraints are *Boolean constraints.* This is significant, because in that case, the conjunction of a Boolean query $b \in BQL$ with a Boolean-constraint $\mathcal{I}$ is still a Boolean query (albeit one in an expanded version of BQL that allows lattice constraints). We will not go into the (straightforward) extension, $BQL(\mathsf{IL})$, of BQL to handle $(\mathsf{IL}, \sqsubseteq)$-constraints, as this is fairly obvious.

A *generalized elementary query* is of the form:

$$\leftarrow \quad b, \mathcal{I}, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m \tag{4}$$

where $\mathcal{I}$ is an lattice constraint. Similarly, a *generalized elementary clause* is of the form

$$\mathsf{intr3}(q'_0, \nu'_0, \tau'_0) : \xi'_0 \quad \leftarrow \quad b', \mathcal{I}', \mathsf{intr3}(q'_1, \nu'_1, \tau'_1) : \xi'_1, \ldots, \mathsf{intr3}(q'_n, \nu'_n, \tau'_n) : \xi'_n. \tag{5}$$

Note that as $b, \mathcal{I}$ is still a Boolean query, we may, without loss of generality, continue to think of generalized elementary clauses as full interestingness clauses over $BQL(\mathsf{IL})$ instead of BQL. Thus, all the semantical constructs of FIPs continue to apply to sets of generalized elementary clauses, and we will continue to use them with no loss of generality.

Generalized elementary queries may be processed in the standard way, using a form of resolution developed by Lu et.al. [22]. However, the procedure of Lu et. al. [22] does need to be modified in the same respect as the procedure described earlier in Section 2.4.

We say that the generalized elementary clause C having the form given in (5) above can be used to *expand* a generalized elementary query having the form shown in (4) iff there exists a maximally[1] general substitution θ and an $1 \leq i \leq m$ such that:

- $q'_0\theta \sim q_i\theta$ and
- $\nu'_0\theta = \nu_0\theta$ and
- $\tau'\theta = \tau\theta$ and
- the constraint $\xi_i\theta \sqsubseteq \xi_0\theta$ is solvable.

In general, the reader should note that the first condition above causes the unique-ness of "most general unifiers" to be destroyed, and there may be several "maximally" general substitutions satisfying the aforesaid conditions.

In this case, the *expansion* of a query of the form shown in (4) with a clause C having the form given in (5) above is given by:

$$(\quad \leftarrow \quad b, b', \mathcal{I}, \xi_i \sqsubseteq \xi'_0, \tag{6}$$

$$\mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_{i-1}, \nu_{i-1}, \tau_{i-1}) : \xi_{i-1}, \tag{7}$$

$$\mathsf{intr3}(q_{i+1}, \nu_{i+1}, \tau_{i+1}) : \xi_{i+1}, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m, \tag{8}$$

$$\mathsf{intr3}(q'_1, \nu'_1, \tau'_1) : \xi'_1, \ldots, \mathsf{intr3}(q'_n, \nu'_n, \tau'_n) : \xi'_n)\theta. \tag{9}$$

[1]A substitution θ satisfying the conditions given here is said to be maximal iff for any substitution γ satisfying these conditions, if γ is more general thanθ, then γ is a variant of θ.

A *simple FIP-refutation* of a generalized interestingness query Q is a sequence $(Q_1, C_1, \theta_1), \ldots, (Q_k, C_k, \theta_k)$ where:

- $Q_1 = Q$;
- for $i < k$, Q_{i+1} is the expansion of query Q_i with respect to clause C_i and maximally general unifier θ_i and
- Q_k contains no $\mathsf{intr3}$-atoms and
- Q_k's constraint part is solvable.

The following result tells us that if our lattice is a linear chain, then expansions guarantee completeness.

Theorem 4.1 *Suppose* $(\mathsf{IL}, \sqsubseteq)$ *is a linear chain. Suppose further that P is either free of lattice annotation functions or that* $(\mathsf{IL}, \sqsubseteq)$ *is finite. Then simple FIP-refutations are a sound and complete query processing procedure for generalized interestingness queries.*

Unfortunately, as is well known in annotated logic [15, 18, 3, 21], using just expansions is not adequate to ensure completeness of the resulting simple FIP-refutations when the interestingness lattice is not linear. This is because two or more derivations using the notion of expansion above may lead to different levels of interest, and the least upper bound of these levels of interest may exceed both of them. Thus, levels of interest from different derivations may need to be combined. There is a standard way to solve this problem [15, 18, 3, 21] which leads to a second rule of inference called *reduction.* Reductions may be applied in full interestingness programs in the following way.

Suppose we consider two generalized interestingness clause C and C' respectively, given below:

$$\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0 \quad \leftarrow \quad b, \mathcal{I}, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_n, \nu_n, \tau_n) : \xi_n. \tag{10}$$

$$\mathsf{intr3}(q_0', \nu_0', \tau_0') : \xi_0' \quad \leftarrow \quad b', \mathcal{I}', \mathsf{intr3}(q_1', \nu_1', \tau_1') : \xi_1', \ldots, \mathsf{intr3}(q_m', \nu_m', \tau_m') : \xi_m'. \tag{11}$$

Clauses C, C' are said to be *mergeable* iff there exists a maximally general substitution θ such that:

- $q_0\theta \sim q_0'\theta$ and
- $\nu_0\theta = \nu_0'\theta$ and
- $\tau_0\theta = \tau_0'\theta$ and
- $(b, b', \mathcal{I}, \mathcal{I}')\theta$ is solvable.

In this case, the *merge* of C, C' is the clause:

$$\begin{aligned} (\mathsf{intr3}(q_0, \nu_0, \tau_0) : \sqcup(\xi_0, \xi'_0) \;\leftarrow\;& b, \mathcal{I}, b', \mathcal{I}' && (12)\\ & \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_n, \nu_n, \tau_n) : \xi_n && (13)\\ & \mathsf{intr3}(q'_1, \nu'_1, \tau'_1) : \xi'_1, \ldots, \mathsf{intr3}(q'_m, \nu'_m, \tau'_m) : \xi'_m)\theta. && (14) \end{aligned}$$

A *Full FIP-refutation* of a generalized interestingness query Q may now be defined as a sequence

$$(E_1, C_1, \theta_1), \ldots, (E_k, C_k, \theta_k)$$

where:

1. $E_1 = Q$ and
2. whenever $E_i (i \leq k)$ is a generalized interestingness query, then there exists a j $(j < i)$, such that E_j is a generalized interestingness query and E_i is the expansion of E_j with generalized interestingness clause C_j via maximally general substitution θ_j and
3. whenever $E_i (i \leq k)$ is a generalized interestingness clause, then E_i is the result of merging two clauses in $P \cup \{C_1, \ldots, C_{i-1}\}$ and
4. E_k is a query containing no intr3-atoms and
5. E_k's guard component is solvable.

Full FIP refutations guarantee soundness and completeness even when the interestingness lattice, $(\mathsf{IL}, \sqsubseteq)$, is not a chain. However, this completeness comes at a price – full FIP-refutations are less efficient than simple FIP-refutations as they must make use of two operations – merging and expansion – to guarantee completeness.

Theorem 4.2 *Suppose P is either free of lattice annotation functions or that $(\mathsf{IL}, \sqsubseteq)$ is finite. Then simple FIP-refutations are a sound and complete query processing procedure for generalized interestingness queries.*

4.2 Interest Materialization

Interest materialization deals with the following problem: we have a user u, we know the current time t, and we have set an interestingness threshold ξ, and we want to know what are all the queries that user u is interested in with over ξ degree of interest. In this section, we will show how interest materialization may be efficiently processed.

Definition 4.1 *Suppose* $\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0$ *is an interestingness* intr3 *atom, u is a user, t is a time point, and ξ is an interestingness level.* $\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0$ *is said to be* compatible *with (u, t, ξ) iff the following conditions hold:*

1. The pair (ν_0, τ_0) and the pair (u, t) is unifiable via some most general unifier θ and

2. The lattice constraint, $\xi\theta \sqsubseteq \xi_0\theta$, is solvable, i.e. there exists an assignment to the variables in $\xi\theta, \xi_0\theta$ that makes the above relation true.

For example, consider the FIP described earlier (and reproduced below for convenience):

$$\begin{array}{rcl} \mathsf{intr3}(q_1, mr\_auditor, T) : 0.5 & \leftarrow & q_1. \\ \mathsf{intr3}(q_4, mr\_auditor, T) : 0.7 & \leftarrow & q_3. \\ \mathsf{intr3}(q_4, mr\_auditor, T) : min(1, 0.7 + \dfrac{V}{2}) & \leftarrow & q_3, \mathsf{intr3}(q_1, mr\_auditor, T) : V. \end{array}$$

Suppose we take $u =$Mr. Auditor, $t = 5$ and $\xi = 0.6$. Then the head of the second full interesting clause in the above set of rules is compatible with (u, t, ξ). The first is not because the annotated atom, $\mathsf{intr3}(q_1, mr\_auditor, T) : 0.5$ has an annotation, viz. 0.5 which is not greater than 0.6. The head, $\mathsf{intr3}(q_4, mr\_auditor, T) : min(1, 0.7 + \frac{V}{2})$ of the third rule is also compatible with (u, t, ξ) because in this case, $\theta = \{T = 5\}$, and the constraint

$$0.6 \sqsubseteq min(1, 0.7 + \frac{V}{2})$$

is certainly solvable (e.g. take $V = 1$).

Note that the definition of compatibility between an $\mathsf{intr3}$-atom, $A : \xi_0$, and a triple (u, t, ξ) only verifies, intuitively, whether a clause with $A : \xi_0$ in the head has a chance of contributing to knowledge about whether user u is interested in some query (to be determined) at time t with interest level ξ or more. Compatibility does not guarantee that user u is in fact interested in some query at time t with interest level ξ or more. The following result guarantees that compatibility is easy to check.

Theorem 4.3 *Given an interestingness atom $\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0$, and a triple (u, t, ξ), determining if $\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0$ is compatible with (u, t, ξ) can be solved in linear time (assuming existence of an oracle for checking satisfiability of a constraint over the lattice $(\mathsf{IL}, \sqsubseteq)$).*

Note that this theorem tells us clearly that choosing an interestingness lattice and annotation functions that guarantee "easy" solvability of constraints is critical to the success of compatibility tests. The more complicated the annotation functions one picks, the harder it is to establish solvability of $(\mathsf{IL}, \sqsubseteq)$-constraints, thus making it harder to establish compatibility. As the following definition shows, compatibility testing is a critical part of computing the set of queries a user is interested in at time t with at least level ξ of interest. Compatibility enables us to prune the search space efficiently.

One way to compute the set of all queries that user u is interested in at time t with at least level ξ of interest (with respect to FIP P) is the naive algorithm shown in Figure 5. This algorithm is naive because it computes the entire least fixpoint, $F3_{P,\mathcal{D}}$ of the operator $T3_{P,\mathcal{D}}$, which includes consideration of many rules (or instances of rules) in P that do not contribute to information about user u's interests at time t with level ξ.

Naive Algorithm for Interest Materialization.
Input: A finite interestingness program P, a database $\mathcal{D}$, and a user,time, interest level triple (u, t, ξ).
Output: The set of all queries q such that $\xi \sqsubseteq F3_{P,\mathcal{D}}\mathsf{intr3}(q, u, t)$, i.e. the set of all queries q such that $P \models \mathsf{intr3}(q, u, t)$.

```
compute $F3_{P,\mathcal{D}}$;
for each atom $\mathsf{intr3}(q', u', t)$ do
    if $\xi \sqsubseteq F3_{P,\mathcal{D}}(\mathsf{intr3}(q', u', t))$ then return $\mathsf{intr3}(q', u', t)$
    endif
endfor
```

Figure 5: Naive Algorithm for Interest Materialization

The operator $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}$ below "specializes" the $T3_{P,\mathcal{D}}$ operator, by taking into account, the triple (u, t, ξ). In other words, this specializes the fixpoint operator to "focus" on the queries that user u is interested in at time t with at least ξ level of interest.

Definition 4.2 *The operator $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}$ maps FIPs to FIPs, and is defined as the smallest set of full interestingness clauses that satisfy the following conditions:*

1. *Suppose P' is an FIP and*

$$\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0 \quad \leftarrow \quad b, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m \tag{15}$$

   is a clause in P'. Suppose $\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0$ is compatible with (u, t, ξ) via mgu θ and

$$\leftarrow \quad (b, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m)\theta \tag{16}$$

   is non-empty. Then:

$$(\mathsf{intr3}(q_0, \nu_0, \tau_0) : \xi_0 \quad \leftarrow \quad b, \xi \sqsubseteq \xi_0, \mathsf{intr3}(q_1, \nu_1, \tau_1) : \xi_1, \ldots, \mathsf{intr3}(q_m, \nu_m, \tau_m) : \xi_m)\theta \tag{17}$$

   is in $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}(P')$.

2. *If C is a full interestingness clause in $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}(P')$ of the form (15), and if there exists a clause C' in P' of the form $\mathsf{intr3}(q', u', t') : \xi' \leftarrow b', Body'$ and is such that $\mathsf{intr3}(q', u', t') : \xi'$ is compatible (via mgu θ_i) with (ν_i, τ_i, ξ_i) for some $1 \leq i \leq m$, then $(\mathsf{intr3}(q', u', t') : \xi' \leftarrow b', \mathcal{I}', Body')\theta_i$ is in $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}(P')$ where $\mathcal{I}'$ is the constraint $\xi_i \sqsubseteq \xi'$.*

3. *Nothing else is in $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}(P')$.*

The operator $\mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}$ may easily be computed by the algorithm shown in Figure 6 below.

Algorithm to Compute $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$.
Input: a finite interestingness program P, a database $\mathcal{D}$, a user u, a time t, and a interestingness level ξ.
Output: $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$.

```
$change = true$;
$Todo$ = { all rules in $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$ due to condition (1) in the definition of $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$};
$Done = \{\mathsf{intr3}(-,u,t) : \xi\}$;
$Todo = \emptyset$;
    while $change$ do
      { for each rule $r \in (Todo)$ do
      $NewDone$ = List of all intr3-atoms in the body of some rule in $P$ that are
      not subsumed by an atom in $Done$;
      $Todo = Todo \cup$ { all rules with heads compatible with an atom in $NewDone$};
      if $NewDone = Done$ then $change = false$;
      }
    end-while
    Return $Todo$
```

Figure 6: Algorithm for Computing $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$

Intuitively, the iterative application of the loop shown in Figure 6 causes the initial FIP P to be "fine-tuned" so that it only focuses on rules that have some reasonable chance of contributing to the derivation of atoms of the form $\mathsf{intr3}(q,u,t) : \xi'$ where $\xi \sqsubseteq \xi'$. The following result shows that the operator $\mathcal{G}_{P,\mathcal{D}}$ modifies the FIP P in such a way that the set $\{q | \mathcal{G}_{P,\mathcal{D}}^{u,t,\xi} \models \mathsf{intr3}(q,u,t) : \xi\}$ *coincides* with the set of all queries that $F3_{P,\mathcal{D}}$ says user u is interested in (with at least level ξ at time t). In other words, "whittling" down the FIP P by the iterative application of the loop shown in Figure 6 operator leads to no loss of information about the queries that user u is interested in (with at least level ξ at time t).

Theorem 4.4 *The following expressions coincide:*

1. *The set of all* $\mathsf{intr3}$*-atoms of the form* $\mathsf{intr3}(\star,u,t)$ *such that* $\xi \sqsubseteq F3_{P,\mathcal{D}}(\mathsf{intr3}(\star,u,t))$ *that are entailed by* $\mathcal{G}_{P,\mathcal{D}}^{u,t,\xi}$.
2. *The set of all* $\mathsf{intr3}$*-atoms in* $F3_{P,\mathcal{D}}$ *of the form* $\mathsf{intr3}(\star,u,t)$ *such that* $\xi \sqsubseteq F3_{P,\mathcal{D}}(\mathsf{intr3}(\star,u,t))$.

The above theorem allows us to produce the following improved algorithm (see Figure 7) for computing interestingness. The algorithm is an improvement on the naive algorithm because the algorithm only focuses on a "whittled down" version, Q, of the original program P, when computing the fixpoint $F3$.

The "improved" algorithm above can now be further improved by computing $F3_{P,\mathcal{D}}$ not as the fixpoint of the operator $T3_{P,\mathcal{D}}$, but by a differential computing method. In differential

Improved Algorithm for Interest Materialization.
Input: A finite interestingness program P, a database $\mathcal{D}$, and a user,time, interest level triple (u, t, ξ).
Output: The set of all queries q such that $\xi \sqsubseteq F3_{P,\mathcal{D}}(\mathsf{intr3}(q, u, t))$, i.e. the set of all queries q such that $P \models \mathsf{intr3}(q, u, t) : \xi$.

```
    compute $Q = \mathcal{G}^{u,t,\xi}_{P,\mathcal{D}}$.
    compute $F3_{Q,\mathcal{D}}$.
    for each atom $\mathsf{intr3}(q', u', t')$ do
        if $u' = u$ and $t' = t$ and $\xi \sqsubseteq F3_{Q,\mathcal{D}}(\mathsf{intr3}(q', u', t'))$ then return $\mathsf{intr3}(q', u', t)$
        endif
    endfor
```

Figure 7: Improved Algorithm for Interest Materialization

computations of an iterative, monotonic procedure, we try to use the "difference" between successive steps in the computation to speed up the computation. In the case of computing the least fixpoint, $F3_{Q,\mathcal{D}}$ in the algorithm of Figure 7, this may be done as follows. The standard way of computing $F3_{Q,\mathcal{D}}$ is to iteratively apply the operator $T3_{P,\mathcal{D}}$. However, we may wish to write $T3_{P,\mathcal{D}}$ as the following equation:

$$T3_{P,\mathcal{D}}(\mathsf{II})(A) \quad = \quad \mathsf{II}(A) \sqcup \partial T3_{P,\mathcal{D}}(\mathsf{II})(A) \tag{18}$$

In this equation, we would like $\partial T3_{P,\mathcal{D}}$ to be a new operator that returns only the "changes" to II that occur as we apply $T3_{P,\mathcal{D}}$ to it.

Definition 4.3 *The operator $\partial T3_{P,\mathcal{D}}$ maps INT^3 to INT^3 and is defined as follows. Given an interestingness interpretation II, and an interestingness atom $\mathsf{intr3}(q, u, t)$: $\partial T3_{P,\mathcal{D}}(\mathsf{II})(\mathsf{intr3}(q, u, t)) = \sqcup\{\xi_0 \mid$ There is a clause $C = \mathsf{intr3}(q_0, u_0, t_0) : \xi_0 \leftarrow b, \mathsf{intr3}(q_1, u_1, t_1) : \xi_1 \,\&\ldots\&\, \mathsf{intr3}(q_n, u_n, t_n) : \xi_n$ in P such that for all $1 \leq i \leq n$. there exists a query $q'_i \sim q_i$ such that $\xi_i \sqsubseteq \mathsf{intr3}(q'_i, u_i, t_i)$ and b is true in $\mathcal{D}\}$ and $\underline{\mathsf{II}(\mathsf{intr3}(q, u, t)) \not\sqsubseteq \mu_0\}}$.*

The underlined portion of the above definition specifies why the operator $\partial T3_{P,\mathcal{D}}$ is different from the operator $T3_{P,\mathcal{D}}$. It says that the operator $\partial T3_{P,\mathcal{D}}$ only "focuses" on clauses that are not already dependent on its input.

The following result is now immediate.

Proposition 4.1 *Equality 18 holds for the above definition of $\partial T3_{P,\mathcal{D}}$.*

Based on this property, we may now create an improved differential algorithm to compute $F3_{P,\mathcal{D}}$. The new differential algorithm is shown in Figure 8.

Algorithm to compute $F3_{P,\mathcal{D}}$. *Input:* a finite interestingness program P, a database $\mathcal{D}$. *Output:* $F3_{P,\mathcal{D}}$.

```
$\mathsf{II} = \lambda A.\bot$. (i.e. $\mathsf{II}$ assigns $\bot$ to all atoms)
$change = true$;
    while $changed$ do
    {
      $\mathsf{II}_{new} = \mathsf{II} \sqcup \partial T3_{P,\mathcal{D}}(\mathsf{II})$;
      if $\mathsf{II}_{new} = \mathsf{II}$ then $changed = false$
      else $\mathsf{II} = \mathsf{II}_{new}$;
    };
Return                                                        $\mathsf{II}$;
```

Figure 8: Differential algorithm for computing $F3_{P,\mathcal{D}}$

5 Related Work

Though there has been extensive work on data mining[5, 12, 13], as well as on intelligent agents and profiling tools[10, 11, 17], there has been almost no work to date on what "interestingness" means, and/or what a theory of interestingness should look like. In this paper, we have made a first attempt to answer this question by proposing the use of a very tightly restricted fragment of logic to identify interesting queries, and shown that our framework can handle the fact that in reality, different users have different levels of interest in different things at different points in time, depending upon the circumstances involved. In other words, our notion of interesting is expressive enough to address most real-life application needs, yet is "tight" enough that it is amenable to efficient implementations (such an implementation is currently ongoing, using the HERMES Heterogeneous Reasoning and Mediator System [28, 1, 20] as the query language QL).

The initial framework of BIPs proposed in this paper is a very tightly syntactic restricted fragment of constraint logic programs[14]. However, the readers must note the following points: first, unlike CLP where truth/falsity in a domain is fixed (e.g. the truth of constraints over the reals doesn't change!), in our case, the truth/falsity of constraints changes with time because databases change with time, and hence Boolean queries over databases potentially return different answers over time. In addition, because different queries may be equivalent over a given database $\mathcal{D}$, but not over other databases $\mathcal{D}'$, our theory must take $\sim$-equivalences of queries into account – something CLP does not need to do. This makes our treatment of fixpoints and query processing procedures somewhat different from those for CLPs. In particular, though BIPs are a *syntactic* fragment of CLPs, their *semantics* is different from the standard CLP-semantics for this syntactic fragment.

The final framework of FIPs proposed in this paper similarly reflects a tightly reflected syntactic fragment of the Hybrid Knowledge Base Paradigm[22]. Again, things are complicated by the fact that different queries may be equivalent over a given database $\mathcal{D}$, but not over other databases $\mathcal{D}'$, causing the semantics of FIPs to be different from the semantics of HKBs in this fragment.

6 Conclusions and Future Work

There is now a vast variety of applications that informally use the intrinsic concept of "interestingness." For example:

- Data mining systems attempt to identify "interesting" patterns;
- Computer security profiling tools attempt to identify "interesting" usage patterns in computer logs and audit files;
- Marketing systems attempt to identify people whose "interests" relate to the product(s) being marketed;
- Intelligent agents monitoring changing data sources (e.g. e-mail, news) attempt to filter out "uninteresting" data.

All these applications use some innate concept of *interestingness* – yet, none of these applications answer the question: *what is interestingness? How can interestingness be efficiently represented and manipulated computationally?*

In this paper, we have argued that:

- Interests vary with users;
- Interests vary over time;
- Interests are relative – users may have a "greater" interest in some things as opposed to others;
- Interests depend upon what is true in the world or application domain.

As a consequence, any attempt to define a *rigid* concept of interestingness is doomed to failure (in our opinion). Rather, a framework is needed within which interestingness may be defined either by an application developer or by a user.

Based on these observations, we started by proposing the notion of a *Basic Interestingness Program* that merely takes the last item into account and provided BIPs with a formal syntax and semantics. Later, we generalized this to *Full Interestingness Programs* (FIPs) that capture all the above desiderata. In addition, we provided FIPs with a formal syntax and semantics, as well as with efficient computation techniques.

One can naturally extend interestingness programs to take into account other intentional properties of queries such as awareness or importantness. We argue that this and other important properties of queries may be treated similarly to the way we handle interestingness.

We can add negation, as often done in logic programming, allowing for derivation of interestingness of queries of the lack of interestingness of other queries. This yields a natural extension of our formalism incorporating the notions such as completion and stability into our approach. This will be studied in the subsequent papers.

We are currently building *Interestingness Servers* on top of the HERMES Heterogeneous Reasoning and Mediator System. HERMES uses a logical query language (which is used in our implementation for QL) which accesses not just a wide variety of databases (Ingres, Dbase, Paradox, ObjectStore), but also a wide variety of data structures (flat files, image data, geographic quadtree data, video data, face databases), as well as a variety of software packages (such as operations research software, a US Army terrain reasoning and route planning system, a nonlinear planner, etc.). By picking an implemented, highly expressive query language that can access heterogeneous and distributed data sources, we hope to maximize the impact of our Interestingness Servers.

Acknowledgments

This work was supported by the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, by ARPA/Rome Labs contract F30602-93-C-0241 (ARPA Order Nr. A716), by Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002 Federated Laboratory ATIRP Consortium, by an NSF Young Investigator award IRI-93-57756, and by NSF grant IRI-9619233.

References

[1] S. Adali, K.S. Candan, Y. Papakonstantinou and V.S.Subrahmanian. *Query Caching and Optimizing in Distributed Mediator Systems*, Proc. 1996 ACM SIGMOD Conf. on Management of Data, pages 137–148, 1996.

[2] S. Adali and R. Emery. *A Uniform Framework For Integrating Knowledge In Heterogeneous Knowledge Systems*, Proc. of the Eleventh International Conference on Data Engineering, 1994.

[3] S. Adali and V.S. Subrahmanian. *Amalgamating Knowledge Bases, III: Algorithms, Data Structures and Query Processing.* Journal of Logic Programming, 28:57-100, July 1996.

[4] K. Apt. *Logic programming.* In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, pages 493–574. MIT Press, Cambridge, MA, 1990.

[5] I. Bhandari, E. Colet, J. Parker, Z. Pines, R. Pratap and K. Ramanujam *Advanced Scout: Data Mining and Knowledge Discovery in NBA Data*, Data Mining and Knowledge Discovery, 1, 1997.

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

[7] B.F. Chellas. *Modal logic, an introduction.* Cambridge University Press, 1980.

[8] W.F. Dowling and J.H. Gallier. *Linear-time algorithms for testing the satisfiability of propositional Horn formulae.* Journal of Logic Programming, 3:267–284, 1984.

[9] T. Eiter, J. Lu, and V.S. Subrahmanian. *Computing non-ground representation of stable models.* Logic Programming and Nonmonotonic Reasoning, Springer Lecture Notes in CS 1265. pages 198-217, 1997.

[10] O. Etzioni and D. Weld. *A Softbot-Based Interface to the Internet*, Communications of the ACM,37:72–76, 1994.

[11] M.R. Genesereth and S.P. Ketchpel. *Software Agents*, Communications of the ACM,37:49–53. 1994.

[12] J. Han, Y. Fu, W. Wang, K. Koperski, O. Zaiane . *DMQL: A data mining query language for relational databases*, SIGMOD 1996 Workshop on Data Mining and Knowledge Discovery. 1996.

[13] T. Imielinski. *From file mining to database mining*, SIGMOD 1996 Workshop on Data Mining and Knowledge Discovery. 1996.

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

[15] M. Kifer and V.S. Subrahmanian. *Theory of Generalized Annotated Logic Programming and its Applications*, Journal of Logic Programming, 12, 4, pages 335–368, 1992.

[16] H. Korth and A. Silberschatz. *Database System Concepts*, McGraw Hill. 1986.

[17] Y. Lashkari, M. Metral and P. Maes. *Collaborative Interface Agents*, Proc. AAAI 1994.

[18] S. Leach and J. Lu. *Computing Annotated Logic Programs*, Proceedings of the 11th International Conference on Logic Programming (ed. P. Van Hentenryck), MIT Press, pages 257-271. 1994.

[19] J. Lloyd. *Foundations of logic programming.* Berlin: Springer-Verlag, 1984.

[20] J. Lu, G. Moerkotte, J. Schue, and V.S. Subrahmanian. *Efficient Maintenance of Materialized Mediated Views*, in: Proc. 1995 ACM SIGMOD Conf. on Management of Data, San Jose, CA, May 1995.

[21] J. Lu, N. Murray and E. Rosenthal. *Signed Formulas and Annotated Logics*, Proceedings of the International Symposium on Multiple-Valued Logic, pages 48-53. IEEE Computer Society Press, 1993,

[22] J. Lu, A. Nerode and V.S. Subrahmanian. Hybrid Knowledge Bases, *IEEE Transactions on Knowledge and Data Engineering*, 8:773–785, Oct. 1996.

[23] R. Ng and V.S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110:42–83, 1994.

[24] E. Oomoto and K. Tanaka. *OVID: Design and Implementation of a Video-Object Database System*, IEEE Trans. on Knowledge and Data Engineering, 5:629–643, 1993.

[25] N. Roussopoulos, C. Faloutsos and T. Sellis. *An Efficient Pictorial Database System for PSQL*, IEEE Transactions on Software Engineering, 14:639–650, 1988.

[26] R.T. Snodgrass (ed). *The TSQL2 Temporal Query Language*, Kluwer Academic Press, 1995.

[27] V.S. Subrahmanian. *Computational Reasoning with Nonclassical and Paraconsistent Logics.* PhD thesis, University of Syracuse, 1989.

[28] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J.J. Lu, A. Rajput, T.J. Rogers, R. Ross and C. Ward. *HERMES: A Heterogeneous Reasoning and Mediator System User Manual*, draft, 1997.