Constraint objects

Divesh Srivastava  
*AT&T Labs, divesh@research.att.com*

Raghu Ramakrishnan  
*Microsoft*

Peter Revesz  
*University of Nebraska-Lincoln, prevesz1@unl.edu*

Follow this and additional works at: [http://digitalcommons.unl.edu/cseconfwork](http://digitalcommons.unl.edu/cseconfwork)

Part of the [Computer Engineering Commons](http://digitalcommons.unl.edu/computerengineering), [Electrical and Computer Engineering Commons](http://digitalcommons.unl.edu/electricalandcomputerengineering), and the [Other Computer Sciences Commons](http://digitalcommons.unl.edu/othercomputersciences)

[http://digitalcommons.unl.edu/cseconfwork/322](http://digitalcommons.unl.edu/cseconfwork/322)

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
Constraint Objects

Divesh Srivastava\textsuperscript{1}, Raghu Ramakrishnan\textsuperscript{2}, Peter Z. Revesz\textsuperscript{3}

\textsuperscript{1} AT&T Bell Laboratories, Murray Hill, NJ 07974, USA
\textsuperscript{2} University of Wisconsin, Madison, WI 53706, USA
\textsuperscript{3} University of Nebraska, Lincoln, NE 68588, USA

Abstract. We describe the Constraint Object Data Model (CODM), which enhances an object-based data model with existential constraints to naturally represent partially specified information. We present the Constraint Object Query Language (COQL), a declarative, rule-based language that can be used to infer relationships about and monotonically refine information represented in the CODM. COQL has a model-theoretic and an equivalent fixpoint semantics, based on the notions of constraint entailment and "proofs in all possible worlds". We also provide a novel polynomial-time algorithm for quantifier elimination for set-order constraints, a restricted class of set constraints that uses $\in$ and $\subseteq$.

1 Introduction

Object-oriented database (OODB) systems will, most probably, have a significant role to play in the next generation of commercial database systems. While OODB systems have a sophisticated collection of features for data modeling, current-day OODB systems provide little or no support for representing and manipulating partially specified values.

For example, suppose that an OODB is used to represent knowledge about plays and playwrights. If Shakespeare's year of birth were known to be 1564, this could be represented easily in the database. However, historians do not have complete information about playwrights such as Shakespeare; they only have estimates of his date of birth and when he wrote his various plays. Partial information about Shakespeare’s year of birth can be naturally represented as a conjunction of constraints, $\text{Shakespeare.Year.of.birth} \geq 1560 \land \text{Shakespeare.Year.of.birth} \leq 1570$.

Occasionally these estimates are refined reflecting the results of new research. For example, suppose research determined that Shakespeare could have been born no later than 1565. Then the information about Shakespeare’s year of birth can be refined by conjoining the constraint, $\text{Shakespeare.Year.of.birth} \leq 1565$, to the previous conjunction of constraints.

Combining constraints with the notion of objects with unique identifiers offers a powerful mechanism for constraint specification and refinement. Our technical contributions in this paper are as follows:

1. We describe how an object-based data model can be enhanced with (existential) constraints to naturally represent partially specified information (Section 2). We refer to this as the Constraint Object Data Model (CODM).
2. We present a declarative, rule-based, language that can be used to manipulate information represented in the CODM. We refer to this as the Constraint Object Query Language (COQL) (Section 3). COQL has a model-theoretic and an equivalent fixpoint semantics, based on the notions of constraint entailment and "proofs in all possible worlds". One of the novel features of COQL is the notion of monotonic refinement of partial information in object-based databases.

3. We present a novel polynomial-time algorithm for quantifier elimination for a restricted class of set constraints that uses $\in$ and $\subseteq$ (Section 4). We refer to this class as set-order constraints. The quantifier elimination algorithm can also be used to check satisfiability and entailment of conjunctions of set-order constraints in polynomial time.

Both the constraint object data model and the constraint object query language are easily extended to compactly represent sets of fully specified values using universal constraints, and manipulate such values using a declarative, rule-based language, following the approach of [5, 7]. Indeed, it appears that both existential and universal constraints are instances of a more general paradigm—in essence, an underlying set of values is defined intensionally, and a number of different interpretations arise by considering various operations on this set. For reasons of space, we do not pursue this further in the paper.

Integration of constraints with objects has also been considered by Freeman-Benson and Borning ([2, 3]). Their work differs from ours, since their languages, Kaleidoscope’90 and Kaleidoscope’91, are imperative languages. We are interested in the incorporation of constraints into objects in a more declarative setting.

This paper is based on work in progress, and the various ideas are motivated primarily through examples.

2 Constraint Object Data Model

The Constraint Object Data Model (CODM) allows facts (tuples) as well as objects, which are essentially facts with unique identifiers [1]. The novel feature is that certain attribute values can be "don’t know" nulls whose possible values are specified using constraints. We refer to attributes that can take on such values as E-attributes. Relations and classes are collections of facts and objects, respectively.\(^4\)

Conceptually, all the constraints on E-attributes are maintained globally. This allows for specification of, e.g., inter-object constraints, which are very useful in many situations. However, in many of the examples discussed in the paper it suffices to associate constraints with the objects whose E-attributes they constrain; when possible, we depict the constraints in this fashion for ease of understanding.

\(^4\) Classes can be organized into an inheritance hierarchy; however, this is orthogonal to our discussion, and we do not deal with inheritance in this paper.
Example 1 (Playwrights and Plays). There are two classes of objects in the database: playwrights and plays. Partial information is represented about the year of composition of the plays, the writers of the plays, and the year of birth of the playwrights.

<table>
<thead>
<tr>
<th>playwrights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Od</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>oid1</td>
</tr>
<tr>
<td>oid2</td>
</tr>
<tr>
<td>oid3</td>
</tr>
</tbody>
</table>

The constraints are existential constraints in that the value of the E-attribute is some unique value from the domain satisfying these constraints. Note that there is no information on Fletcher’s year of birth, which is equivalent to stating that Fletcher could have been born in any year.

<table>
<thead>
<tr>
<th>plays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Od</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>oid10</td>
</tr>
<tr>
<td>oid11</td>
</tr>
</tbody>
</table>
| oid12| Henry VIII | S1 | Y12 | Y12 \leq 1613 ∧ Y12 \geq 1608∧ 
oid2 ∈ S1 ∧ S1 ⊆ { oid1, oid2 } |
| oid13| Meghdoot | { oid3 } | Y13 | Y13 \leq 1050 |

The form of the constraints allowed depends on the types of the E-attributes. The Year of birth and the Year of composition E-attributes are of type integer, and hence they are constrained using arithmetic constraints over integers. Similarly, the Writers E-attribute of plays is of type set of playwrights and it is constrained using set constraints \( \subseteq, \in \). For example, the constraint on the Writers attribute of oid12 indicates that either Fletcher is the sole writer of Henry VIII, or Fletcher and Shakespeare are joint writers of that play; this represents partial information on the set of playwrights.

We note that the Constraint Object Data Model only allows first-order constraints, i.e., the names and types of the attributes are fixed for each fact and object, and cannot be partially specified using constraints; only the values of these attributes can be partially specified using constraints.

3 Constraint Object Query Language

We present the declarative Constraint Object Query Language (COQL) that can be used to reason with facts and objects in the CODM. A COQL program is a collection of rules similar to Horn rules, where each rule has a body and a head.
The body of a rule is a conjunction of literals and constraints, and the head of the rule can be either a positive literal or a constraint. COQL allows arbitrary constraints, not just conjunctions of primitive constraints, to occur in the bodies and heads of program rules.

3.1 COQL: Inferring New Relationships

A COQL program can be used to infer new relationships, as facts, between existing objects and facts. For simplicity, we assume that COQL rules do not create new objects; this condition can be checked syntactically by having a safety requirement, that any object identifier appearing in the head of a COQL rule also appears in a body literal of that rule. Our results are orthogonal to proposals that permit the creation of new objects using rules (e.g., [6]), and can be combined with them in a clean fashion. We now present some example queries to motivate the inference of new relationships using COQL rules.

Example 2 (Selection). Consider the database of plays and playwrights from Example 1. Suppose we want to know the names of all playwrights born before the year 1700. The following rule expresses this query, using the dot notation for accessing object attributes:

q1 (P.Name) : − playwrights (P), P.Year_of_birth < 1700.

If the years of birth of all the playwrights in the database are completely specified, the meaning of this query is straightforward. In the presence of partial information about the years of birth of the playwrights, there are (at least) three possible semantics that can be used to assign meaning to this query.

1. Proof/Truth in at least one possible world.

Under this semantics, a playwright “satisfies” the query if at least one assignment of fully specified values to the Year_of_birth attribute of the playwright, consistent with the object constraints, satisfies the query. All three playwrights, Shakespeare, Fletcher and Kalidasa would be retrieved as answers to the query under this semantics. Shakespeare could have been born in 1564, Fletcher in 1600 and Kalidasa in 975; these values are consistent with the constraints on the object attributes.

To compute this answer set to the query, we need to check satisfiability of the conjunction of constraints present in the object and the constraints present in the query. For example, the conjunction of constraints oid3.Year_of_birth \leq 1000 \land oid3.Year_of_birth < 1700 (where oid3 is the identifier of the object representing Kalidasa) is satisfiable in the domain of integers.

2. Truth in all possible worlds.

Under this semantics, a playwright “satisfies” the query if every assignment of a fully specified value to the Year_of_birth attribute of the playwright, consistent with the object constraints, satisfies the query. Only Shakespeare and Kalidasa would be retrieved as answers to the query under this semantics. Fletcher could have been born in 1800; this value is consistent with the object constraints, while being inconsistent with the query constraints.
3. Proofs in *all possible worlds*.
Under this semantics, a playwright “satisfies” the query if: (1) every assignment of a fully specified value to the \texttt{Year.of.birth} attribute of the playwright, consistent with the object constraints, satisfies the query, and (2) the derivation trees corresponding to each of the possible assignments are “similar”. (The derivation trees constitute the “proofs”.) In this example, the answers to the query under this semantics are the same as under the “truth in all possible worlds” semantics.

To compute this answer set to the query, we need to check that the constraints present in the objects *entail* (i.e., imply) the query constraints. For example, the object constraints \( \texttt{oid1.Year.of.birth} \leq 1570 \land \texttt{oid1.Year.of.birth} \geq 1560 \) entails the (instantiated) query constraint \( \texttt{oid1.Year.of.birth} < 1700 \) (where \( \texttt{oid1} \) is the identifier of the object representing Shakespeare) in the domain of integers. However, the object constraints associated with Fletcher do not entail the (instantiated) query constraint \( \texttt{oid2.Year.of.birth} < 1700 \) (where \( \texttt{oid2} \) is the identifier of the object representing Fletcher) in the domain of integers.

The first two semantics are closely related to the semantics of Imielinski et al. [4] for OR-objects; we do not elaborate on these relationships in the paper for lack of space.

*Example 3 (Set Constraints).* Suppose we want to know the names of all the plays written by Shakespeare. The following rule expresses the query:

\[
\text{q3 (P.Name) : - plays (P), playwrights (W), W.Name = “Shakespeare”, P.Writers = S, W \in S.}
\]

Under the “proof/truth in at least one possible world” semantics, the play Henry VIII would be an answer (since Shakespeare could have written it together with Fletcher) as would Othello and Macbeth (since Shakespeare is known to have written these). The first answer can be obtained by checking the satisfiability of the conjunction of the object constraints \( \texttt{oid2 \in oid12.Writers} \land \texttt{oid12.Writers} \subseteq \{ \texttt{oid1, oid2} \} \) with the (instantiated) query constraint \( \texttt{oid1 \in oid12.Writers} \).

Under the “truth in all possible worlds” as well as “proofs in all possible worlds” semantics, however, Henry VIII would not be an answer. This is because the object constraints \( \texttt{oid2 \in oid12.Writers} \land \texttt{oid12.Writers} \subseteq \{ \texttt{oid1, oid2} \} \) do not entail the (instantiated) query constraint \( \texttt{oid1 \in oid12.Writers} \). Othello and Macbeth would be the only answers in this case.

### 3.2 COQL: Monotonically Refining Objects

COQL programs can also be used to *monotonically refine* objects, in response to additional information available about the objects. For example, suppose research determined that Shakespeare could have been born no later than 1565,
then the object Shakespeare can be refined by conjoining the constraint Shakespeare.Year_of_birth \leq 1565.

The notion of *declarative monotonic refinement* of partially specified objects is one of the novel contributions of this paper. Object refinement can be formalized in terms of a lattice structure describing the possible states of an object, with a given information theoretic ordering. The value \( \perp \) corresponds to having no information about the attribute values of the object, and \( \top \) corresponds to having inconsistent information about the object. (There are many different complete and consistent values for the object attributes; each of these is just below \( \top \) in this information lattice.) Object refinement now can be thought of as moving up this information lattice.

We give an example of declarative, rule-based, object attribute refinement next. The body of a refinement rule is similar to the body of a rule used to infer new relationships. The head of a refinement rule, on the other hand, is a constraint (not necessarily a conjunction of primitive constraints).

**Example 4 (Refining Attributes of Objects).** The following refinement rule expresses the intuition that a playwright cannot write a play before birth:

\[
W.\text{Year of birth} \leq P.\text{Year of composition} : \quad \neg \text{playwrights (W)},
\]

\[
\text{plays (P)}, \quad W \in P.\text{Writers}.
\]

The right hand side (body) of the rule is the condition, and the left hand side (head) is the action of the rule. If the body is satisfied, then the instantiated head constraint is conjoined to the (global) constraints on the E-attributes. (This is an example where the instantiated head constraint is an inter-object constraint, and hence cannot be associated with a single object.)

In the presence of partial information, we give a meaning to refinement rules based on the “proofs in all possible worlds” semantics. (In Example 6, we show that the “proof/truth in at least one possible world” is order dependent, and that the “truth in all possible worlds” semantics requires case-based reasoning, which is computationally intractable.) In this case, we would conjoin the constraint Fletcher.Year_of_birth \leq Henry VIII.Year_of_composition to the global collection of constraints. Conflicting refinements could, of course, result in an inconsistent constraint set.

\[\square\]

Rules that refine objects can be combined cleanly with rules that infer relationships between existing objects in COQL programs. For example, the rule in Example 4 can be combined with the rule in Example 2. In the resulting program, Fletcher would also be an answer to the query q1 under the “proofs in all possible worlds” semantics.

Rules that refine objects can be used to create new objects as well, in a fashion similar to the proposals that permit the creation of new objects using rules (e.g., [6]). Our technique avoids the problem faced by many object-creating proposals of ensuring that the “same” object is not created multiple times. If an object is created multiple times, possibly with different constraints on the E-attributes, the result is to conjoin all the constraints on the E-attributes.
4 Set-Order Constraints

In the examples discussed in the paper, we used order constraints (i.e., arithmetic constraints involving $<, \leq, =, \geq$ and $>$, but no arithmetic functions such as $+$, $-$ or *) and set-order constraints, a restricted form of set constraints involving $\subseteq$ and $\supseteq$, but no set functions such as $\cup$ and $\cap$. Techniques for quantifier elimination, checking satisfaction and entailment for order constraints over various domains are known (see [9], for instance). We now briefly describe a polynomial-time quantifier elimination algorithm for a conjunction of set-order constraints. Satisfaction and entailment of conjunctions of set-order constraints can be solved (in polynomial-time) using our quantifier elimination algorithm.

4.1 Quantifier Elimination for Set-Order Constraints

We will use the symbols $X, Y, Z$ to denote set variables that range over finite sets of elements of type $D$. A set-order constraint is of one of the following types:

$$c \in X, X \subseteq s, s \subseteq X, \bar{X} \subseteq \bar{Y}$$

where $c$ is a constant of type $D$, and $s$ is a set of constants of type $D$.

---

procedure Quantifier Elimination

Input: A conjunction $Q$ of set-order constraints and a set variable $\bar{Y}$ to be eliminated.
Output: A conjunction $Q'$ of set-order constraints, such that $\exists \bar{Y} Q$ and $Q'$ are equivalent.
Algorithm: Do the following steps in order:

1. First rewrite every constraint of the form $c \in X$ into $\{c\} \subseteq \bar{X}$.
2. For each set variable $X$, take the union of all sets $s$, such that $s \subseteq X$ is in the conjunction. Let the union be the set $L_X$. Delete all constraints of the form $s \subseteq X$ from the conjunction, and add the constraint $L_X \subseteq X$ to the conjunction.
3. For each set variable $X$ take the intersection of all sets $s$, such that $X \subseteq s$ is in the conjunction. Let the intersection be the set $U_X$. Delete all constraints of the form $X \subseteq s$ from the conjunction and add the constraint $X \subseteq U_X$.
4. For each pair of constraints of the form $\bar{N} \subseteq \bar{Y}$ and $\bar{Y} \subseteq \bar{M}$, where $\bar{Y}$ is the set variable to be eliminated, and $\bar{N}$ and $\bar{M}$ are either set variables or sets of constants, add the constraint $\bar{N} \subseteq \bar{M}$. After this is done for each such pair, delete all constraints in which $\bar{Y}$ occurs. Repeat steps 2 and 3.
5. Check each constraint of the form $s_1 \subseteq s_2$ where $s_1$ and $s_2$ are sets of constants from domain $D$. If they are all satisfied, delete all such constraints from the conjunction and return the conjunction of the remaining constraints. If any one of these constraints is not satisfied, then return FALSE. □
Example 5 (Quantifier elimination). Let \( Q \) be the following conjunction of set-order constraints: \( 3 \in Z, Z \subseteq X, X \subseteq \{3, 4, 8, 9\}, X \subseteq Y, Y \subseteq \{2, 3, 5, 7, 8\} \). From constraint \( Q \) we can eliminate set variable \( Y \) as follows:

**Step 1**: Replace \( 3 \in Z \) by \( \{3\} \subseteq Z \).

**Steps 2-3**: No change.

**Step 4**: We get \( \{3\} \subseteq Z, Z \subseteq X, X \subseteq \{3, 4, 8, 9\}, X \subseteq \{2, 3, 5, 7, 8\} \).

**Step 2**: No change.

**Step 3**: We get \( \{3\} \subseteq Z, Z \subseteq X, X \subseteq \{3, 8\} \).

**Step 5**: No change. Hence, we return \( \{3\} \subseteq Z, Z \subseteq X, X \subseteq \{3, 8\} \).

Suppose now, that we also want to eliminate set variable \( Z \). This will be done by quantifier elimination algorithm as follows:

**Steps 1–3**: No change.

**Step 4**: We get \( \{3\} \subseteq X, X \subseteq \{3, 8\} \).

**Step 5**: No change. Hence, we return \( \{3\} \subseteq X, X \subseteq \{3, 8\} \).

**Theorem 1.** Let \( Q \) be a conjunction of set-order constraints and \( Y \) be a set variable. The quantifier elimination algorithm on input \( Q \) and \( Y \) will yield in \( \mathsf{PTIME} \), in the size of \( Q \), a conjunction of set-order constraints \( Q' \) such that \( \exists Y Q \) and \( Q' \) are equivalent.

Further, if \( Q' \) has \( n \) set variables, then the number of conjuncts in \( Q' \) is at most \( n^2 + 2 \ast n \).

The quantifier elimination algorithm can also be used to check for satisfiability of a conjunction of set-order constraints, by successively eliminating set variables until either there are no more set variables remaining (in which case the original conjunction is satisfiable) or the quantifier elimination algorithm returns FALSE (in which case the original conjunction is unsatisfiable). The bound on the maximum number of conjuncts after eliminating a set variable guarantees a polynomial-time algorithm for checking satisfiability.

**Theorem 2.** Let \( Q \) be a conjunction of set-order constraints. Checking whether \( Q \) is satisfiable is in \( \mathsf{PTIME} \), in the size of \( Q \).

The algorithm for checking satisfiability cannot be used for checking for entailment of conjunctions of set-order constraints (using the reduction from a check for entailment to a polynomial number of checks for satisfaction), since set-order constraints are not closed under negation. (For example, \( X \not\subseteq Y \) is not a set-order constraint.) However, the quantifier elimination algorithm can be used directly as a basis for checking entailment of conjunctions of set-order constraints in \( \mathsf{PTIME} \) as follows.

Checking the entailment of a conjunction of set-order constraints \( Q_2 \) by a conjunction of set-order constraints \( Q_1 \) can be done by reduction to a number of entailment checks of each set-order constraint in \( Q_2 \) by the conjunction \( Q_1 \). The following result shows how the quantifier elimination algorithm can be used to check for entailment of a set-order constraint by an arbitrarily large conjunction of set-order constraints.
Theorem 3. Let $Q$ be a conjunction of set-order constraints over the set variables $X_1, \ldots, X_m$. Let $Q_1$ be the result of elimination of variables $X_3, \ldots, X_m$ from $Q$. Let $Q_2$ be the result of elimination of variable $X_2$ from $Q_1$. Then, (1) $Q$ entails $X_1 \subseteq X_2$ if and only if $Q_1$ entails $X_1 \subseteq X_2$, (2) $Q$ entails $X_1 \subseteq s$ if and only if $Q_2$ entails $X_1 \subseteq s$, and (3) $Q$ entails $s \subseteq X_1$ if and only if $Q_2$ entails $s \subseteq X_1$.

The following two results show how to check whether a set-order constraint is entailed by a simple form of a conjunction of set-order constraints.

Theorem 4. Let $Q$ be a conjunction of set-order constraints over $X$. Let $U_X$ be the upper bound (possibly the set of all elements in domain $D$) on $X$ and $L_X$ be the lower bound (possibly the empty set) on $X$.

Then $Q$ entails $X \subseteq s$ if and only if $U_X \subseteq s$. Also, $Q$ entails $s \subseteq X$ if and only if $s \subseteq L_X$.

Theorem 5. Let $Q$ be a conjunction of set-order constraints over $X_1, X_2$. Let $U_{X_1}$ be the upper bound (if any) on $X_1$ and $L_{X_2}$ be the lower bound (if any) on $X_2$. Then $Q$ entails $X_1 \subseteq X_2$ if and only if (1) $Q$ is unsatisfiable, or (2) $X_1 \subseteq X_2$ is in $Q$, or (3) $U_{X_1} \subseteq L_{X_2}$.

5 COQL: Model Theory and Fixpoint Semantics

COQL has a model-theoretic and an equivalent fixpoint semantics, based on the notions of constraint entailment and “proofs in all possible worlds”. The semantics of COQL is based on the notion of “proofs in all possible worlds” for several reasons.

First, if we adopted the “proof/truth in at least one possible world” semantics, object refinement becomes order dependent, and the program cannot be assigned a unique meaning. Under the “proofs in all possible worlds” semantics, object refinement is order independent. The following example illustrates this:

Example 6 (Order Dependence). Consider a program with the following two refinement rules.\(^5\)

\[
\begin{align*}
\text{W.Year of birth} & \leq 1560 : - \text{playwrights (W)}, \text{W.Year of birth} \leq 1565. \\
\text{W.Year of birth} & \geq 1570 : - \text{playwrights (W)}, \text{W.Year of birth} \geq 1566.
\end{align*}
\]

In Example 1, Shakespeare's year of birth is known to be between 1560 and 1570. Under the “proof/truth in at least one possible world” semantics, the order in which these two rules are applied could result in Shakespeare’s year of birth being refined to either 1560 or 1570. (Once one of the rules is applied, the other rule becomes inapplicable.) Under the “proofs in all possible worlds” semantics, neither of these rules would be applicable, and Shakespeare’s year of birth would not be refined.

\(^5\) Although the rules do not make intuitive sense, this example is purely for illustrating a point.
Second, if we adopted the “truth in all possible worlds” semantics, answering a query is computationally intractable. The “proofs in all possible worlds” semantics of a program can, however, be computed more efficiently, as we show later.

Example 7 (Computational Intractability). Consider a program with the following two refinement rules, and the objects from Example 1:

\[
\begin{align*}
\text{W.Year} &\cdot \text{of.birth} = 1565 : - \text{playwrights} (W), \text{W.Year} \cdot \text{of.birth} \leq 1566. \\
\text{W.Year} &\cdot \text{of.birth} = 1565 : - \text{playwrights} (W), \text{W.Year} \cdot \text{of.birth} \geq 1564.
\end{align*}
\]

The constraint Shakespeare Year of birth = 1565 would be conjoined to the object constraints, in each of the possible worlds consistent with the constraints on Shakespeare’s age, and hence under the “truth in all possible worlds” semantics. Determining this, however, requires reasoning by cases, which can be computationally intractable. Under the “proofs in all possible worlds” semantics, Shakespeare’s year of birth would not be refined, since the different possible worlds have different justifications for the addition of this constraint. \qed

Finally, an answer to a query is unconditionally true under the “proofs in all possible worlds” semantics. An answer to a query continues to be true, even after the database objects are monotonically refined.

We briefly describe the model-theoretic and fixpoint semantics here; details and the equivalence proof are omitted for reasons of space. Consider a COQL program \( P \), and a collection of facts and objects \( I \). We assume that all the variables in each rule body of \( P \) have been standardized apart (i.e., no variable occurs more than once in the body literals of a rule), possibly by introducing equality constraints between some of the variables; this is important for checking for entailment.

5.1 Model-theoretic Semantics

An assignment of facts and objects to the body literals of a rule \( r \) of program \( P \) makes the body of \( r \) true if the constraints associated with the facts and the objects entail the (instantiated) constraints between the variables present in the body of rule \( r \). A relationship inferring rule \( r \) is true in \( I \) if, for every assignment of facts and objects to the body literals of \( r \) that makes the body true, the instantiated head fact of rule \( r \) is entailed by (the constraints associated with) some fact \( f \) in \( I \). An object refinement rule \( r \) is true in \( I \) if, for every assignment of facts and objects to the body literals of \( r \) that makes the body true, the instantiated head object of \( r \) occurs in \( I \), and the instantiated head constraint of the rule is entailed by the object constraints associated with \( o_r \). The collection \( I \) of facts and objects is said to be a model of a COQL program if each program rule is true in \( I \).

The model-theoretic semantics of COQL is a least model semantics, where model \( M_1 \leq \) model \( M_2 \), if for each fact (or object) \( f_i \) in \( M_1 \), there is a fact
(or object) \( f_2 \) in \( M_2 \), such that \( f_1 \) entails \( f_2 \). The existence of a least model is guaranteed since we can show that the “intersection” of COQL program models is also a COQL program model.

### 5.2 Fixpoint semantics

The fixpoint semantics is defined in terms of an immediate consequence operator, \( T_P \). Given the collection \( I \) of facts and objects, we define \( T_P(I) \) as follows. Let \( r \) be a rule. If there is an assignment of facts and objects from \( I \) to literals in the body of \( r \) such that the body is true, then the instantiated head fact (or head object) is in \( T_P(I) \).

The fixpoint semantics of COQL is based on the least fixpoint of the \( T_P \) operator, which can be computed starting from the empty collection of facts and objects, as \( T_P(\emptyset) \cup T_P(T_P(\emptyset)) \cup \ldots \). In computing the unions, all the object constraints have to be conjoined together. The existence of the least fixpoint is guaranteed by the monotonicity of the \( T_P \) operator.

**Theorem 6.** Consider a COQL program \( P \). It has a least model semantics and a least fixpoint semantics, which coincide.

Note that in the absence of objects with object identifiers, the semantics of COQL is very similar to the standard semantics [10], except that constraint entailment is used instead of constraint satisfaction; this is required by the notion of “proofs in all possible worlds”.

The techniques of [5] can be used to show that if a COQL program and facts/objects use only arithmetic order constraints, the answer to a query can be computed in PTIME data complexity. The following result shows that a similar complexity is achieved for (a restricted case of) COQL programs with set-order constraints.

**Theorem 7.** Consider a COQL program \( P \) with only refinement rules using set-order constraints, and a collection of objects \( I \). Let the \( E \)-attributes of the objects in \( I \) be constrained using only set-order constraints. Computing the answer to a query can be done in PTIME in the size of \( I \).

### 6 Conclusions and Future Work

We presented the Constraint Object Data Model, and the Constraint Object Query Language, which we believe go a long way in incorporating the ability to represent and manipulate partially specified information in object-based database systems.

There are many interesting directions to pursue. Determining classes of programs with tractable data complexity is extremely important. Optimizing COQL queries is another important direction of research. Stuckey and Sudarshan [8] present compilation techniques for query constraints in logic programs, essentially extending Magic sets to handle general query constraints, not just equality.
constraints on queries. It would be interesting to see how these techniques apply to COQL programs. Finally, many of our ideas and techniques seem applicable to temporal database languages. Exploring the interconnections is likely to be an interesting direction of research.

Acknowledgements

The research of Raghu Ramakrishnan was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award with matching grants from DEC, Tandem and Xerox, and NSF grant IRI-9011563. The research of Peter Z. Revesz was supported by an Oliver E. Bird faculty fellowship. The electronic mail addresses of the authors are divesh@research.att.com, raghu@cs.wisc.edu and revesz@cse.unl.edu.

References


This article was processed using the LATEX macro package with LLNCS style