ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

A. Formica

SATISFIABILITY OF OBJECT-ORIENTED DATABASE CONSTRAINTS WITH SET AND BAG ATTRIBUTES

R. 549 Maggio 2001

Anna Formica – Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" del CNR, Viale Manzoni 30 - 00185 Roma, Italy. Email : anna.formica@iasi.cnr.it

This paper appears in Information Systems (IS) 28(3), pp. 213-224, 2003.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

The definition of methodologies for checking database constraint *satisfiability*, i.e., the absence of contradictions independently of any database state, is a fundamental and critical problem that has been marginally addressed in the literature. In this paper, a sound and complete algorithm is proposed for checking the satisfiability of a specific class of database integrity constraints in a simplified Object-Oriented model. Such a class includes cardinality constraints, *set* and *bag attributes*, and explicit integrity constraints involving comparison operators. The algorithm, conceived to support database schema design, allows the designer to distinguish among five different kinds of contradictions, each identified by an ad-hoc procedure.

Keywords: Object-Oriented databases, types, constraint satisfiability, typed properties, set attributes, bag attributes, cardinality constraints.

1. INTRODUCTION

With the advent of data models more expressive than the relational one, the existence of facilities to handle semantic integrity constraints is an essential premise to manage semantically rich data. In particular, the definition of methodologies for checking database constraint *satisfiability*, i.e., the absence of contradictions independently of any database state [20], becomes a fundamental problem, and the need for automatic tools supporting schema design becomes mandatory. This paper focuses on the satisfiability of a specific class of integrity constraints in a simplified Object-Oriented database (OODB) [2] model. Even if we concentrate on the static component of the Object-Oriented model, neglecting the behavioural part, the verification of the satisfiability of integrity constraints is a difficult activity that has been marginally addressed in the literarture [24]. In particular, the class of integrity constraints addressed in this paper includes cardinality constraints, and explicit integrity constraints involving comparison operators, referred to as *θ-constraints*. Particular emphasis is given to the presence of *set* and *bag attributes*, i.e., multivalued attributes that allow an object to be associated with a set or a bag of values, respectively. In the data model, recursive properties are also allowed and, furthermore, attributes can be typed with enumerated sets of values and intervals (*value_set* types). In this work, we will see how the coexistence of set attributes, bag attributes, *value_set* types, cardinality constraints, and θ-constraints impacts on schema satisfiability. The need for bags has been recognized not only in the database area, such as, for instance, in [21] (since they allow aggregate functions to be expressed in a natural way), but also in different fields, such as, for instance, in the Object-Oriented Analysis & Design area, where the *Object Constraint Language* (OCL) has been proposed as a standard for constraint specification [28].

The data model adopted in this paper is TQL^*, a fragment of the language TQL^{++} [19], [26] for conceptual modeling of OODB applications. The contribution of this work consists in the definition of a sound and complete algorithm that allows the satisfiability of TQL^* schemas to be checked in polynomial time. The algorithm also allows the designer to distinguish among five different kinds of contradictions, each identified by an ad-hoc procedure. Furthermore, one more contribution of this paper concerns the definition of the formal semantics of bag attributes according to Description Logics [10].

The paper is organized as follows. In Section 2, the data model is formally introduced, and the formal semantics of the TQL^* specification language is presented. In Section 3, five procedures are defined on which the satisfiability algorithm mentioned above is based. Successively, the algorithm is introduced, followed by the related soundness and completeness proof. In Section 4, the conclusion and future work are briefly discussed. Below the related work is given.

1.1. Related work

Implication constraints and $referential$ constraints are commonly used to specify semantic integrity constraints in database and knowledge base systems. The former generalize $functional$ *dependencies*, whereas the latter are *inclusion dependencies*. Since the implication problem for functional and inclusion dependencies is, in general, undecidable [12], in the literature the analysis focused on the identification of specific classes of integrity constraints that allow the implication problem to be decidable. For instance, in [29] the problem of reasoning with implication constraints and acyclic referential constraints has been addressed. In particular, the decidability of the IRC (*Implication* and $Referential\ Constraints$) refuting problem has been

proved, and a novel characterization for it has been given. In [16], a decidable, sound, and complete method for checking finite satisfiability of a restricted form of cyclic referential constraints for Object-Oriented databases has been proposed. In [24], the satisfiability problem has been investigated in the context of the Entity Relationship model, for a specific class of integrity constraints, namely the *cardinality ratio* constraints, used to express functional and numerical dependencies. In [9], a technique has been presented for checking the satisfiability of class definitions in an extended Object-Oriented data model, where it is possible to specify ISA relationships, disjointness constraints, inverse attributes (properties), and cardinality constraints. In [4], the satisfiability of Object-Oriented database schemas enriched with integrity constraints has been addressed. In particular, two alternative formalisms have been presented: one including *path relations* (that are more general than θ-constraints), but disallowing recursive schemas, the other one with the capability of expressing recursive schemas, but disallowing path relations. The implication problem has been investigated in the context of semistructured data and XML, for instance, in [8] and [15]. In [8], *path constraints* have been investigated, that are a generalization of the explicit integrity constraints addressed in this paper. In particular, the implication and finite implication problems for path constraints have been analyzed in the context of two Object-Oriented models, namely $\mathcal{M}^+$, essentially containing record types and set types, and $\mathcal{M}$, a restriction of the former that does not allow sets. Both problems are decidable in $\mathcal{M}$, whereas are undecidable in $\mathcal{M}^+$. In [15], the implication problem has been addressed for two subclasses of functional and inclusion dependencies, *keys* and $foreign\ keys$ respectively, extensively analyzed in relational databases. In particular, in the mentioned paper, four classes of constraints including keys and foreign keys for XML have been studied, and the implication problem for each of them has been analyzed. However, it is important to notice that the assumptions made in that paper for XML data model are quite different from those generally addressed in the Object-Oriented database setting (e.g., at an extensional level, finite trees vs cyclic graphs).

With respect to all the aforementioned papers, this work addresses the satisfiability of a specific class of Object-Oriented integrity constraints, including recursive types, bags, sets, cardinality constraints, and a restricted form of path constraints. Due to the different data model assumptions or modeling constructs addressed, none of the above mentioned papers contains the class of constraints investigated in this work.

It is worth recalling that, in the context of deductive databases, constraint satisfiability checking has been widely investigated by relying on theorem prover techniques. For instance in [7] a schema is a set of First Order Logic formulas, and schema satisfiability checking is performed by using the theorem prover *Satchmo*, successively refined in *Sic* [6]. However, in general, checking the satisfiability of schemas containing set attributes and cardinality constraints is one of the 75 problems that have been recognized difficult to be processed by theorem provers (but suitable to test their efficiency) [27].

Furthermore, in [5] and [13], *path equation* constraints on complex objects have been addressed, that can be compared with the equality constraints defined in this paper. In particular, such constraints have been analyzed in combination with functional dependencies, without addressing the satisfiability of constraints involving sets, bags, cardinality constraints and comparison operators.

Finally, regarding the bag constructor, in the literature we find some results concerning, for instance, the extension of the usual set operations to bags (see for instance [1]), algebraic query optimizations for data models supporting bags (as for instance [14]), or the expressive power of algebras for manipulating bags (see for instance [22]). However, again, these proposals do not

address constraint satisfiability checking in the presence of set and bag attributes.

2. FORMAL BASIS

The data model on which this work is based is TQL^*, that is a derivation of the language TQL^{++} [19], [26] aimed at modeling the structural aspects and the integrity constraints of an OODB application. TQL^* is a language compliant with ODMG [11], a standard for OODBs that is gaining a wide consensus within the database community.

2.1. TQL^* Syntax

In TQL^*, a *schema* is a set of *types*. A type contains the description of the structure of the objects that will populate the related *class* (i.e., the set of all the type *instances*), together with the integrity constraints that these objects have to satisfy. The formal syntax of TQL^* is presented in Definition 2.1: non-terminal symbols are enclosed between angular parenthesis, terminal symbols are in bold, whereas the remainder symbols represent user-defined strings (the underscore character stands for iteration).

Definition 2.1. [Syntax of TQL^*]

$\langle type \rangle$ ::= $t\_term \doteq \langle tuple \rangle$
| $\langle tuple \rangle, \langle c\_expr_1 \rangle, \_ \langle c\_expr_n \rangle$
$\langle tuple \rangle$ ::= $[\langle tp_1 \rangle, \_ \langle tp_m \rangle]$
$\langle tp \rangle$ ::= p_term **:** $\{\langle body \rangle\}_{m,M}$
| p_term^* **:** $\{\langle a\_type \rangle\}_{m,M}$
$\langle body \rangle$ ::= $\langle a\_type \rangle$
| t_term
$\langle a\_type \rangle$ ::= **integer**
| **real** | **boolean** | **string** | $value\_set$
$\langle c\_expr \rangle$ ::= *label:* **this**.$\langle path \rangle$ $\langle\ \theta\ \rangle$ C
$\langle path \rangle$ ::= $p\_term_1. \_ p\_term_{h-1}.p\_term_h$
| $p\_term_1. \_ p\_term_{h-1}.p\_term_h{}^*$
$\langle\ \theta\ \rangle$ ::= $\leq \mid < \mid > \mid \geq \mid = \mid \neq$

Example 2.1. *The following types form a* schema*:*
$person \doteq [name : string, age : integer]$,
$\quad ic_1 : this.age \leq 150$
$student \doteq [teacher : \{person\}+,$
$\qquad friend : \{student\},$
$\qquad exam\_mark^* : \{integer\}_{0,40},$
$\qquad topic : (comp\_sc, engin, math)]$,
$\quad ic_2 : this.exam\_mark^* \geq 18$

A type has a *label* (t_term) and a *tuple*, that is a set of *typed properties* (tp). A property is identified by a label (p_term), and can be typed by using: (i) *atomic* types (a_type), e.g., *integer* or *string* (for instance, in the example, $person.age$) or $value\_set$ types, that are specified between ordinary parenthesis by the interval extremes, or by enumeration, as for instance $student.topic$; (ii) t_terms (as, in the example, $student.teacher$), establishing an explicit link (or association) between two types. In a tuple, multiple occurrences of the same property label are not allowed.

Properties may be either *attributes* or *relationships*, according to ODMG. The former are typed according to the case (i) above, while the latter according to the case (ii). Relationships can form cycles, hence resulting in recursive types (for instance, in the example, *student* is a self recursive type). Properties are multivalued, that is, it is possible to specify that an object, instance of a type, can take more than one value in correspondence to a given property. Multivalued properties are specified by using curly braces (see, for instance, $friend$ in the type *student*). In particular, multivalued relationships are *set relationships*, i.e., an object can be associated with a *set* of objects, whereas multivalued attributes may be *set* or *bag attributes*, i.e., it is possible to associate an object with a *set* or a *bag* of values, respectively. Notationally, bag attributes differ from set attributes since they are followed by the superscript symbol *. For instance, in the Example 2.1, $exam\_mark^*$ is a bag attribute, whereas $friend$ is a set relationship, both of the *student* type.

As already mentioned, a property can be typed by using enumeration or an interval specification (in the case of ordered domains). In both these cases, the values that the objects can take are only the ones specified in the enumerated set or belonging to the interval, respectively.

In TQL^*, it is possible to express constraints on the minimum and maximum cardinality of the sets or bags of objects associated through multivalued properties. In the case of minimum cardinality equal to 0, the property is also referred to as a *null* property. In the previous example, $exam\_mark^*$ is a null property, and a *student* instance may have from 0 to 40 values (possibly with duplicates) associated with it, through this attribute. In TQL^*, the absence of curly braces is a short form to denote singlevalued properties (as, for instance, $person.name$), curly braces that are not followed by cardinality constraints stand for any cardinality (as, for instance, $student.friend$), whereas curly braces followed by the + symbol stand for minimum cardinality 1 (as, for instance, $student.teacher$).

TQL^* explicit integrity constraints are θ-*constraints* (c_expr), where θ stands for a comparison operator, such as "=", "$\geq$", ">". A θ-constraint has *label* identifying the constraint, a left hand side specified by the keyword *this* followed by a *path*, and a right hand side represented by a constant, say C, that can be an integer, a real, a string, or a boolean constant. The keyword *this* refers to a single object of the class of the type the integrity constraint is associated with. A *path* is a sequence of p_terms, expressed according to the dot-notation formalism, that allows navigation through types. In a *path*, the same property label may occur more than once. θ-constraints are specified in the schema, being an integral part of types. For example, in the Example 2.1, the explicit integrity constraint ic_2 states that the examination marks of a *student* must be all greater than 18.

In order to present the notion of a schema, the $\mathcal{T}$ function is introduced below. Such a function is defined on a *t_term* followed by a *path*, and returns the type of the first property of the *path* (as defined in the tuple associated with the *t_term*), followed by the remainder sequence of properties of that *path*.

Definition 2.2. **[The $\mathcal{T}$ function]** *Given a schema Σ, the $\mathcal{T}$ function is defined on a* t_term*, say τ, followed by a sequence of n (≥ 1)* p_terms *of Σ, as follows:*

- $\mathcal{T}(\tau.p_1.\ \ldots\ .p_n) = \sigma.p_2.\ \ldots\ .p_n$
  if $\tau = [..., p_1 : \{\sigma\}_{m,M}, ...]$
- $\mathcal{T}(\tau.p_1.\ \ldots\ .p_n)$ *is undefined otherwise.*

For $h = 1 \ldots n$, $\mathcal{T}^h$ is the composition of $\mathcal{T}$ h times, i.e.:
$\mathcal{T}^h(\tau.p_1.\ \ldots\ .p_n) =$

$\mathcal{T}(\mathcal{T}(...(\mathcal{T}(\tau.p_1.\ ...\ .p_n))...))$
and $\mathcal{T}^0(\tau.p_1.\ ...\ .p_n) = \tau.p_1.\ ...\ .p_n$ *(the identity function).*

For instance, in Example 2.1:
$\mathcal{T}(student.friend.exam\_mark^*) =$
$student.exam\_mark^*$
and:
$\mathcal{T}^2(student.friend.exam\_mark^*) = integer$
whereas:
$\mathcal{T}(student.child)$ is undefined.

Definition 2.3. **[Non-null integrity constraint]** *Consider a type* τ *and an integrity constraint:*
ic: $this.p_1.\ ...\ .p_n\ \theta\ C$
associated with τ*. Then ic is* non-null *if each* p_i*,* $i = 1 \ldots n$*, is not a null property, i.e., the minimum cardinality associated with the* p_i *property in the traversed type is strictly greater than zero.*

The formal definition of a TQL^* schema follows.

Definition 2.4. **[TQL^* schema]** *A finite set of types is a* TQL^* schema *(*schema*, for short) if:*

- *every type and integrity constraint label is uniquely defined (i.e., the same label is not associated with more than one definition);*
- *there are no dangling type labels (i.e., every* t_term *declared in the schema is defined);*
- *for each type* τ *and each associated integrity constraint, say ic:* $this.p_1.\ ...\ .p_n\ \theta\ C$*,* $\mathcal{T}^h(\tau.p_1.\ ...\ .p_n)$ *are defined, for* $h = 1 \ldots n$*.*

The first two requirements, in the above definition, are obvious: to prevent ambiguity, types and integrity constraints must have unique definitions, and every type label appearing in the schema must be defined. The last point requires that, for each integrity constraint, *paths* must be defined according to the tuples defined in the schema (i.e., each property of a *path* must be present in the tuple of the "traversed" type).

Notice that in a schema, besides the above requirements, the integrity constraints are supposed to be correctly typed, e.g., in Example 2.1, the integrity constraint:
$ic_2 : this.friend > 27$
associated with *student* would be rejected at a pre-processing stage, by using a type-checker. Therefore, we suppose that, for each θ-constraint, the constant C is compatible with the type associated with the last property of the path, in the related type definition. Notice that in this paper inheritance hierarchies (extensively investigated by the author, for instance, in [3], [17], [18]) are not addressed.

2.2. Semantics of TQL^*

The formal semantics of a TQL^* schema is given according to the formal semantics of Description Logics, as defined in [10]. In particular, in this paper the notion of interpretation has been extended in order to deal with bag attributes.

Given a TQL^* schema Σ, let $\mathcal{S}$ be the set of *t_terms*, *atomic* types, and *value_set* types of Σ (that correspond to the *atomic concepts* in [10]). Furthermore, let $\mathcal{P}$ and $\mathcal{P}^*$ be a partition of the set of properties, say $\mathcal{Q}$, of Σ, i.e., $\mathcal{P} \cap \mathcal{P}^* = \emptyset$ and $\mathcal{P} \cup \mathcal{P}^* = \mathcal{Q}$. In particular, $\mathcal{P}$ contains all the set properties (*p_terms*) of Σ (corresponding to the *atomic roles* in [10]), whereas $\mathcal{P}^*$ contains all the bag attributes (*p_terms**) of Σ[1].

An *interpretation* $\mathcal{I}^* = (\Delta^{\mathcal{I}^*}, .^{\mathcal{I}^*})$ over Σ consists of a non-empty finite set $\Delta^{\mathcal{I}^*}$, that is the *domain* of $\mathcal{I}^*$, and a function $.^{\mathcal{I}^*}$, that is the *interpretation function* of $\mathcal{I}^*$, that maps:

- every type $\tau \in \mathcal{S}$ to an element $(\tau)^{\mathcal{I}^*}$ of the powerset $\wp(\Delta^{\mathcal{I}^*})$ (i.e., $(\tau)^{\mathcal{I}^*}$ is the set of *instances* of τ);
- every *p_term* $\in \mathcal{P}$ to an element $(p\_term)^{\mathcal{I}^*}$ of the powerset $\wp(\Delta^{\mathcal{I}^*} \times \Delta^{\mathcal{I}^*})$;
- every *p_term*$^* \in \mathcal{P}^*$ to an element $(p\_tem^*)^{\mathcal{I}^*}$ of the *powerbag* [21] $\wp^*(\Delta^{\mathcal{I}^*} \times \Delta^{\mathcal{I}^*})$.

Then, a type:
$\tau \doteq$ *tuple*, *c_expr*$_1$, ..., *c_expr*$_n$
is an *inclusion assertion* as defined in [10] (i.e., it specifies only necessary conditions for an object to be an instance of the type τ), for which the interpretation function is defined as follows ($\#S$ denotes the cardinality of the set S):

1. $(tuple, c\_expr_1, ..., c\_expr_n)^{\mathcal{I}^*} =$
   $(tuple)^{\mathcal{I}^*} \cap (\bigcap_i (c\_expr_i)^{\mathcal{I}^*})$

2. $(tuple)^{\mathcal{I}^*} =$
   $\bigcap_j ([p_j{:}\{type_j\}_{m_j,M_j}])^{\mathcal{I}^*}$
   where p_j can be either a set or a bag property, and $type_j$ an *atomic* type or a *t_term* according to the syntax;

3. $([p{:}\{type\}_{m,M}])^{\mathcal{I}^*} =$
   $\{x \in \Delta^{\mathcal{I}^*} \mid m \leq \#\ \{y{:}\ (x,y) \in (p)^{\mathcal{I}^*},$
   $y \in (type)^{\mathcal{I}^*}\} \leq M\}$
   where p and *type* are defined similarly to p_j and $type_j$ above, respectively;

4. $(c\_expr)^{\mathcal{I}^*} =$
   (label: this.*path* θ $C)^{\mathcal{I}^*} =$
   (label: this.$p_1.p_2 \ldots .p_h$ θ $C)^{\mathcal{I}^*} =$
   $\{x \in \Delta^{\mathcal{I}^*} \mid \forall\ y_1, \ldots, y_h{:}\ (x, y_1) \in (p_1)^{\mathcal{I}^*},$
   $(y_1, y_2) \in (p_2)^{\mathcal{I}^*} \ \ldots$
   $(y_{h-1}, y_h) \in (p_h)^{\mathcal{I}^*} \Rightarrow y_h\ \theta\ C\}$
   where p_i, $i = 1$ ... h-1, are set properties, p_h can be either a set or a bag attribute, and θ is one of the comparison operators $\geq$, $>$, $=$, $\neq$, etc..

A *model* of Σ is an interpretation $\mathcal{I}^* = (\Delta^{\mathcal{I}^*}, .^{\mathcal{I}^*})$ that *satisfies* all the inclusion assertions in Σ, i.e.:
$(\tau)^{\mathcal{I}^*} \subseteq (tuple, c\_expr_1, ..., c\_expr_n)^{\mathcal{I}^*}$.
A type $\tau \in \mathcal{S}$ is *satisfiable* in the schema Σ if the schema admits a model for which $(\tau)^{\mathcal{I}^*} \neq \emptyset$.

[1]Therefore set attributes and bag attributes having the same name, as for instance *phone* and *phone**, will be considered different properties.

Finally, the schema Σ is *satisfiable* if it admits a model for which each type is satisfiable in Σ.

Example 2.2. *Consider the following schema:*
$university \doteq [name : string,$
$\quad teacher : \{graduate\}+],$
$\quad ic : this.teacher.exam\_mark \geq 27$
$graduate \doteq [name : string,$
$\quad exam\_mark : \{(18..30)\}_{10,40},$
$\quad topic : string]$
In this schema, exam_mark *is a set attribute. In particular, for a* graduate *at least 10 examination marks are required and, in the case of graduates that are teachers of a* university*, they must be greater than* 27*, according to the explicit integrity constraint ic (but not exceeding 30, as specified by the* value_set *type of* exam_mark*). Since* exam_mark *is a set attribute, it is not possible to instantiate any* graduate *object that is a* teacher *of a* university*, since it is not possible to define any set of cardinality at least* 10 *with a set of* 4 *elements. Therefore this schema is unsatisfiable. Notice that the* graduate *type is satisfiable since the cardinality of the* value_set *type is compliant with the minimum cardinality required. Of course, if the set attribute* exam_mark *is replaced with the bag attribute* exam_mark**, the schema becomes satisfiable.*

In the rest of the paper, an algorithm for checking the satisfiability of a TQL^* schema is introduced.

3. SCHEMA SATISFIABILITY CHECKING

In this section the notion of a *range* of an integrity constraint is first introduced. Successively, five procedures are presented, on which the algorithm for checking the satisfiability of a TQL^* schema is based. Regarding the notation, in the following, given a schema Σ and a *t_term* τ of Σ, $I(\tau)$ indicates the set of θ-constraints associated with the type τ in Σ. Furthermore, in the absence of specific assumptions, given any *ic*: $this.p_1...p_n\ \theta\ C$, the attribute p_n can be either a set or a bag attribute.

Definition 3.1. [The range $\mathcal{R}$] *Given a schema* Σ*, consider a* t_term τ *of* Σ *such that* $I(\tau) \neq \emptyset$*, and suppose that* $ic \in I(\tau)$ *where:*
$\quad ic: this.path\ \theta\ C$
Then, the range *of the integrity constraint ic, indicated as* $\mathcal{R}(ic)$*, is defined as follows:*

- *if* C *is an integer or a real constant,* $\mathcal{R}(ic)$ *is a set of integers or reals respectively, defined by the intervals:*
  $\mathcal{R}(ic) = (-\infty, C)$ *if* θ *is "<";*
  $\mathcal{R}(ic) = (-\infty, C]$ *if* θ *is "≤";*
  $\mathcal{R}(ic) = [C, C]$ *if* θ *is "=";*
  ... (and so on in all the other cases)
  where square brackets and ordinary parenthesis denote closed and open intervals, respectively;

- *if* C *is a string or a boolean constant, there are two possible cases. If* θ *is "=",* $\mathcal{R}(ic)$ *is the singleton formed by the* C *constant; otherwise, if* θ *is "≠",* $\mathcal{R}(ic)$ *is the complement of* C

Procedure **IntraTuple(τ):** *boolean*
input: $\tau \doteq [q_1 : \{\delta_1\}_{r_1,R_1}, \ldots, q_n : \{\delta_n\}_{r_n,R_n}]$;
output: true or false.
$IntraTuple(\tau)$:= true;
if for some set attribute q_j of τ, $1 \leq j \leq n$,
δ_j is $value\_set_j$, and $\#\{value\_set_j\} < r_j$,
then $IntraTuple(\tau)$:= false.

Figure 1: The *IntraTuple* procedure

in the set of all possible strings, or the singleton formed by the opposite boolean constant, respectively.

In Figure 6, the algorithm for schema satisfiability checking is presented. It is organized according to five procedures. The first one, called *IntraTuple*, is shown in Figure 1 and concerns the compatibility between the cardinality of the *value_set* type and the minimum of the associated cardinality constraints, in the case of set attributes. Given a *t_term* τ, this procedure returns a boolean value, corresponding to the absence of *IntraTuple cardinality contradictions*. For instance, in the following type:

$graduate \doteq [name : string,$
$exam\_mark : \{(18..30)\}_{20,40}, topic : string]$

the set attribute *exam_mark* generates an *IntraTuple* cardinality contradiction since it is not possible to define any set of cardinality (at least) 20 with a set of 13 elements.

Once the absence of *IntraTuple* cardinality contradictions has been checked, the rationale on which the algorithm is based is the following. For each integrity constraint of the schema, the range is progressively refined by the *IntraType*, *InterType*, and *ValueSet* procedures, each devoted to check the absence of specific kinds of contradictions. Finally, if the resulting range is non-empty, the *CardCheck* procedure allows the designer to check if there are further incompatibilities due to the cardinality constraints. Such procedures are illustrated below, and an example for each kind of detected contradictions is shown.

The *IntraType* procedure concerns the presence of contradicting integrity constraints within the same type. It takes a type and one integrity constraint as input, and returns the range of the integrity constraint, possibly refined. For instance, consider the types:

$university \doteq [name : string,$
$teacher : \{graduate\}+],$
$ic_1 : this.teacher.exam\_mark^* \geq 27,$
$ic_2 : this.teacher.exam\_mark^* \leq 25$
$graduate \doteq [name : string,$
$exam\_mark^* : \{integer\}+, topic : string]$

Procedure **IntraType**(τ, ic_i)**:** V
input: τ, $ic_i \in I(\tau)$, ic_i: $this.p_1$. ... $.p_n$ θ_i C_i;
output: an interval V.
  $V := \mathcal{R}(ic_i)$
  for each non-null $ic_j \in I(\tau)$ s.t.:
    ic_j: $this.p_1$. ... $.p_n$ θ_j C_j
      $V := V \cap \mathcal{R}(ic_j)$.

Figure 2: The *IntraType* procedure

(recall that the + symbol stands for minimum cardinality 1). In this case, the range of the constraint ic_1 (and similarly of ic_2) is refined with the empty set, since the ranges of ic_1 and ic_2 are disjoint. This is an example of an *IntraType contradiction* detected by the procedure shown in Figure 2.

The *InterType* procedure shown in Figure 3, as well as the other three remaining procedures, takes as input a type τ, an integrity constraint ic_i associated with τ, and an interval, standing for the refined range of the constraint ic_i, at that stage. In this procedure such a range is further refined taking into account the other integrity constraints associated with the traversed types of the schema.

For instance, the following types:

$university \doteq [name : string,$
  $teacher : \{graduate\}+],$
  $ic_1 : this.teacher.exam\_mark^* \geq 27$
$graduate \doteq [name : string,$
  $exam\_mark^* : \{integer\}+, topic : string],$
  $ic_2 : this.exam\_mark^* \leq 25$

contain an *InterType contradiction* in correspondence to the integrity constraint ic_1, since the attribute $exam\_mark^*$ requires values greater than 27, within the type $university$, and values lesser than 25 within the type $graduate$.

The $ValueSet$ procedure, shown in Figure 4, regards the check between the range of the integrity constraint, as refined at that stage, and the *value_set* type (if there are any) associated with the last property of the path, in the related type. Notice that the $IntraType$, $InterType$, and $ValueSet$ contradictions are independent of the presence of bag or set attributes. For instance, in the following schema:

$university \doteq [name : string,$
  $teacher : \{graduate\}+],$
  $ic : this.teacher.exam\_mark^* \geq 27$
$graduate \doteq [name : string,$
  $exam\_mark^* : \{(18..26)\}+, topic : string]$

Procedure **InterType**(τ, ic_i, V)**:** W
input: τ, $ic_i \in I(\tau)$, ic_i: $this.p_1.\ ...\ .p_n\ \theta_i\ C_i$,
an interval V;
output: an interval W.
$W := V$;
for each non-null ic_j, σ, and k,
$1 \leq k \leq n-1$, s.t.:
ic_j: $this.p_{k+1}.\ ...\ .p_n\ \theta_j\ C_j$, $ic_j \in I(\sigma)$,
and $\mathcal{T}^k(\tau.p_1.\ ...\ .p_n) = \sigma.p_{k+1}.\ ...\ .p_n$,
$W := W \cap \mathcal{R}(ic_j)$.

Figure 3: The *InterType* procedure

Procedure **ValueSet**(τ, ic_i, W)**:** U
input: τ, $ic_i \in I(\tau)$, ic_i: $this.p_1.\ ...\ .p_n\ \theta_i\ C_i$,
an interval W;
output: an interval U.
$U := W$;
if $\mathcal{T}^{n-1}(\tau.p_1.\ ...\ .p_n) = \delta.p_n$,
and $\delta \doteq [..., p_n : \{value\_set_{p_n}\}_{m,M}, ...]$
then $U := U \cap \{value\_set_{p_n}\}$.

Figure 4: The *ValueSet* procedure

we have a *ValueSet contradiction* since a *university* must have teachers of type *graduate*, with examination marks greater than 27 (according to the explicit integrity constraint), but lesser than 26 (as required for *graduate*).

Finally the last procedure, namely *CardCheck*, is shown in Figure 5 and concerns the verification about the compatibility between the cardinality of the range of the input integrity constraint, and the minimum cardinality constraint associated with the last property of the path, in the case it is a set attribute. This procedure returns a boolean value corresponding to the absence of an *InterType cardinality contradiction*. This is the case of the Example 2.2.

Regarding the complexity of the algorithm, consider a schema with n types, each having at most m properties and r integrity constraints. Furthermore, suppose that for each integrity constraint the length (number of property occurrences) of the path is at most s. Then the satisfiability algorithm of Figure 6 has complexity $O(n(m + r^2 s^2))$, taking into account that the cost for the comparison between paths is linear in the length of the paths.

Procedure **CardCheck**(τ, ic_i, U)**:** *boolean*
input: τ, $ic_i \in I(\tau)$, ic_i: $this.p_1$. ... $.p_n$ θ_i C_i,
an interval U;
output: true or false.
$CardCheck(\tau, ic_i, U)$:= true
if $\mathcal{T}^{n-1}(\tau.p_1. \ldots .p_n) = \delta.p_n$,
$\delta \doteq [..., p_n : \{value\_set_{p_n}\}_{m,M}, ...]$,
p_n is a set attribute, and $\#U < m$
then $CardCheck(\tau, ic_i, U)$:= false.

Figure 5: The *CardCheck* procedure

Algorithm **Sat**(Σ)**:** *boolean*
input: a schema Σ;
output: true or false.
$Sat(\Sigma)$:= false;
if for some $\tau \in \Sigma$, $IntraTuple(\tau)$ = false, then
print '*IntraTuple cardinality contradiction*',
exit;
else
for all $\tau \in \Sigma$
for all non-null $ic_i \in I(\tau)$
$IntraType(\tau, ic_i) = V$
if $V = \emptyset$, then print
'*IntraType contradiction*', exit;
$InterType(\tau, ic_i, V) = W$,
if $W = \emptyset$, then print
'*InterType contradiction*', exit;
$ValueSet(\tau, ic_i, W) = U$,
if $U = \emptyset$, then print
'*ValueSet contradiction*', exit;
if $CardCheck(\tau, ic_i, U)$ = false,
then print '*InterType cardinality contradiction*', exit;
$Sat(\Sigma)$:= true, and print 'Σ *is satisfiable*'.

Figure 6: The *Sat* algorithm for schema *satisfiability* checking

3.1. Soundness and completeness of the algorithm

In this section, the satisfiable TQL^* schemas are characterized, that is, the soundness and the completeness of the algorithm shown in Figure 6 are given.

Theorem 3.1. [Characterization of satisfiable schemas] *A TQL^* schema Σ is satisfiable if and only if $Sat(\Sigma) = true$.*

Proof.
$\Rightarrow$ *By contradiction. Suppose that $Sat(\Sigma) = false$. Then, in Σ there exists at least one type τ, such that one of the following conditions holds:*

- *$IntraTuple(\tau) = false$. Therefore, there exists at least one set attribute p of τ, $\tau \doteq [..., p : \{value\_set_p\}_{r,R}, ...]$, such that $\#\{value\_set_p\} < r$: contradiction;*
- *there exists at least one non-null $ic_i \in I(\tau)$ s.t.:*
  - *$IntraType(\tau, ic_i) = V = \emptyset$. Therefore, there exists at least one non-null $ic_j \in I(\tau)$ such that the refined range of ic_i is empty: contradiction;*
  - *$IntraType(\tau, ic_i) = V \neq \emptyset$ and $InterType(\tau, ic_i, V) = W = \emptyset$. Therefore, there exists at least one non-null ic_j, one type σ, $ic_j \in I(\sigma)$, and an integer $1 \leq k \leq n$ - 1, such that the refined range of ic_i is empty: contradiction;*
  - *$InterType(\tau, ic_i, V) = W \neq \emptyset$ and $ValueSet(\tau, ic_i, W) = U = \emptyset$. Therefore, the intersection between the refined range of ic_i and the value_set type of the last property of the path of ic_i is empty: contradiction;*
  - *$ValueSet(\tau, ic_i, W) = U \neq \emptyset$ and $CardCheck(\tau, ic_i, U) = false$. Therefore, the last property of the path of ic_i is a set attribute whose associated minimum cardinality constraint, in the traversed type, is strictly greater than the cardinality of the refined range of ic_i: contradiction.*

$\Leftarrow$ *By construction. Let $\mathcal{S}$ be the set of type labels (*t_terms*), atomic types (*a_types*), and* value_set *types of Σ. For each* t_term *$\tau \in \mathcal{S}$, construct a labeled directed graph $G_\tau = (N_\tau, A_\tau)$ as follows. Let $x_0^\tau \in N_\tau$ be one node of the graph G_τ, that is referred to as the* representative *element of τ instances. Furthermore:*

- tuple drawing*: for each set or bag property p defining the tuple of τ, where:*
  $\tau \doteq [..., p : \{\eta\}_{r,R}, ...]$
  let $x_i^\eta \in N_\tau$, $i = 1...r$, be r nodes of G_τ, and $\langle x_0^\tau, x_i^\eta \rangle_p \in A_\tau$, $i = 1...r$, be r directed arcs of G_τ, labeled with the property p, taking into account that different properties of τ require the introduction of disjoint sets of nodes. Therefore, if the type τ has u properties, each associated with minimum cardinality r_i, $i = 1...u$, the number of nodes introduced so far is $1 + r_1 + r_2 + ... + r_u$.
- path drawing*: for each non-null integrity constraint $ic \in I(\tau)$:*
  ic: $this.p.p_1.p_2$ ... $.p_n$ θ C
  where:
  $\mathcal{T}(\tau.p.p_1.p_2. ... .p_n) = \eta.p_1...p_n$ and
  $\eta \doteq [\ldots, p_1 : \{\eta_1\}_{s_1,S_1}, \ldots]$
  $\mathcal{T}^2(\tau.p.p_1.p_2. ... .p_n) = \eta_1.p_2...p_n$ and

$\eta_1 \dot{\preceq} [\ldots, p_2 : \{\eta_2\}_{s_2,S_2}, \ldots]$

....

$\mathcal{T}^{n-1}(\tau.p.p_1.p_2. \ldots .p_n) = \eta_{n-1}.p_n$

and $\eta_{n-1} \dot{\preceq} [\ldots, p_n : \{\eta_n\}_{s_n,S_n}, \ldots]$

and η_n *is necessarily an atomic type, let* $z_{k_j}^{\eta_j} \in N_\tau$, $j = 1...n$, $k_j = 1...s_j$ *be nodes of* G_τ *such that, if* $\langle x_0^\tau, x_i^\eta \rangle_p \in A_\tau$, $i = 1...r$, *then:*

$\langle x_i^\eta, z_{k_1}^{\eta_1} \rangle_{p_1} \in A_\tau$, $i = 1...r$, $k_1 = 1...s_1$,

$\langle z_{k_j}^{\eta_j}, z_{k_{j+1}}^{\eta_{j+1}} \rangle_{p_{j+1}} \in A_\tau$, $j = 1...n$-1,

$k_j = 1...s_j$,

taking into account that, in the case of integrity constraints having a common subpath of the form $p.p_1.p_2....p_h$, $1 \leq h \leq n$, *the nodes* $z_{k_j}^{\eta_j}$, $j = 1...h$, *are introduced only once. Therefore, for the integrity constraint ic,* $s_1 + s_2 + \ldots + s_n$ *further nodes have been introduced.*

Suppose the graph $G_\tau = (N_\tau, A_\tau)$ *has been constructed for any* t_term $\tau \in \mathcal{S}$. *Then, for each* G_τ, *consider all the paths starting from the representative element of* τ *instances* x_0^τ, *i.e., each defined as:*

$\{\langle x_0^\tau, w_1^{\delta_1} \rangle_{v_1}, \langle w_1^{\delta_1}, w_2^{\delta_2} \rangle_{v_2}, ..., \langle w_{l-1}^{\delta_{l-1}}, w_l^{\delta_l} \rangle_{v_l}\}$

where, for $k = 1...l$, $\delta_k \in \mathcal{S}$ *and* v_k *are properties of* Σ. *Then, for each node* $w_k^{\delta_k}$, $k = 1...l$, *where* δ_k *is a* t_term *(that can also coincide with* τ*):*

- path sharing: *consider the graph* G_{δ_k} *and all the sets of paths starting from the representative element of* δ_k *instances, say* $y_0^{\delta_k}$, *labeled with the same sequences of properties, that is, suppose that* $q = q_1.q_2...q_m$ *is one of these sequences, consider the set:*

  $P_q = \{..., \{\langle y_0^{\delta_k}, z_1^{\gamma_1} \rangle_{q_1}, \langle z_1^{\gamma_1}, z_2^{\gamma_2} \rangle_{q_2}, ...,$
  $\langle z_{m-1}^{\gamma_{m-1}}, z_m^{\gamma_m} \rangle_{q_m}\}, ...\}$.

  and let s_q *be the cardinality of the set* P_q. *If there exists in* G_τ *a path starting from the node* $w_k^{\delta_k}$, *labeled with the sequence of properties* $q_1...q_h$, $1 \leq h \leq m$, *assume that* $v_{k+1} = q_1$, ... , $v_{k+h} = q_h$, *where* $k + h \leq l$. *Then, connect the graphs* G_τ *and* G_{δ_k} *by adding* s_q *arcs labeled with* q_{h+1}, *in such a way that the paths of* G_{δ_k} *labeled with the sequences* $q_{h+2}...q_m$ *can be shared* [2] *also by* G_τ. *That is, from the node* x_0^τ *of* G_τ *the paths:*

  $\{\langle x_0^\tau, w_1^{\delta_1} \rangle_{v_1}, ..., \langle w_{k-1}^{\delta_{k-1}}, w_k^{\delta_k} \rangle_{v_k},$
  $\langle w_k^{\delta_k}, w_{k+1}^{\delta_{k+1}} \rangle_{v_{k+1}=q_1}, \cdots$
  $\langle w_{k+h-1}^{\delta_{k+h-1}}, w_{k+h}^{\delta_{k+h}} \rangle_{v_{k+h}=q_h},$
  $\langle w_{k+h}^{\delta_{k+h}}, z_{h+1}^{\gamma_{h+1}} \rangle_{q_{h+1}}, \cdots$
  $\langle z_{m-1}^{\gamma_{m-1}}, z_m^{\gamma_m} \rangle_{q_m}\}$

  are defined, where $\langle w_{k+h}^{\delta_{k+h}}, z_{h+1}^{\gamma_{h+1}} \rangle_{q_{h+1}}$ *are* s_q *newly defined arcs. Otherwise, in the case the node* $w_k^{\delta_k}$ *of* G_τ *has no outgoing arcs labeled with* q_1, *then* $h = 0$, *that is, the previous paths are replaced with the following:*

  $\{\langle x_0^\tau, w_1^{\delta_1} \rangle_{v_1}, ..., \langle w_{k-1}^{\delta_{k-1}}, w_k^{\delta_k} \rangle_{v_k},$
  $\langle w_k^{\delta_k}, z_1^{\gamma_1} \rangle_{q_1}, ..., \langle z_{m-1}^{\gamma_{m-1}}, z_m^{\gamma_m} \rangle_{q_m}\}$

  where $\langle w_k^{\delta_k}, z_1^{\gamma_1} \rangle_{q_1}$ *are* s_q *newly defined arcs.*

Once the above steps have been performed for any t_term $\tau \in \mathcal{S}$, *let* $G_\Sigma = (N_\Sigma, A_\Sigma)$ *be the resulting (possibly non-connected) graph, and consider an interpretation* $\mathcal{I}^* = (\Delta^{\mathcal{I}^*}, .^{\mathcal{I}^*})$ *over* Σ

[2] Path sharing allows object sharing, according to the Object-Oriented data model.

such that:
$\Delta^{\mathcal{I}^*} = N_\Sigma$
for any $\tau \in \mathcal{S}$*:*
$(\tau)^{\mathcal{I}^*} = \{y_k^\tau \in N_\Sigma \mid k \geq 0\}$
and for any property p *of the types of* Σ*:*
$(p)^{\mathcal{I}^*} = \{(x_h^\tau, y_k^\eta) \mid \langle x_h^\tau, y_k^\eta \rangle_p \in A_\Sigma,$
$\tau, \eta \in \mathcal{S}, k, h \geq 0\}$
From the hypothesis of the theorem, i.e., $Sat(\Sigma) = true$*, it is possible to instantiate all the variables, say* y_j^γ*,* $\gamma \in \mathcal{S}$*,* $j \geq 0$*, labeling the nodes of the graph* G_Σ *in such a way that* y_j^γ *is an instance the type* γ*, i.e., the interpretation* $\mathcal{I}^* = (\Delta^{\mathcal{I}^*}, .^{\mathcal{I}^*})$ *over* Σ *is also a model* [3] *of* Σ*.*

4. CONCLUSION AND FUTURE WORK

In this paper a sound and complete algorithm for checking the satisfiability of a specific class of OODB integrity constraints has been proposed. The schema is specified by using TQL^*, a fragment of the language TQL^{++} for conceptual modeling of OODB applications. As a future work, there are no obstacles in extending this result to more expressive OODB schemas including, for instance, inverse attributes, keys, inheritance, and explicit integrity constraints comparing paths. In particular, the satisfiability of TQL^* schemas enriched with integrity constraints comparing paths is an interesting topic to investigate, however requiring a deeper understanding of the underlying theory. For instance, the methodology defined in [16], mentioned in the Related Work, has been conceived as an ad-hoc methodology to deal with integrity constraints comparing paths that is not suited to deal also with the integrity constraints defined in this paper.

References

[1] J.Albert. *Algebraic Properties of Bag Data Types*; Very Large Data Bases (VLDB), Barcelona, Spain, pp.211-219, 1991.

[3] For instance, consider the types:
$person \doteq [child : \{person\}_{2,10}, age : integer]$,
$ic_1 : this.child.age < 10$
$course \doteq [taken\_by : \{person\}_{1,30}]$,
$ic_2 : this.taken\_by.age > 10$
the resulting model is defined by mapping types and properties of the schema as follows:
$(person)^{\mathcal{I}^*} = \{x_0^{person}, x_1^{person}, x_2^{person}, w_1^{person}\}$
$(course)^{\mathcal{I}^*} = \{w_0^{course}\}$
$(child)^{\mathcal{I}^*} = \{(x_0^{person}, x_1^{person}), (x_0^{person}, x_2^{person}),$
$(x_1^{person}, x_1^{person}), (x_1^{person}, x_2^{person}),$
$(x_2^{person}, x_1^{person}), (x_2^{person}, x_2^{person}),$
$(w_1^{person}, x_1^{person}), (w_1^{person}, x_2^{person})\}$
$(age)^{\mathcal{I}^*} = \{(x_0^{person}, x_1^{integer}), (x_1^{person}, z_1^{integer}),$
$(x_2^{person}, z_1^{integer}), (w_1^{person}, y_1^{integer})\}$
$(taken\_by)^{\mathcal{I}^*} = \{(w_0^{course}, w_1^{person})\}$
$(integer)^{\mathcal{I}^*} = \{x_1^{integer}, z_1^{integer}, y_1^{integer}\}$
where the instances x_1^{person}, x_2^{person}, and w_1^{person} share the objects associated with x_0^{person} (the representative element of *person* instances) as regards the property *child*. Whereas, since the attribute *age* is constrained by ic_1 and ic_2, the further instances $z_1^{integer}$, and $y_1^{integer}$ have been introduced, the former lesser than 10, the latter greater than 10.

[2] C.Beeri. *A formal approach to object-oriented databases*; Data & Knowledge Engineering (DKE), North-Holland, Vol.5, pp.353-382, 1990.

[3] C.Beeri, A.Formica, M.Missikoff. *Inheritance Hierarchy Design in Object-Oriented Databases*; Data & Knowledge Engineering (DKE), North-Holland, Vol.30, No.3, pp.191-216, 1999.

[4] D.Beneventano, S.Bergamaschi, S.Lodi, C.Sartori. *Consistency Checking in Complex Object Database Schemata with Integrity Constraints*; IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol.10, No.4, pp.576-598, 1998.

[5] M.F.van Bommel, G.E. Weddell. *Reasoning About Equations and Functional Dependencies on Complex Objects*; IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol.6, No.3, pp.455-469, 1994.

[6] F.Bry, N.Eisinger, H.Schutz, S.Torge. *SIC: Satisfiability Checking for Integrity Constraints*; Deductive Databases and Logic Programming (DDLP), Manchester, UK, pp.25-36, 1998.

[7] F.Bry, R.Manthey. *Checking Consistency of Database Constraints: a Logical Basis*; Very Large Data Bases (VLDB), Kyoto, Japan, pp.13-20, 1986.

[8] P.Buneman, W.Fan, S.Weinstein. *Interaction between Path and Type Constraints*; Symposium on Principles of Database Systems (PODS), Philadelphia, PA, pp.56-67, 1999.

[9] D.Calvanese, M.Lenzerini. *Making Object-Oriented Schemes More Expressive*; Symposium on Principles of Database Systems (PODS), Minneapolis, USA, pp.243-254, 1994.

[10] D.Calvanese, M.Lenzerini, D. Nardi. *Description Logics for Conceptual Data Modeling*; in *Logics for Databases and Information Systems*, Kluwer Academic Publisher, Jan Chomicki and Günter Saake (Eds.), pp.229-263, 1998.

[11] R.G.G.Cattel, D.Barry. *ODMG-97: The Object Database Standard*; Release 2.0, Morgan Kaufmann Series in Data Management Systems, Jim Gray Series Editor, 1997.

[12] A.K.Chandra, M.Vardi. *The Implication Problem for Functional and Inclusion Dependencies is Undecidable*; SIAM Journal on Computing, Vol.14, No.3, 1985.

[13] N.Coburn, G.E.Weddell. *Path Constraints for Graph-Based Data Models: Towards a Unified Theory of Typing Constraints, Equations, and Functional Dependencies*; Deductive and Object-Oriented Databases (DOOD), Munich, Germany, pp.312-331, 1991.

[14] U.Dayal, N.Goodman, R.H.Katz. *An Extended Relational Algebra with Control Over Duplicate Elimination*; Symposium on Principles of Database Systems (PODS), Los Angeles, CA, pp.117-123, 1982.

[15] W.Fan, L.Libkin. *On XML Integrity Constraints in the Presence of DTDs*; Symposium on Principles of Database Systems (PODS), Santa Barbara, CA, pp.114-125, 2001.

[16] A.Formica. *Finite Satisfiability of Integrity Constraints in Object-Oriented Database Schemas*; IEEE Transactions on Knowledge & Data Engineering (TKDE), Vol.14, No.1, pp.123-139, 2002.

[17] A.Formica, H.D.Groger, M.Missikoff. *Object-Oriented Database Schema Analysis and Inheritance Processing: a Graph-Theoretic Approach*; Data & Knowledge Engineering (DKE), North-Holland, Vol. 24, No. 2, pp.157-181, 1997.

[18] A.Formica, H.D.Groger, M.Missikoff. *An Efficient Method For Checking Object-Oriented Database Schema Correctness*; ACM Transactions on Database Systems (TODS), Vol.23, No.3, pp.333-369, 1998.

[19] A.Formica, M.Missikoff. *Integrity Constraints Representation in Object-Oriented Databases*; In "Information and Knowledge Management", T.W.Finin, C.K.Nicholas, Y.Yesha (Eds.), Lecture Notes in Computer Science (LNCS) 752, Springer-Verlag, pp.69-85, 1993.

[20] H.Gallaire, J.M.Nicolas. *Logic and Databases: An assessment*; International Conference on Database Theory (ICDT'90), S.Abiteboul, P.Kanellakis (Eds.), Lecture Notes in Computer Science (LNCS) 470, Springer-Verlag, 1990.

[21] S.Grumbach, T.Milo. *Towards Tractable Algebras for Bags*; Symposium on Principles of Database Systems (PODS), Washington D.C., USA, pp.49-58, 1993.

[22] S.Grumbach, T.Milo, Y.Kornatzky. *Calculi for Bags and their Complexity*; Workshop on Database Programming Languages (DBPL), New York City, USA, pp.65-79, 1993.

[23] S.N.Khoshafian, G.P.Copeland. *Object Identity*; Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Portland, Oregon, pp.406-416, 1986.

[24] M.Lenzerini, P.Nobili. *On the Satisfiability of Dependency Constraints in Entity-Relationship Schemata*; Information Systems (IS) , Vol.15, N.4, pp.453-461, 1990.

[25] L.Libkin, L.Wong. *Some Properties of Query Languages for Bags*; Workshop on Database Programming Languages (DBPL), New York City, USA, pp.97-114, 1993.

[26] M.Missikoff, M.Toiati. *MOSAICO - A System for Conceptual Modeling and Rapid Prototyping of Object-Oriented Database Applications*; International Conference on Management of Data (SIGMOD), Minneapolis, USA, p.508, 1994.

[27] F.J.Pelletier. *Seventy-Five Problems for Testing Automatic Theorem Provers*; Journal of Automated Reasoning Vol.2, No.2, pp.191-216, 1986.

[28] J.Warmer, A.Kleppe. *OCL: The Constraint Language of the UML*; Journal of Object-Oriented Programming (JOOP), pp.10-13, 1999.

[29] X.Zhang, Z.M.Ozsoyoglu. *Implication and Referential Constraints: A New Formal Reasoning*; IEEE Transactions on Knowledge and Data Engineering (TKDE), V.9, No.6, 1997.