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


ISSN: 1128–3378
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 literature [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 $\theta$-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 $M^+$, essentially containing record types and set types, and $M$, a restriction of the former that does not allow sets. Both problems are decidable in $M$, whereas are undecidable in $M^+$. 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*]

\[
\begin{align*}
\langle\text{type}\rangle & ::= \langle\text{t_term} \lessgtr \langle\text{tuple} \rangle \rangle \\
\langle\text{tuple}\rangle & ::= \langle\text{tp} \rangle, \langle\text{tp}\rangle
\end{align*}
\]

\[
\begin{align*}
\langle\text{tp}\rangle & ::= \langle\text{p_term} \colon \{\langle\text{body}\rangle\}\rangle_{m,M} \\
\langle\text{body}\rangle & ::= \langle\text{a_type}\rangle
\end{align*}
\]

\[
\begin{align*}
\langle\text{a_type}\rangle & ::= \text{integer} \\
\langle\text{c_expr}\rangle & ::= \text{label} : \text{this.(path)} \langle\theta\rangle C \\
\langle\text{path}\rangle & ::= \langle\text{p_term}\rangle, \ldots, \langle\text{p_term}\rangle
\end{align*}
\]

\[
\begin{align*}
\langle\theta\rangle & ::= \leq | < | \geq | = | \neq
\end{align*}
\]

Example 2.1. The following types form a schema:

\[
\begin{align*}
\text{person} & \lessgtr \{\text{name} : \text{string}, \text{age} : \text{integer}\}, \\
\text{ic}_1 & : \text{this.age} \leq 150 \\
\text{student} & \lessgtr \{\text{teacher} : \{\text{person}\}+, \\
\text{friend} & : \{\text{student}\}, \\
\text{exam_mark} & : \{\text{integer}\}_{0,40}, \\
\text{topic} & : \{\text{comp_sc, engin, math}\}, \\
\text{ic}_2 & : \text{this.exam_mark} \geq 18
\end{align*}
\]

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 *.

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 $\theta$-constraints ($c_{\text{expr}}$), where $\theta$ stands for a comparison operator, such as "$=", "\geq", "\gt$". A $\theta$-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 $p_{\text{terms}}$, expressed according to the dot-notation formalism, that allows navigation through types. In a path, the same property label may occur more than once. $\theta$-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 $T$ function is introduced below. Such a function is defined on a $t_{\text{term}}$ followed by a path, and returns the type of the first property of that path. For example, in the Example 2.1, the $T$ function defined on a path student.friend returns the type of the first property of that path.

**Definition 2.2. [The $T$ function]** Given a schema $\Sigma$, the $T$ function is defined on a $t_{\text{term}}$, say $\tau$, followed by a sequence of $n \geq 1$ $p_{\text{terms}}$ of $\Sigma$, as follows:

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

- $T(\tau.p_1.\ldots.p_n)$ is undefined otherwise.

For $h = 1 \ldots n$, $T^h$ is the composition of $T$ $h$ times, i.e.:

- $T^h(\tau.p_1.\ldots.p_n) =$
\[ T(T(...(T(\tau.p_1. \ldots .p_n))...)) \]
and \[ T^0(\tau.p_1. \ldots .p_n) = \tau.p_1. \ldots .p_n \] (the identity function).

For instance, in Example 2.1:
\[ T(\text{student.friend.exam}* \text{mark}^*) = \text{student.exam}* \text{mark}^* \]
and:
\[ T^2(\text{student.friend.exam}* \text{mark}^*) = \text{integer} \]
whereas:
\[ T(\text{student.child}) \] is undefined.

**Definition 2.3.** [Non-null integrity constraint] Consider a type \( \tau \) and an integrity constraint:
\[ \text{ic}: \text{this}.p_1. \ldots .p_n \theta C \]
associated with \( \tau \). Then \( \text{ic} \) 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 \( \tau \) and each associated integrity constraint, say \( \text{ic}: \text{this}.p_1. \ldots .p_n \theta C \), \( T^h(\tau.p_1. \ldots .p_n) \) 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:
\[ \text{ic}_2: \text{this.friend} > 27 \]
associated with \text{student} would be rejected at a pre-processing stage, by using a type-checker. Therefore, we suppose that, for each \( \theta \)-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 $\Sigma$, let $S$ be the set of $t$-terms, atomic types, and value-set types of $\Sigma$ (that correspond to the atomic concepts in [10]). Furthermore, let $P$ and $P^*$ be a partition of the set of properties, say $Q$, of $\Sigma$, i.e., $P \cap P^* = \emptyset$ and $P \cup P^* = Q$. In particular, $P$ contains all the set properties ($p$-terms) of $\Sigma$ (corresponding to the atomic roles in [10]), whereas $P^*$ contains all the bag attributes ($p$-terms*) of $\Sigma$.

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

- every type $\tau \in S$ to an element $(\tau)^* \in \Delta^*$ of the powerset $\varphi(\Delta^*)$ (i.e., $(\tau)^* \in \text{set}$ is the set of instances of $\tau$);
- every $p$-term $\in P$ to an element $(p$-term$)^* \in \Delta^*$ of the powerset $\varphi(\Delta^* \times \Delta^*)$;
- every $p$-term$^*$ $\in P^*$ to an element $(p$-term$)^* \in \text{powerbag}$ [21] $\varphi(\Delta^* \times \Delta^*)$.

Then, a type:

\[ \tau \leq \text{tuple, c\_expr}_1, ..., c\_expr_n \]

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 $\tau$), for which the interpretation function is defined as follows ($\#S$ denotes the cardinality of the set $S$):

1. $(\text{tuple, c\_expr}_1, ..., c\_expr_n)^* = (\text{tuple})^* \cap (\bigcap_i (c\_expr_i)^*)$
2. $(\text{tuple})^* = \bigcap_j ([p_j; \{\text{type}_j\}_{m_j,M_j}])^*$
   where $p_j$ can be either a set or a bag property, and $\text{type}_j$ an atomic type or a $t$-term according to the syntax;
3. $([p; \{\text{type}\}_{m,M}])^* = \{x \in \Delta^* \mid m \leq \# \{y; (x,y) \in (p)^*, y \in (\text{type})^*\} \leq M\}$
   where $p$ and $\text{type}$ are defined similarly to $p_j$ and $\text{type}_j$ above, respectively;
4. $(c\_expr)^* = (\text{label: this, path } \theta C)^* = (\text{label: this, p}_1, p_2, \ldots, p_h \theta C)^* = \{x \in \Delta^* \mid \text{\forall } y_1, \ldots, y_h; (x,y_1) \in (p_1)^*, (y_1,y_2) \in (p_2)^* \ldots (y_{h-1},y_h) \in (p_h)^* \Rightarrow y_h \theta C\}$
   where $p_i, i = 1 \ldots h$ are set properties, $p_h$ can be either a set or a bag attribute, and $\theta$ is one of the comparison operators $\geq, >, =, \neq$, etc..

A model of $\Sigma$ is an interpretation $\mathcal{I}^* = (\Delta^*, \mathcal{I}^*)$ that satisfies all the inclusion assertions in $\Sigma$, i.e.:

\[ (\tau)^* \subseteq (\text{tuple, c\_expr}_1, ..., c\_expr_n)^* \]  

A type $\tau \in S$ is satisfiable in the schema $\Sigma$ if the schema admits a model for which $(\tau)^* \neq \emptyset$.

\[ ^1 \text{Therefore set attributes and bag attributes having the same name, as for instance } \text{phone and phone}^*, \text{will be considered different properties.} \]
Finally, the schema $\Sigma$ is *satisfiable* if it admits a model for which each type is satisfiable in $\Sigma$.

**Example 2.2.** Consider the following schema:

university $\preceq \{(\text{name} : \text{string}),$

  \text{teacher} : \{(\text{graduate}+)\},$

ic : \text{this.teacher.exam} \geq 27$

graduate $\preceq \{(\text{name} : \text{string}),$

  \text{exam\ mark} : \{(18..30)\}_{10,40},$

  \text{topic} : \text{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 $\Sigma$ and a t\_term $\tau$ of $\Sigma$, $I(\tau)$ indicates the set of $\theta$-constraints associated with the type $\tau$ in $\Sigma$. 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 $R$]** Given a schema $\Sigma$, consider a t\_term $\tau$ of $\Sigma$ such that $I(\tau) \neq \emptyset$, and suppose that ic $\in I(\tau)$ where:

ic: this.path $\theta$ C

Then, the range of the integrity constraint ic, indicated as $R(ic)$, is defined as follows:

- if $C$ is an integer or a real constant, $R(ic)$ is a set of integers or reals respectively, defined by the intervals:

  $R(ic) = (-\infty, C)$ if $\theta$ is "<";

  $R(ic) = (-\infty, C]$ if $\theta$ is "$\leq$";

  $R(ic) = [C, C]$ if $\theta$ 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 $\theta$ is "=" $\neq", R(ic)$ is the singleton formed by the $C$ constant; otherwise, if $\theta$ is "$\neq$, $R(ic)$ is the complement of $C$
Procedure IntraTuple(\(\tau\)): boolean

input: \(\tau \supseteq [q_1 : \{\delta_1\}_{r_1}, \ldots, q_n : \{\delta_n\}_{r_n}]\);
output: true or false.

\[\text{IntraTuple}(\tau) := \begin{cases} 	ext{true;} & \text{if for some set attribute } q_j \text{ of } \tau, 1 \leq j \leq n, \\ & \delta_j \text{ is value_set}_j, \text{ and } \#\{\text{value_set}_j\} < r_j, \\ \text{false.} & \end{cases}\]

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 term \(\tau\), this procedure returns a boolean value, corresponding to the absence of IntraTuple cardinality contradictions. For instance, in the following type:

\[
\text{graduate} \geq [\text{name} : \text{string}, \\
\hspace{1cm} \text{exam_mark} : \{(18..30)\}_{20,40}, \text{topic} : \text{string}]
\]

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:

\[
\text{university} \geq [\text{name} : \text{string}, \\
\hspace{1cm} \text{teacher} : \{\text{graduate}\}+], \\
\hspace{1cm} \text{ic}_1 : \text{this.teacher.exam_mark} \geq 27, \\
\hspace{1cm} \text{ic}_2 : \text{this.teacher.exam_mark} \leq 25 \\
\text{graduate} \leq [\text{name} : \text{string}, \\
\hspace{1cm} \text{exam_mark} : \{\text{integer}\}+, \text{topic} : \text{string}]
\]
Procedure **IntraType**(τ, ic_i): V

**input:** τ, ic_i ∈ I(τ), ic_i: this.p_1. ... . p_n θ_i C_i;
**output:** an interval V.

V := R(ic_i)
for each non-null ic_j ∈ I(τ) s.t.:
    ic_j: this.p_1. ... . p_n θ_j C_j
    V := V ∩ 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 ≤ [name : string,
    teacher : {graduate}+],
    ic_1: this.teacher.exam_mark* ≥ 27
graduate ≤ [name : string,
    exam_mark*: {integer}+, topic : string],
    ic_2: this.exam_mark* ≤ 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 ≤ [name : string,
    teacher : {graduate}+],
    ic: this.teacher.exam_mark* ≥ 27
graduate ≤ [name : string,
    exam_mark*: {{(18..26)}+}, topic : string]
Procedure \textbf{InterType}(\tau, ic_i, V): W
\begin{itemize}
\item \textbf{input:} \(\tau, ic_i \in I(\tau), ic_i: this.p_1 \ldots p_n \theta_i C_i,\) an interval \(V;\)
\item \textbf{output:} an interval \(W;\)
\end{itemize}
\begin{itemize}
\item \(W := V;\)
\item for each non-null \(ic_j, \sigma,\) and \(k,\) \(1 \leq k \leq n -1,\) s.t.:
\begin{itemize}
\item \(ic_j: this.p_{k+1} \ldots p_n \theta_j C_j, ic_j \in I(\sigma),\)
\item and \(T^k(\tau.p_1 \ldots p_n) = \sigma.p_{k+1} \ldots p_n,\)
\end{itemize}
\item \(W := W \cap R(ic_j).\)
\end{itemize}

Figure 3: The \textbf{InterType} procedure

Procedure \textbf{ValueSet}(\tau, ic_i, W): U
\begin{itemize}
\item \textbf{input:} \(\tau, ic_i \in I(\tau), ic_i: this.p_1 \ldots p_n \theta_i C_i,\) an interval \(W;\)
\item \textbf{output:} an interval \(U;\)
\end{itemize}
\begin{itemize}
\item \(U := W;\)
\item if \(T^{n-1}(\tau.p_1 \ldots p_n) = \delta.p_n,\)
\begin{itemize}
\item and \(\delta \preceq [...]_n : \{value\_set_{p_n}\}_{m,M}, [...]\)
\end{itemize}
\item then \(U := U \cap \{value\_set_{p_n}\}.\)
\end{itemize}

Figure 4: The \textbf{ValueSet} procedure

we have a \textit{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 \textbf{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 \textit{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 \(O(n(m + r^2s^2)),\) taking into account that the cost for the comparison between paths is linear in the length of the paths.
**Procedure CardCheck(τ, ic, U):** boolean  
**input:** τ, ic ∈ I(τ), ic: this.p₁...pn θ_i Ci, an interval U;  
**output:** true or false.  
CardCheck(τ, ic, U) := true  
if T^{n-1}(τ.p₁...pn) = δ.pn,  
δ ⪯ [..., pn : {value_set}_m.M, ...],  
p_n is a set attribute, and #U < m  
then CardCheck(τ, ic, U) := false.

Figure 5: The CardCheck procedure

**Algorithm Sat(Σ):** boolean  
**input:** a schema Σ;  
**output:** true or false.  
Sat(Σ) := false;  
if for some τ ∈ Σ, IntraTuple(τ) = false, then  
print 'IntraTuple cardinality contradiction',  
exit;  
else  
for all τ ∈ Σ  
for all non-null ic ∈ I(τ)  
IntraType(τ, ic) = V  
if V = Ø, then print  
'IntraType contradiction', exit;  
InterType(τ, ic, V) = W,  
if W = Ø, then print  
'InterType contradiction', exit;  
ValueSet(τ, ic, W) = U,  
if U = Ø, then print  
'ValueSet contradiction', exit;  
if CardCheck(τ, ic, U) = false,  
then print 'InterType cardinality contradiction', exit;  
Sat(Σ) := 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(Σ) = true.

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

- **IntraTuple(τ) = false.** Therefore, there exists at least one set attribute p of τ, τ ≤ [..., p : {value_set_p}_r,R,...], such that #{value_set_p} < r: contradiction;
- there exists at least one non-null ic_i ∈ I(τ) s.t.:
  - IntraType(τ, ic_i) = V = ∅. Therefore, there exists at least one non-null ic_j ∈ I(τ) such that the refined range of ic_j is empty: contradiction;
  - IntraType(τ, ic_i) = V ≠ ∅ and Intertype(τ, ic_i, V) = W = ∅. Therefore, there exists at least one non-null ic_j, one type σ, ic_j ∈ I(σ), and an integer 1 ≤ k ≤ n - 1, such that the refined range of ic_j is empty: contradiction;
  - Intertype(τ, ic_i, V) = W ≠ ∅ and ValueSet(τ, ic_i, W) = U = ∅. 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(τ, ic_i, W) = U ≠ ∅ and CardCheck(τ, 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.

⇐ By construction. Let S be the set of type labels (t_terms), atomic types (a_types), and value_set types of Σ. For each t_term τ ∈ S, construct a labeled directed graph G_τ = (N_τ,A_τ) as follows. Let x_0^n ∈ N_τ 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:
  \[ τ \leq [..., p : \{\eta\}_r,R,...] \]
  let x^i_0^n ∈ N_τ, i = 1...r, be r nodes of G_τ, and (x^i_0^n, x^j_0^n) ∈ A_τ, 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 ∈ I(τ):
  \[ ic : this.p.p_1.p_2 ... .p_n \theta C \]
  where:
  \[ T(τ.p.p_1.p_2 ... .p_n) = \eta.p_1...p_n \] and
  \[ \eta \leq [..., p_1 : \{\eta_1\}_s_1,s_1,...] \]
  \[ T^2(τ.p.p_1.p_2 ... .p_n) = \eta_1.p_2...p_n \] and
Path sharing allows object sharing, according to the Object-Oriented data model. Therefore, for the integrity constraint

\[ I \subseteq \{ (\tau, p, \eta) \mid \tau, p, \eta \in \mathcal{S} \} \]

considering all the paths starting from the representative element of \( \tau \) instances \( x_0^\tau \), i.e., each defined as:

\[ \{ (x_0^\tau, \eta_1^1, \eta_2^1, \ldots, \eta_l^1)_{\nu_1}, (x_1^\delta, w_1^2, \ldots, w_{l-1}^2, w_l^2)_{\nu_2}, \ldots, (\eta_{l-1}^1, \eta_l^1)_{\nu_1} \} \]

where, for \( k = 1 \ldots l \), \( \delta_k \in \mathcal{S} \) and \( v_k \) are properties of \( \Sigma \). Then, for each node \( w_k^h \), \( k = 1 \ldots l \), where \( \delta_k \) is a \( t \)-term (that can also coincide with \( \tau \)):

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

\[ P_q = \{ (x_0^\tau, \eta_1^1, \ldots, \eta_l^1)_{\nu_1}, (y_0^\delta, \eta_1^1)_{\nu_1}, (y_1^\delta, \eta_2^1)_{\nu_2}, \ldots, (y_{l-1}^\delta, \eta_l^1)_{\nu_1} \} \]

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

\[ \{ (x_0^\tau, w_1^\delta)_{\nu_1}, \ldots, (w_{l-1}^\delta, w_l^\delta)_{\nu_1}, (w_{k-1}^\delta, w_k^\delta)_{\nu_1}, (w_{k+1}^\delta, w_k^\delta)_{\nu_1} \} \]

are defined, where \( (w_{k+h}^\delta, \eta_{h+1}^\delta)_{\eta_{h+1}^\delta} \) are \( s_q \) newly defined arcs. Otherwise, in the case the node \( w_{k+1}^\delta \) of \( G_\tau \) has no outgoing arcs labeled with \( q_1 \), then \( h = 0 \), that is, the previous paths are replaced with the following:

\[ \{ (x_0^\tau, w_1^\delta)_{\nu_1}, \ldots, (w_{l-1}^\delta, w_l^\delta)_{\nu_1}, (w_{k-1}^\delta, \eta_1^\delta)_{\nu_1}, (w_k^\delta, \eta_1^\delta)_{\nu_1}, (w_{k+1}^\delta, \eta_1^\delta)_{\nu_1} \} \]

where \( (w_k^\delta, \eta_1^\delta)_{\eta_1^\delta} \) 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 \( I^* = (\Delta^*, I^*) \) over \( \Sigma \)

\(^2\)Path sharing allows object sharing, according to the Object-Oriented data model.
such that:
\[ \Delta^* = N_\Sigma \]
for any \( \tau \in S \):
\[ (\tau)^{\ast} = \{ y^*_k \in N_\Sigma | k \geq 0 \} \]
and for any property \( p \) of the types of \( \Sigma \):
\[ (p)^{\ast} = \{(x^*_i, y^*_k) | (x^*_i, y^*_k)p \in A_\Sigma, \tau, \eta \in S, k, h \geq 0 \} \]
From the hypothesis of the theorem, i.e., \( \text{Sat}(\Sigma) = \text{true} \), it is possible to instantiate all the variables, say \( y^*_j, \gamma \in S, j \geq 0 \), labeling the nodes of the graph \( G_\Sigma \) in such a way that \( y^*_j \) is an instance the type \( \gamma \), i.e., the interpretation \( I^* = (\Delta^*, \Sigma^*) \) over \( \Sigma \) is also a model\(^4\) of \( \Sigma \).

4 For instance, consider the types:

- \( \text{person} \leq [\text{child} : \{ \text{person} \}_{2,10}, \text{age} : \text{integer}], \)
- \( i_{c_1} : \text{this.child.age} < 10 \)
- \( \text{course} \leq [\text{taken.by} : \{ \text{person} \}_{1,30}], \)
- \( i_{c_2} : \text{this.taken.by.age} > 10 \)

the resulting model is defined by mapping types and properties of the schema as follows:

- \( \text{person}^{\ast} = \{ x_{0,1}^{\text{person}}, x_2^{\text{person}}, w^{\text{person}} \} \)
- \( \text{course}^{\ast} = \{ w^{\text{course}} \} \)
- \( \text{child}^{\ast} = \{ (x_0^{\text{person}}, x_1^{\text{person}}), (x_0^{\text{person}}, x_2^{\text{person}}), \)
- \( (x_1^{\text{person}}, x_0^{\text{person}}), (x_1^{\text{person}}, x_2^{\text{person}}), \)
- \( (x_2^{\text{person}}, x_0^{\text{person}}), (x_2^{\text{person}}, w^{\text{person}}) \} \)
- \( \text{age}^{\ast} = \{ (x_0^{\text{person}}, x_1^{\text{integer}}), (x_1^{\text{person}}, x_2^{\text{person}}) \}
- \( \text{taken.by}^{\ast} = \{ w^{\text{course}}, w^{\text{person}} \} \)
- \( \text{integer}^{\ast} = \{ x_1^{\text{integer}}, y_1^{\text{integer}} \} \)

where the instances \( x_0^{\text{person}}, x_2^{\text{person}} \), and \( w^{\text{person}} \) share the objects associated with \( x_0^{\text{person}} \) (the representative element of \( \text{person} \) instances) as regards the property \( \text{child} \). Whereas, since the attribute \( \text{age} \) is constrained by \( i_{c_1} \) and \( i_{c_2} \), the further instances \( z_1^{\text{integer}} \), and \( y_1^{\text{integer}} \) have been introduced, the former lesser than 10, the latter greater than 10.

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


