Scalable Analysis of Conceptual Data Models

Matthew J. McGill
Michigan State University
East Lansing, MI, USA
mmcgill@cse.msu.edu

Laura K. Dillon
Michigan State University
East Lansing, MI, USA
ldillon@cse.msu.edu

R.E.K. Stirewalt
LogicBlox, Inc.
Atlanta, GA, USA
kurt.stirewalt@logicblox.com

ABSTRACT

Conceptual data models describe information systems without the burden of implementation details, and are increasingly used to generate code. They could also be analyzed for consistency and to generate test data except that the expressive constraints supported by popular modeling notations make such analysis intractable. In an earlier empirical study of conceptual models created at LogicBlox Inc., Smaragdakis, Csallner, and Subramanian found that a restricted subset of ORM, called ORM^-, includes the vast majority of constraints used in practice and, moreover, allows scalable analysis. After that study, however, LogicBlox Inc obtained a new ORM modeling tool, which supports discovery and specification of more complex constraints than the previous tool. We report findings of a follow-up study of models constructed using the more powerful tool. Our study finds that LogicBlox developers increasingly rely on a small number of features not in the ORM^- subset. We extend ORM^- with support for two of them: objectification and a restricted class of external uniqueness constraints. The extensions significantly improve our ability to analyze the ORM models created by developers using the new tool. We also show that a recent change to ORM has rendered the original ORM^- algorithms unsound, in general; but that an efficient test suffices to show that these algorithms are in fact sound for the ORM^- constraints appearing in any of the models currently in use at LogicBlox.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; H.2.1 [**Database Management**]: Logical Design—*Data models*

General Terms

Verification, algorithms

Keywords

Conceptual modeling, ORM, test data generation, databases

1. INTRODUCTION

Developers of information systems create conceptual models to describe a problem domain in a manner free from implementation details. In early stages of a project, these models serve as a vehicle by which developers and users communicate. In later stages, they are increasingly being used to generate portions of the application code. Because a code generator may silently propagate modeling errors down into an implementation, developers need tools to support verification and validation of conceptual models. Static analysis tools can automatically identify modeling problems, such as inconsistent constraints, that are very difficult to locate manually in large models. Tools that generate sample data conforming to a model's constraints facilitate validation: both developers and customers can inspect the sample data for invalid configurations, which may indicate missing constraints. Unfortunately, it is undecidable whether a conceptual model with unrestricted constraints is consistent, and intractable even for some fairly restrictive constraint classes [1, 2, 20]; generation of sample data that satisfy constraints face similar challenges. Methods for analyzing conceptual models must therefore carefully balance scalability with the expressiveness of the supported constraint set.

The Object Role Modeling (ORM) language is a fact-based conceptual modeling language with a graphical syntax, and a semantics based on first-order logic [12]. Like an Entity-Relationship (ER) diagram [6] or a UML class diagram [9], an ORM model describes a data domain in terms of objects and their relationships. ORM includes primitive constraints types, such as ring constraints, that must be expressed using OCL in a UML model; yet ORM's graphical syntax is more amenable to analysis than arbitrary OCL expressions. While OCL encompasses first-order logic and is thus undecidable in general, nearly all of ORM's graphical primitives can be expressed in a decidable fragment of first-order logic [15].

Smaragdakis, Csallner, and Subramanian identified ORM^-, a subset of ORM version 1 (ORM1) that supports efficient consistency checking and test data generation. They also reported results of a study of ORM1 models constructed at LogicBlox, which found that ORM^- included the vast majority of constraints in use at that time. Shortly after that study, however, LogicBlox obtained a new modeling tool, called Norma [7], which supports finding and specifying additional constraints not supported by the previous modeling tool. Based on ORM version 2 (ORM2) [11], Norma also incorporates subtle changes in the semantics of some of the constraints from ORM1.

This paper extends the results of [20] in four ways. First, we present findings of a study of the ORM2 models that were constructed at LogicBlox since the earlier study. The latter study shows that developers now routinely use features not in ORM^-. Second, we extend the algorithms from [20] to support efficient analysis

of ORM$^-$ models that are extended with the two features not in ORM$^-$: *objectification* and a restricted class of *external uniqueness constraints*. Objectification is by far the most frequent of the constraints not in ORM$^-$ used by LogicBlox developers. External uniqueness constraints occur much less frequently, but are needed in specifying *compound reference schemes*, which explain how multiple values combine to identify entities. Third, we show that a change in the semantics of *value constraints* from ORM1 to ORM2 makes the ORM$^-$ algorithm for generating test data unsound under ORM2 semantics. Fourth, to address this problem, we give an efficient test that implies an ORM$^-$ model has the same *instances* under ORM1 semantics and ORM2 semantics. The ORM$^-$ algorithm generates correct instances for models that satisfy this test. The results of our study suggest that the majority of models created in practice fall into this category.

The remainder of the paper is structured as follows. First, we describe the relevant features of ORM (Section 2). Next, we present the results of our study of ORM2 models developed at LogicBlox since the study reported in [20] (Section 3). We then summarize the algorithms from [20] (Section 4), and describe how to extend these algorithms to support a restricted class of external uniqueness constraints (Section 5) and objectification (Section 6). Next, we discuss ORM2 changes to the semantics of value constraints, showing that deciding if an ORM$^-$ model under ORM2 semantics has any full instances is NP-Hard and presenting a test to check if an ORM$^-$ model has the same instances under ORM1 semantics as under ORM2 semantics (Section 7). Finally, we review related work (Section 8) and discuss plans for future work (Section 9). In the sequel, we distinguish between ORM1 and ORM2 only in contexts where the distinction matters.

2. OVERVIEW OF ORM FEATURES

In the interest of space, we limit this overview to features of ORM belonging to ORM$^-$ or relating to our extensions. We now briefly introduce the basic elements of all ORM models (Sec. 2.1), the constraints supported by ORM$^-$ (Sec. 2.2), and two additional constraints, which modelers have found useful and which are the focus of the extensions in this paper (Sec. 2.3).

2.1 Basic ORM Elements

An ORM [12] model represents a business domain in terms of *object types*, which classify business objects; *fact types*, which classify facts that relate these objects; and *constraints* over the objects and facts that belong to the respective types. Figure 1 shows a fragment from an ORM model of a project management system, which we use as a running example. Rounded rectangles depict object types, e.g., Employee and StartDate. A sequence of one or more adjacent squares, called *roles*, depicts a fact type. A solid line connects each role in a fact type to the type whose objects may play that role in any set of facts of that type. The meaning of a fact type is indicated by its *reading text*, which is written next to its role sequence, e.g., Employee reports to Manager. We use M, R, F, and O in the sequel to range over ORM$^-$ models, roles, fact types, and object types, respectively, and X to range over generic types, i.e., fact types and object types. These and other symbols, defined in the sequel, may appear with or without subscripts.

An object type denotes either a set of *values*, in which case it is called a *value type* and drawn with a dashed border, or a set of *entities*, in which case it is called an *entity type* and drawn with a solid border. Intuitively, a value is a simple self-identifying object, e.g., the number 16; whereas an entity is an opaque object that is identified by reference to one or more values, e.g., the employee (entity of type Employee) identified by employee number 16. When
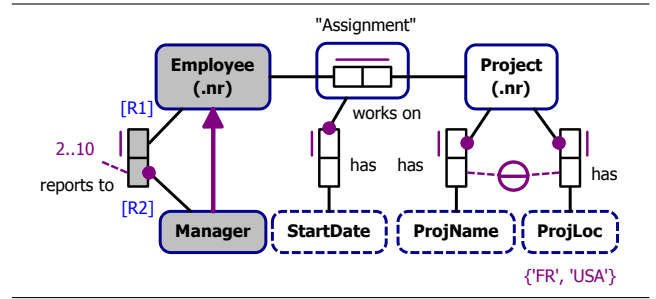

Figure 1: An example of an ORM model.

necessary to distinguish between entity types and value types, we use E and V, respectively, to range over the former and the latter.

Every entity type must be given a reference scheme, which explains how its entities are identified. A *simple reference scheme* uses an injective binary fact type from the entity type to another object type, called the *identifying type*, for this purpose. To reduce clutter in diagrams, a simple reference scheme is often abbreviated by annotating the entity type's shape with a parenthesized *reference mode*. The reference mode stands for both the identifying type and the injective fact type, with the name of the identifying type determined from the reference mode by notational convention. For example, the Employee entity type in Figure 1 has the reference mode "(.nr)". The reference mode, in this case, stands for a value type named Employee_nr, which serves as the identifying type, and an injective binary fact type from Employee to Employee_nr. This fact type signifies that each Employee is uniquely identified by a value of type Employee_nr. Section 2.3 describes *compound reference schemes*, which use facts from multiple fact types to identify an entity.

An instance of an ORM model maps each object type and fact type in the model to a *population*, where a population of an object type is a set of objects (values or entities) and a population of a fact type is a set of tuples of objects. A tuple belonging to the population of a fact type is referred to as a *fact*. We use I to range over model instances, which we assume are type correct according to the typing constraints of ORM [12]. If I is an instance of a model containing object type O and fact type F, then $I(O)$ and $I(F)$ denote, respectively, the populations of O and F in I. We also adopt a convenient shorthand when referring to members of a population, using the type name as a prefix. For example, we use "an O-object" to mean "an object in the population of O."

Adornments on object types, fact types, and roles express constraints on the *legal* instances of a model, where an instance of a model is legal if it satisfies all of the model's constraints. When generating test data, we often want a fully populated instance—i.e., an instance that assigns a non-empty population to each object type and fact type. In keeping with [2, 13], we therefore refer to an ORM model as *consistent* when it has at least one fully populated legal instance[1]. In the sequel, we use C to range over constraints. Additionally, we say C *spans* roles $R_1, \ldots, R_k$, if C applies to the objects that play these (and only these) roles. More generally, we say C *covers* a role (set of roles) if this role belongs to (is a subset of) the roles spanned by C.

2.2 ORM$^-$

Restrictions on the constraints included in the ORM$^-$ subset of ORM were chosen to ensure that a fully populated instance can

---

[1] In [20], the term "satisfiability" is used instead

be generated automatically and efficiently from every consistent ORM⁻ model (under ORM1 semantics) [20]. The constraints supported by ORM⁻ include limited forms of the following six kinds of ORM constraints.

Simple mandatory constraints. A simple mandatory constraint spans a single role. It is drawn as a dot on the line that connects the role to the type of its role players. When present, it requires each object in the role players' type to play that role in at least one fact. For instance, the mandatory constraint on the second role of Employee reports to Manager requires that each manager has at least one employee reporting to her.

Implied mandatory constraints. The need for implied mandatory constraints arises because we seldom want to include *independent objects*, or objects that do not participate in any roles, in the populations of object types. By default, therefore, each object type O has an implied mandatory constraint, which spans all roles that can be played by O-objects. This constraint requires that each O-object plays a role in at least one fact of some fact type. For example, Figure 1 requires that each project name applies to some project. To override this default, a modeler explicitly designates an object type as possibly containing independent objects by attaching an exclamation point to the end of the object name. An object type that is so designated is called an *independent type*. None of the object types in Figure 1 are independent types.

Frequency constraints. A frequency constraint is drawn as an annotation of the form '$i..j$' and connected by a dotted line to the roles that it spans. This constraint requires the number of facts containing any given combination of objects from these roles to lie in the specified range. For example, the frequency constraint '2..10' on the second role of Employee reports to Manager states that each manager who has any employee reporting to her has between 2 and 10 such employees. Although, in general, frequency constraints may span roles in multiple fact types and multiple frequency constraints may cover some of the same roles of a fact type, ORM⁻ precludes both of these situations. Thus, in an ORM⁻ model, each frequency constraint spans one or more roles from the same fact type and the sets of roles spanned by different frequency constraints do not overlap.

Internal uniqueness constraints. An internal uniqueness constraint spans one or more roles of a fact type, and is drawn as a line above the spanned roles. This constraint states that each combination of role players for the spanned roles occurs in at most one fact of the fact type. For example, the internal uniqueness constraint on the first role of Employee reports to Manager in Figure 1 requires that each employee reports to at most one manager. We say an internal uniqueness constraint is *spanning* if it spans all the roles of a fact type. For instance, the constraint on the roles of Employee works on Project is spanning. A uniqueness constraint abbreviates a frequency constraint of the form "1..1".

Subtype constraints. A *subtype constraint* is drawn as a solid arrow from one object type (the subtype) to another (the supertype). For example, Figure 1 indicates that every manager is also an employee. An object type may have multiple supertypes, but they must all share a common ancestor type.

Value constraints. A *value constraint* applies to a single object type and is drawn using set notation alongside the type's shape. In ORM1, this constraint restricts the type's population to be a subset of this set. For example, the value constraint on ProjLoc in Figure 1 asserts that the population of project location values is a subset of {'FR′, 'USA′}. The ORM⁻ algorithms assume ORM1 semantics for value constraints. ORM2 introduces subtle changes to their semantics.

2.3 Some Useful Features Not in ORM⁻

Modelers at LogicBlox now routinely use two other kinds of constraints more frequently than at the time of the study reported in [20]. ORM⁻ does not support either of these constraints.

External uniqueness constraints. An external uniqueness constraint generalizes an internal uniqueness constraint to span roles in multiple fact types $F_1, \ldots, F_k$, where $k \geq 2$. It is drawn as a circle with either one or two horizontal lines inscribed in its center and connected by dashed lines to the roles that it spans. Our extensions build on a restricted form of external uniqueness constraints, in which one role in each F_i is not covered, and the types of the role players of the uncovered roles of F_i and F_j are type compatible, for $1 \leq i < j \leq k$. The constraint restricts the set of *compound* facts obtained by joining the fact type populations on the uncovered roles. Specifically, it requires that any given combination of objects from the spanned roles appears in at most one such compound fact. For example, Figure 1 contains an external uniqueness constraint spanning the second role of Project has ProjLoc and the second role of Project has ProjName. The roles not covered by the constraint are type compatible, as both are played by objects of type Project. Each compound fact produced by joining the populations of these fact types contains a project, and the location and name of that project. Thus, the constraint effectively says that no distinct projects have the same project location and project name. When drawn with two horizontal bars, rather than one, an external uniqueness constraint defines a *compound reference scheme* for an entity type. Other constraints that must also be present in this case are discussed in Section 5.

Objectification. An objectification relates a fact type, called the *objectified type*, to an object type, called the *objectifying type*. It is drawn by inscribing the objectified fact type inside a rounded rectangle and writing the objectifying object type's name above the rectangle in quotes. Objectification allows a modeler to classify facts about other facts, by having objects of the objectifying type stand for facts of the objectified type as role players in other fact types. An objectified type is roughly analogous to an association class in the UML [10]. For example, the fact type Employee works on Project in Figure 1 is objectified as Assignment, an objectifying type. Each Assignment object stands for the fact that a particular employee works on a particular project. A fact in Assignment has StartDate records the date on which that employee started working on that project. The mandatory and uniqueness constraints on the first role of Assignment has StartDate stipulate that, for each employee and project that the employee works on, exactly one start date is recorded.

To permit efficient analysis of a larger class of ORM models, we extend the ORM⁻ algorithms to support objectification and a limited class of external uniqueness constraints, which includes specifications of compound reference schemes. Prior to introducing the ORM⁻ algorithms and explaining how we extend them, however, we report findings of a case study to assess the anticipated impact of these extensions.

3. IMPACT OF OUR EXTENSIONS

Smaragdakis, Csallner, and Subramanian chose the specific features to include in ORM^- based on an empirical study of models developed at LogicBlox, a company that uses ORM to construct enterprise decision-analytics software. Since that study, LogicBlox developers have been creating ORM2 [11] models using Norma [7], a mature ORM modeling tool that supports the modeler in finding and specifying more complex constraints during conceptual modeling. Consequently, newer models developed at LogicBlox feature ORM2 constraints that are not supported in ORM^-. To update the findings of [20] and to decide which specific features need to be supported, we conducted a followup study.

We reviewed 28 models, spanning 4 separate projects: a retail forecasting application, a promotion planning application, an insurance policy administration application, and a model-based code generator. These 28 models represent all of the models created with Norma at LogicBlox since the study described in [20]. There is no overlap between the models reviewed in our study and those reviewed in the previous study.

We detail the data collected during this study in a companion report.[2] Collectively, the 28 models contain 1700 object types, 3430 fact types, and 6089 explicit constraints. The vast majority (5919, or 97%) of these constraints are in ORM^-; however, each model contains some ORM features that are not in ORM^-. The most common of these features is objectification: 310 (18%) of the object types in the surveyed models objectify fact types. When counting an objectification, we count the constraints that the objectification implies [10][3] When these implied constraints are counted, the surveyed models contain 790 total constraints outside of ORM^-.

The extensions we describe in the sequel account for 630 (80%) of the constraints outside of ORM^-, including all constraints abbreviated by objectification and 10 out of 15 external uniqueness constraints. Two of the 28 models we studied fall entirely within ORM^- plus our extensions. We also examined the value constraints in the models in order to determine the impact of the change in the semantics of value constraints under ORM2. We found 17 total value constraints on entity types, and all have equivalent semantics under ORM1 and ORM2 interpretations.

4. ORM^- ALGORITHMS

Smaragdakis, Csallner, and Subramanian introduce two algorithms that operate on ORM^- models: one for deciding the consistency of an ORM^- model M and, when M is consistent, another for generating a fully populated legal instance of M [20]. In extending these algorithms, we do not actually modify the algorithms themselves, but introduce wrappers around the original algorithms. In this section, we describe the main idea behind the ORM^- consistency algorithm, which our extension to include objectification builds upon. We also describe the inputs and outputs of the ORM^- population algorithm. We treat this latter algorithm as a black box because correctness of our extensions depend only on its correctness, not on the details of its implementation.

ORM^- consistency algorithm. The consistency problem for ORM^- is reducible to the problem of solving a special class of integer inequalities. The algorithm in [20] encodes the constraints in an ORM^- model M as a system of integer inequalities Q^M over variables that stand for the sizes of various populations in a fully populated legal instance of M. The rules for introducing *population variables* for such a model are as follows.

---

[2] http://www.cse.msu.edu/~ldillon/studyDetails.pdf
[3] See Section 6 for details.

1. For each object type O, generate a variable o representing the cardinality of O's population.

2. For each fact type F, generate a variable f representing the cardinality of F's population.

3. For each role R belonging to a fact type F, generate a variable r representing the number of distinct objects that play R in F's population.

4. For each non-spanning frequency constraint C on a fact type F, generate a variable c representing the cardinality of the projection of F's population on the roles spanned by C.[4]

We use the symbols in these rules when referring to specific kinds of population variables, and use x and y when referring to generic population variables. A *solution* α to Q^M is a function from population variables to natural numbers such that each inequality in Q^M is true after replacing each population variable x with its value $\alpha(x)$.

Smaragdakis, Csallner, and Subramanian [20] provide rules for constructing Q^M for an ORM^- model M, and prove that Q^M has a solution if and only if M is consistent. Our extensions depend only on the correctness of these rules, not on their specific definitions. For this reason, we do not reproduce the full rules here; instead, we show a representative set of inequalities. The inequalities in Table 1 are generated from the shaded portion of Figure 1. Here, variables o_e, o_m, and f represent the cardinalities of the sets of Employee-objects, Manager-objects, and Employee reports to Manager-facts, respectively; r_e and r_m denote the cardinality of the projection of the fact type's population onto the fact type's first and second roles, respectively; and c_{uc} and c_{fc} denote the cardinality of the projection of the fact type's population onto the roles spanned by the internal uniqueness constraint and the frequency constraint, respectively.[5] Besides these inequalities, the ORM^- consistency algorithm generates an inequality of the form $1 \leq x$, for every population variable x, to require that all the populations are non-empty.

In summary, given an ORM^- model M as input, the ORM^- consistency algorithm generates Q^M. It then applies a decision procedure and either returns a solution α to Q^M or indicates that Q^M is unsatisfiable. The proof that this algorithm is polynomial in the model's size rests on two results from [20]. First, the size of Q^M is polynomial in the size of M. Second, the time to solve Q^M is polynomial in the size of Q^M. This latter result holds because of the form of the inequalities in Q^M. Specifically, each inequality fits one of the following two forms:

$$i \cdot x \leq j \cdot y_1 \cdot y_2 \cdot \ldots \cdot y_n$$
$$i \cdot x \leq j + y_1 + y_2 + \ldots + y_n$$

where i, j, n are non-negative integers and x, y_i are integer-valued (population) variables. Table 2 depicts an example output of the ORM^- consistency algorithm given the shaded portion of Figure 1 as input.

---

[4]i.e. the number of tuples in the projection. No variable is created for a spanning uniqueness constraint (the only form of spanning frequency constraint permitted in ORM), because such a variable would be redundant with the variable f created by rule 2 for F's population.

[5]Because the internal uniqueness constraint and the frequency constraint in this example are both unary, the variables c_{uc} and c_{fc} are "redundant" with variables r_e and r_m. Both types of variables are needed, however, if a constraint applies to multiple roles. We show the inequalities that are generated by the ORM^- algorithm, which is designed for the general case.

Table 1: Inequalities: Shaded portion of Fig. 1

Inequality			Generated from
r_e	$\leq$	o_e	Employee role
r_e	$\leq$	f	
r_m	$\leq$	o_m	Manager role
r_m	$\leq$	f	
o_m	$\leq$	r_m	Mandatory constraint
f	$\leq$	$c_{fc} \cdot c_{uc}$	Emp. reports to Mgr.
o_m	$\leq$	o_e	Subtype relation
f	$\leq$	$10 \cdot c_{fc}$	Freq. constraint
$2 \cdot c_{fc}$	$\leq$	f	
c_{fc}	$\leq$	r_m	
r_m	$\leq$	c_{fc}	
f	$\leq$	c_{uc}	Uniqueness constraint
c_{uc}	$\leq$	f	
c_{uc}	$\leq$	r_e	
r_e	$\leq$	c_{uc}	

Table 2: Solution to inequalities: Shaded portion of Fig. 1

x	$\alpha(x)$	x	$\alpha(x)$	x	$\alpha(x)$
o_e	3	r_e	2	c_{uc}	2
o_m	1	r_m	1	c_{fc}	1
f	2				

ORM$^-$ population algorithm. The ORM$^-$ population algorithm takes as input an ORM$^-$ model M and a solution α to Q^M; and produces as output a legal instance I of M for which the cardinalities of the object and fact type populations equal the values that α assigns to their corresponding variables [20]. It generates I in time linear in the number of objects and facts in I, which may be exponential in the size of the model M. This runtime behavior is optimal in the sense that any algorithm for generating an instance of M must at least enumerate each object and fact in the instance. While the solution α given as input to the population algorithm is typically that produced by the ORM$^-$ consistency algorithm, any solution to Q^M can be used. Table 3 depicts an example output of the ORM$^-$ population algorithm given the solution in Table 2 as input, where o_i denotes an object, $1 \leq i \leq 3$. Thus, the instance described by this table is a fully populated legal instance for the shaded portion of Figure 1.

Table 3: Full legal instance: Shaded portion of Fig. 1

X	$I(X)$
Employee	$\{\, o_1, o_2, o_3 \,\}$
Manager	$\{\, o_1 \,\}$
Employee reports to Manager	$\{\, (o_2, o_1), (o_3, o_1) \,\}$

5. EXTENSION: EXTERNAL-UNIQUENESS PATTERN

Our empirical analysis suggests that, in practice, most uses of an external uniqueness constraint conform to a pattern that can be expressed in ORM$^-$. Figure 2 depicts this pattern. For convenience, names for the roles covered by the constraint are shown using adornments of the form "[R$_i$]." More precisely:
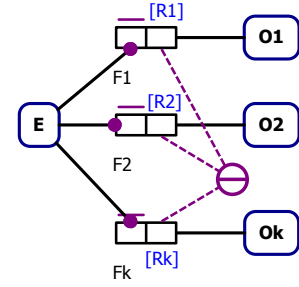

Figure 2: The ExtUC$^-$ pattern

DEFINITION 1. *An external uniqueness constraint C, spanning roles $R_1, \ldots, R_k$, where $k \geq 2$, belongs to ExtUC$^-$ if the following two additional conditions hold:*

1. *For each $i \in \{1 \ldots k\}$, the constraint C is the only uniqueness constraint that covers R_i and no frequency constraint covers R_i.*

2. *There exist an entity type E and k fact types $F_1, \ldots, F_k$, all binary, such that, for each $i \in \{1, \ldots, k\}$, R_i is one of the roles in F_i and the other role is: played by entities of type E, mandatory, covered by a unary internal uniqueness constraint, and not covered by any additional constraints.*

This pattern is of special interest because modelers often use it to define a compound reference scheme: The conditions in the definition ensure that E-entities are uniquely identified by collections of facts, one from each of the fact type populations.[6] In the sequel, we use Γ when referring to a set of constraints from ExtUC$^-$, M' when referring to an ORM model that is an ORM$^-$ model extended with constraints supported by our extensions, and I' when referring to an instance of M'.

We use a transformation, called *absorption*, to rewrite an external uniqueness constraint in ExtUC$^-$, together with the fact types that it applies to, into an *absorption* fact type and a set of ORM$^-$ constraints.[7]

DEFINITION 2. *Let constraint $C \in ExtUC^-$ be as in Definition 1. Absorption replaces C and the set $\{\, F_1, \ldots F_k \,\}$ of fact types containing the roles $R_1, \ldots, R_k$ spanned by C with the following:*

- *An* absorption *fact type F consisting of $R_1, \ldots, R_k$ and a new role, R, played by entities of the type E specified in Definition 1.*

- *An internal uniqueness constraint that spans $R_1, \ldots, R_k$.*

- *An internal uniqueness constraint and a mandatory constraint that each spans R.[8]*

---

[6]More formally, for a given collection of objects $o_1, o_2, \ldots, o_k$ of types $O_1, O_2, \ldots O_k$ respectively, there exists at most one entity e for which $(e, o_1), (e, o_2), \ldots, (e, o_k)$ are in the populations of fact types $F_1, F_2, \ldots, F_k$ respectively.

[7]This transformation is similar to the relational mapping procedure of [12][p. 499], which groups the fact types in Figure 2 to a single table, thereby absorbing the roles.

[8]ORM methodology warns against specifying uniqueness constraints that span fewer than $k - 1$ roles of a k-ary fact type on grounds that they conceal elementary fact types; but the meaning is completely well defined and the transformation produces a model in ORM$^-$.

In addition, any simple mandatory constraints that applied to any of the absorbed roles, i.e., to any of the R_i, are applied accordingly to roles in the absorption fact type.
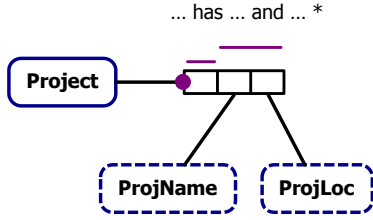

Figure 3: Absorption fact type for the external uniqueness constraint in Figure 1

Figure 3 shows the result of applying absorption to the relevant portion of the example model in Figure 1. The new absorption fact type, Project has ProjName and ProjLoc, and three associated constraints replace the two binary fact types Project has ProjName and Project has ProjLoc, the simple mandatory and internal uniqueness constraints that covered roles of these fact types, and the external uniqueness constraint from Figure 1.

THEOREM 1. *If M' is an ORM$^-$ model extended with a set Γ of constraints in ExtUC$^-$, and M is the ORM$^-$ model obtained by applying absorption to M', then M is consistent if and only if M' is consistent.*

PROOF. Consider the "only-if" direction: Suppose that M is consistent, and let I be a legal instance of M. Construct an instance I' of M' as follows: First, let $I'(O) = I(O)$ for each object type O. Next, let $I'(F) = I(F)$ for each fact type F that is not affected by absorption. Finally, for each absorption fact type F in M created by absorbing roles $R_1, \ldots, R_k$ in M': let $I'(F_i) = \pi_{R,R_i}(I(F))$, where $F_1, \ldots, F_k$ are the fact types in M' containing $R_1, \ldots R_k$, respectively; R is the new role introduced (first bullet of Def. 1) in creating F; and π_{R,R_i} denotes projection on roles R and R_i. It remains to show that I' is a legal instance of M'. We elide the full details of the proof for brevity. The key idea is that, for each absorption fact type F in M, the instance I' can be shown to satisfy the external uniqueness constraint spanning the roles $R_1, \ldots, R_k$ in the fact types $F_1, \ldots, F_k$ that are absorbed into F, as well as the mandatory and uniqueness constraints on the roles in $F_1, \ldots, F_k$ not covered by the external uniqueness constraint by virtue of the assumption that I satisfies the constraints in M that cover roles of F.

The "if" direction of the proof follows by similar reasoning. In this case, given a legal instance I' of M', we define an instance I of M by taking: $I(O) = I'(O)$ for each object type O; $I(F) = I'(F)$ for each fact type F that is not affected by absorption; and $I(F) = \{(e, o_1, \ldots, o_k) \,|\, \forall i : 1 \leq i \leq k \cdot (e, o_i) \in I'(F_i)\}$ for each absorption fact type F, where $F_1, \ldots, F_k$ are as defined above. For brevity, we elide details of the proof that I is a legal instance of M. $\square$

The next theorem follows easily from the observation that the "only-if" direction of the previous proof sketches a linear-time procedure for mapping a legal instance of a model produced by absorption to a legal instance of the original model.

THEOREM 2. *If M' is an ORM$^-$ model extended with a set Γ of constraints in ExtUC$^-$, and M is the ORM$^-$ model obtained by*

applying absorption to M', *then any legal instance I of M can be used to construct a legal instance I' of M', in time that is linear in the size of I.*

In summary, Theorem 1 suggests an efficient algorithm for checking consistency of an ORM model M' produced by extending an ORM$^-$ model with constraints in ExtUC$^-$: Reduce it to an equivalent ORM$^-$ model M and run the ORM$^-$ consistency algorithm with M as input. Additionally, if M' is consistent, Theorem 2 suggests a method to populate it: Run the ORM$^-$ population algorithm with M and a solution to Q^M as inputs to produce a fully populated legal instance I of M. Then, transform this instance back into a fully populated legal instance I' of M'. The time to transform M' is linear in the size of the model, while the time to map the generated population back to a population of M' is linear in the size of the generated population.

6. EXTENSION: OBJECTIFICATION

As discussed previously, objectification allows objects of an object type, i.e, the objectifying type, to stand for facts in a fact type, i.e., the objectified type, in order that the objectified facts can be viewed as playing roles in facts of yet other types. ORM2 defines the semantics of objectification in terms of more elementary constructs: Briefly, an objectification of fact type F as object type O abbreviates a set of implicit *link* fact types and constraints on those fact types, which collectively imply the existence of a bijection between the populations assigned to F and O by any legal instance of the model [10]. This bijection is not stored directly in the instance but can be recovered from the populations that the instance assigns to the link fact types. Conversely, given this bijection, we can recover the populations of the link fact types. Because the link fact types and link constraints are implied, and not shown by the modeling tool, they cannot be further constrained, modified, or used. Thus, modelers conventionally think of an objectification more abstractly, in terms of the implied bijection, rather than in terms of populations of link types. For simplicity, we adhere to this conventional view and treat an objectification as implying the existence of a bijective relation between populations of the objectified type and the objectifying type. Because no model element explicitly stands for this relation, when a model is adorned with an objectification, an instance of the model must specify the bijection.

DEFINITION 3. *We represent an objectification as a pair (F, O), where F is the objectified (fact) type and O is the objectifying (object) type.*

We use Ω to range over a set of objectifications. To adorn an ORM model M' with a set of objectifications Ω, we require that all types in the objectifications in Ω belong to M' and no type in M' participates in more than one objectification in Ω. In this case, we write $M' \cup \Omega$ for the extended ORM model.

For simplicity, we impose one additional restriction on objectification: No fact type containing a role that is covered by an external uniqueness constraint is objectified.[9] This restriction permits us to use the techniques of the previous section to transform any ORM model formed by extending an ORM$^-$ model with both constraints in ExtUC$^-$ and objectifications into an equivalent model formed by extending another ORM$^-$ model with objectifications only. In the sequel, therefore, we assume that objectifications are added to an ORM$^-$ model.

We now extend the definitions for instances, legality, and consistency to encompass these extended models.

---

[9]The absorption procedure of the previous section can be modified to remove this restriction.

DEFINITION 4. *Let I' be a mapping with domain M', where $M' = M \cup \Omega$, M is an ORM$^-$ model, and Ω is a set of objectifications.*

- *I' is an instance of M' if I' maps each object type $O \in M$ to a population, $I'(O)$, of objects; each fact type $F \in M$ to a population, $I'(F)$, of facts; and each objectification $(F, O) \in \Omega$, to an objectification relation, $I'(F, O) \subseteq I'(F) \times I'(O)$.*

- *I' is legal if the mapping $M \lhd I'$, which restricts the domain of I' to the types contained in M, is a legal instance of M and if, for each $(F, O) \in \Omega$, the objectification relation $I'(F, O)$ is a bijection from $I'(F)$ to $I'(O)$.*

- *I' is fully populated if $M \lhd I'$ is fully populated.*

- *M' is consistent if there exists a fully popuated legal instance of M'.*

An objectification (F, O) in a model M' implies that a legal instance of M' maps F and O to populations of equal sizes. To extend the ORM$^-$ algorithms with support for objectifications, we encode this constraint as a pair of inequalities.

DEFINITION 5. *Let $M' = M \cup \Omega$, where M is an ORM$^-$ model and Ω is a set of objectifications as in Definition 3. Then $Q^{M,\Omega}$ is the set of inequalities containing, for each $(F, O) \in \Omega$, the inequalities $f \leq o$ and $o \leq f$, where f and o are the population variables used, when constructing Q^M, to stand for the cardinalities of the populations of F and O, respectively.*

We extend the ORM$^-$ consistency algorithm to check a model produced by adorning an ORM$^-$ model with objectifications by adding this new set of inequalities to those generated from the ORM$^-$ model before calling the decision procedure:

ALGORITHM 1.
Input: An ORM$^-$ model M.
    A set Ω of objectifications as in Definition 3.
Output: A solution to $Q^M \cup Q^{M,\Omega}$ or,
    if none exists, an empty map.

1. *Generate Q^M according to the rules in [20].*

2. *Generate $Q^{M,\Omega}$ according to Definition 5.*

3. *Run the algorithm in [20] with $Q^M \cup Q^{M,\Omega}$ as the input and return the output.*

No modifications are necessary to either the rules for computing Q^M or the decision procedure itself.

The next lemma justifies concluding that, if Algorithm 1 returns an empty map, the model $M' = M \cup \Omega$ is inconsistent.

LEMMA 1. *Let $M' = M \cup \Omega$, where M is an ORM$^-$ model and Ω is a set of objectifications as in Definition 3. If M' is consistent, then $Q^M \cup Q^{M,\Omega}$ has a solution.*

PROOF. Assume M' is consistent and let I' be a fully populated legal instance of M'. Define α so that it maps each population variable x in Q^M to the cardinality of $I'(x)$. Because I' is a legal instance of M', then by Definition 4 $M \lhd I'$ is a legal instance of M. To conclude that α is a solution to Q^M, we therefore appeal to [20]. To show that α is a solution to $Q^{M,\Omega}$, consider any arbitrary $(F, O) \in \Omega$ and let f and o be the population variables in Q^M representing the populations of F and O, respectively.

We need to show that α satisfies both $f \leq o$ and $o \leq f$. Because I' is a legal instance of M', it follows from Definition 4 that $I'(F, O)$ is a bijection between $I'(F)$ and $I'(O)$, and therefore that $|I'(F)| = |I'(O)|$. Because $\alpha(f) = |I'(F)|$ and $\alpha(o) = |I'(O)|$ by definition, α satisfies the inequalities. $\square$

In keeping with [20], we establish the converse of the conclusion in Lemma 1, i.e., that M' is consistent if $Q^M \cup Q^{M,\Omega}$ has a solution, by showing how to generate a fully populated legal instance of M' from a solution to $Q^M \cup Q^{M,\Omega}$. The ORM$^-$ algorithm is used to find a fully populated legal instance of M from a solution to $Q^M \cup Q^{M,\Omega}$ and then this instance is extended to become an instance of M':

ALGORITHM 2.
Input: An ORM$^-$ model M.
    A set Ω of objectifications as in Definition 3.
    A solution α to $Q^M \cup Q^{M,\Omega}$.
Output: A fully populated legal instance of $M \cup \Omega$.

1. *Run the ORM$^-$ algorithm from [20] with inputs M and α to produce a fully populated legal instance I of M. Initialize $I' := I$.*

2. *For each $(F, O) \in \Omega$:*
   Let f be the population variable in Q^M for F,
       n be the cardinality of $I(F)$,
       $\{ f_1, \ldots, f_n \}$ be an enumeration of $I'(F)$, and
       $\{ o_1, \ldots, o_n \}$ be an enumeration of $I'(O)$.
   Assign $I'(F, O) := \{ f_i \mapsto o_i \}_{i \in \{1, \ldots n\}}$.

3. *Return I'.*

The next lemma establishes that this algorithm is correct.

LEMMA 2. *Algorithm 2 outputs a fully populated legal instance of $M \cup \Omega$.*

PROOF. We need to show that I' is a fully populated legal instance of M'. From [20, pp. 87–88], we know that: (1) I is a fully populated legal instance of M, and (2) $\alpha(x) = |I(X)|$, for each type $X \in M$, where x is the population variable in Q^M for X.

Because $M \lhd I' = I$, (1) implies that $M \lhd I'$ is a fully populated legal instance of M. Additionally, for each $(F, O) \in \Omega$, because α is a solution to $Q^{M,\Omega}$, Definition 5 implies that $\alpha(f) = \alpha(o)$, where f is the population variable in Q^M for F and o is the population variable in Q^M for O. But (2) and the definitions of I and I' imply $\alpha(f) = |I(F)| = |I'(F)|$ and $\alpha(o) = |I(O)| = |I'(O)|$. Thus, we conclude that $|I'(O)| = |I'(F)|$ and, therefore, that $I'(F)$ and $I'(O)$ can be enumerated as required for step 2. It follows that $I'(F, O)$ is a bijection from $I'(F)$ to $I'(O)$. $\square$

Finally, we reason that the extended algorithms solve the consistency and population problems for an ORM model that extends an ORM$^-$ model with objectifications, and that these algorithms scale:

THEOREM 3. *Let $M' = M \cup \Omega$, where M is an ORM$^-$ model and Ω is a set of objectifications. Then:*

1. *Consistency of M' can be decided in time polynomial in the size of M'; and*

2. *If M' is consistent, a fully populated legal instance I' of M' can be generated in time linear in the size of I'.*

PROOF. The lemmas in this section demonstrate that M' is consistent if and only if the assignment returned by Algorithm 1 is non-empty. Thus, Algorithm 1 solves the consistency problem. Additionally, this result and Lemma 2 demonstrate that Algorithm 2 solves the population problem. The complexity results follow from those in [20] (since the inequalities in $Q^{M,\Omega}$ are of the required form) and analysis of the time to perform the additional steps in our algorithms (i.e., Step 2 of both algorithms). We elide the details as the argument is straightforward. □

7. ORM2 VALUE CONSTRAINTS

ORM2 introduces subtle changes to value constraint semantics, which unfortunately increase the difficulty of consistency checking and test data generation. A value constraint may apply to either a value type or an entity type. In the case of a value type, the ORM1 semantics and ORM2 semantics are identical: the constraint restricts the values in a population of the type. In the case of an entity type, however, the semantics differ.

Figure 4 illustrates the change in semantics. It shows a model containing value constraints on two entity types, Project and Person (Fig. 4(a)). The reference mode (ID) on each entity type abbreviates a binary fact type and three constraints, which define a simple referencing scheme for that entity type. The implicit fact types relate each Project to a unique ID and each Person to a unique ID; moreover, the ID serves to identify the entity in both fact types. A value constraint may apply to an entity type only if that type has a simple reference scheme in which the identifying type is a value type. The semantics of the value constraint is then defined in terms of the reference scheme.

In ORM1, a value constraint on an entity type directly restricts the values of the identifying type. If multiple entity types with value constraints have the same identifying type, an instance has to satisfy each of the value constraints to be legal. Thus, the constraints effectively restrict the values of the identifying type to lie in the intersection of their ranges. For example, under ORM1 semantics, the two value constraints restrict identifiers to be in the range from 5 to 10 (Fig. 4(b)).

ORM2 redefines the semantics of value constraints on entity types in terms of *role value constraints* (RVCs). A RVC, drawn as a value constraint that is connected by a dotted line to a role, restricts the objects that may play that role. Critically, it does not directly restrict the population of the role players' type.[10] Figure 4(c) illustrates: Because ID is non-independent, each ID-value must identify a project or a person (or both). Thus, an ID value must satisfy $\{1..10\}$ or $\{5..15\}$. The effect of the value constraints is thus to restrict values of the identifying type to lie in the union of the constraints' ranges.

This example shows that a model with value constraints may have instances that are legal under ORM2 semantics, but not under ORM1 semantics. Thus, the ORM$^-$ consistency algorithm, which is sound and complete for ORM1 semantics, may yield false negatives for ORM2 semantics: A full legal instance of an ORM$^-$ model M may exist under the ORM2 semantics, even if Q^M has no solutions.

THEOREM 4. *Under the ORM2 interpretation, the problem of deciding consistency of an ORM$^-$ model with value constraints on both entity and value types is NP-hard.*

We prove this theorem by reduction to the following *set covering problem* [16]:

---

[10]unless the role is mandatory, in which case the RVC effectively applies to every entity of the associated type.

DEFINITION 6. *Given a finite universe U, a finite set $S = \{S_1, \ldots, S_n\}$ of subsets whose union equals U, and a positive integer $k < n$, the set cover problem determines if there exists a set C containing exactly k sets from S whose union also equals U.*

A collection of sets whose union equals the universe is called a *set cover*. Thus, the problem can be rephrased as asking whether the given set cover S contains a set cover C of a size k, for a given $k < |S|$.

For example, let $U = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, and let $\mathbf{S} = \{S_1, S_2, S_3, S_4, S_5\}$, where

$$
\begin{aligned}
S_1 &= \{a_1, a_2, a_6\} & S_2 &= \{a_2, a_4, a_5\} \\
S_3 &= \{a_3, a_4, a_6\} & S_4 &= \{a_3, a_4\} \\
S_5 &= \{a_1, a_3, a_5\}
\end{aligned}
$$

Then there is no set cover of size 2. However, $C = \{S_1, S_4, S_5\}$ is a set cover of size 3.

We encode an instance of the set covering problem as an ORM$^-$ model containing value constraints under the ORM2 semantics as follows:

DEFINITION 7. *Given an instance of the set cover problem with universe U, a set cover $S = \{ S_1, \ldots, S_n \}$, and $k < n$, the ORM$^-$ encoding of this problem consists of:*

1. *A value type, C, with the value constraint $\{1 .. n\}$.*

2. *A value type, X, with the value constraint $\{1\}$.*

3. *A fact type, X has C, whose first role is mandatory and subject to the frequency constraint, k,[11] and whose second role is both mandatory and unique.*

4. *For each atom $a_i \in U$, $1 \le i \le |U|$, an entity type Ai with simple reference scheme C and a value constraint enumerating the indices j such that $a_i \in S_j$.*

The key idea behind this construction is that the C-values in a full legal instance identify the indices of a set cover of size k. For example, Figure 5 shows the encoding (left) of the set cover problem given earlier in the section with $k = 3$, together with a full legal instance (right) of that model, in which the populations of the last five fact types define simple referencing schemes for the entity types, as required by the reference modes. The C-values of this instance identify the sets from S in the set cover of size 3 presented earlier in this section.

PROOF OF THEOREM 4. Given an instance of the set covering problem, as described in Definition 6, and the encoding of this problem as an ORM$^-$ model, as described in Definition 7, we show that a full legal instance of the ORM$^-$ model under ORM2 semantics determines a set cover of cardinality k and, conversely, that a set cover of cardinality k determines a full legal instance of this model.

Consider a set cover C. We use C to construct an instance I of the ORM$^-$ model as follows:

1. Let $I(C) = \{i \mid S_i \in C\}$; $I(X) = \{1\}$; and $I(X \text{ has } C) = \{(1, i) \mid S_i \in C\}$.

2. For each i and j such that $a_i \in S_j$ and $S_j \in C$, create a distinct object o_{ij} and let $I(A_i) = \{o_{ij} \mid a_i \in S_j \wedge S_j \in C\}$ and $I(A_i \text{ has } C) = \{(o_{ij}, j) \mid a_i \in S_j \wedge S_j \in C\}$.
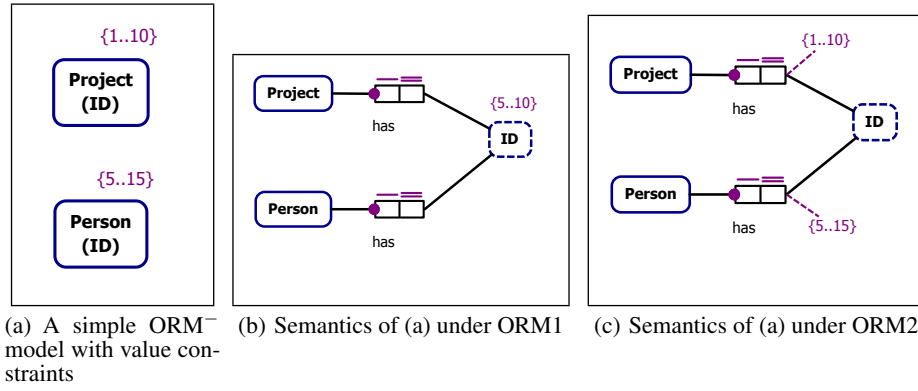
---

[11]k abbreviates k..k

(a) A simple ORM⁻ model with value constraints

(b) Semantics of (a) under ORM1

(c) Semantics of (a) under ORM2

Figure 4: Illustration of ORM1 vs. ORM2 semantics of value constraints

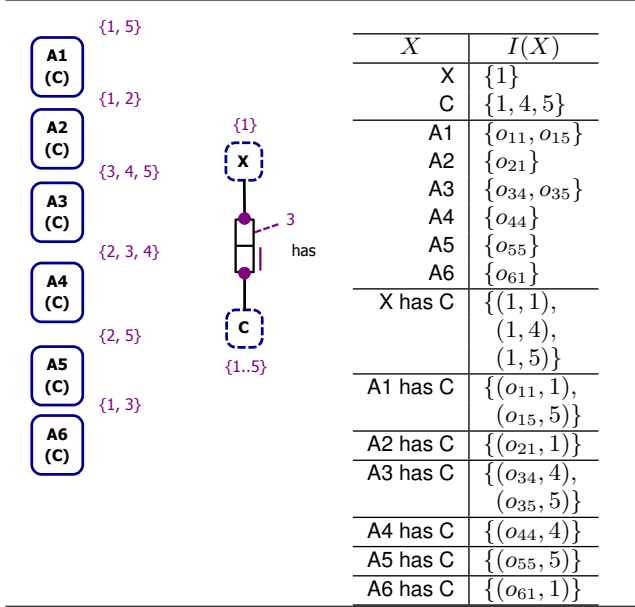| X	$I(X)$
X	$\{1\}$
C	$\{1, 4, 5\}$
A1	$\{o_{11}, o_{15}\}$
A2	$\{o_{21}\}$
A3	$\{o_{34}, o_{35}\}$
A4	$\{o_{44}\}$
A5	$\{o_{55}\}$
A6	$\{o_{61}\}$
X has C	$\{(1, 1),$ $(1, 4),$ $(1, 5)\}$
A1 has C	$\{(o_{11}, 1),$ $(o_{15}, 5)\}$
A2 has C	$\{(o_{21}, 1)\}$
A3 has C	$\{(o_{34}, 4),$ $(o_{35}, 5)\}$
A4 has C	$\{(o_{44}, 4)\}$
A5 has C	$\{(o_{55}, 5)\}$
A6 has C	$\{(o_{61}, 1)\}$

Figure 5: Example set cover problem reduced to an ORM⁻ consistency problem (left) and a full legal instance (right)

Step 1 produces non-empty populations that satisfy the mandatory and uniqueness constraints on both roles of X has C and, as the cardinality of C is k, that also satisfy the frequency constraint on the first role. Additionally, step 2 produces a non-empty population for each Ai and a reference scheme for each that meets Ai's value constraint.

For the converse, consider a full legal instance I for the ORM⁻ model. Then the set $C = \{S_i \mid i \in I(\mathsf{C})\}$ is a set cover of cardinality k: The value constraint on C ensures that $C \subseteq S$; the frequency constraint, together with the value constraint on X, ensures that C has cardinality k; and, because each $I(\mathsf{A}i)$ contains at least one entity and this entity is identified by some $i \in I(\mathsf{C})$, the union of the sets $S_i, i \in I(\mathsf{C})$, contains U.

Because (1) the set covering problem is NP-complete, (2) the ORM⁻ model that encodes an instance of this problem is consistent under ORM2 semantics if and only if there is a set cover of the required size, and (3) the size of the encoding is polynomial in the size of the original problem, we conclude that determining consistency of ORM⁻ models under ORM2 semantics is NP-hard. $\square$

Although the semantics of value constraints in ORM1 and ORM2 differ, many ORM⁻ models have exactly the same legal instances under both interpretations. In fact, a good use case for distinguishing between the two semantics is hard to imagine. A case such as Figure 4, in which the two semantics result in different models, most likely signals a poor modeling decision. For instance, best practice would call for two distinct value types, e.g., perID for identifying persons and prjID for identifying projects, in place ID in Figure 4(a) .[12]

The next theorem establishes conditions on the use of value constraints that guarantee an ORM⁻ model has the same meaning under ORM1 as under ORM2.

THEOREM 5. *Let M be an ORM⁻ model and $E_1 \ldots E_n$ be the entity types in M that are covered by value constraints, if any. In addition, for $1 \leq i \leq n$, let F_i be the injective binary fact type that defines the referencing scheme for E_i, let V_i be the identifying type for E_i, and let R_i be the role in F_i played by V_i. If each V_i plays no other roles in M and if either V_i is non-independent or R_i is mandatory (or both), then M has the same legal instances under ORM1 and ORM2 semantics.*

PROOF. The conditions imposed by the theorem guarantee that, under either semantics, the reference scheme for each E_i is a bijection—that is, in any legal instance, each E_i-entity is identified by a unique V_i-value and, conversely, every V_i-value identifies a unique E_i-entity. This observation implies that, for a legal instance of M, the population of V_i is contained in a given set of values if and only if the projection of F_i's population on R_i is contained in the same set of values. As the ORM1 interpretation of the value constraint on E_i requires the former and the ORM2 interpretation of this same constraint requires the latter, we conclude that the legal instances are the same under both interpretations. $\square$

Thus, if an ORM⁻ model either has no value constraints on any of its entity types or those that it does have satisfy the conditions of this theorem, the ORM⁻ algorithms can be used regardless of which version of the semantics are intended. Every model in our case study (Sec. 3) meets these conditions, and so contain the same legal instances under both ORM1 and ORM2 semantics.

8. RELATED WORK

Much of the existing work on analyzing consistency of data models looks at models described using ER diagrams [6] or UML class

---

[12]The authors thank an anonymous referee for this observation.

models [9]. Artale et al. [1] develop a complexity hierarchy for increasingly expressive Extended ER diagrams [8]. An EER subset called EER$_\text{ref}$, for which consistency can be decided in polynomial time, resembles ORM$^-$ in its feature set: EER$_\text{ref}$ supports multiplicity and cardinality constraints on relationships, which are similar to mandatory and frequency constraints in ORM$^-$, and ISA relationships between entities, which are similar to subtype relationships in ORM$^-$. EER$_{ref}$ also supports disjointness constraints on entities; these are similar to ORM exclusion constraints on subtype relationships, which are not in ORM$^-$. This result suggests that it may be possible to extend ORM$^-$ with limited support for exclusion constraints, without sacrificing polynomial-time performance. Hartmann [13] shows that the consistency of a set of cardinality constraints and functional dependencies can be decided in polynomial time when the constrained EER diagram is in Boyce-Codd Normal Form (BCNF). ORM models are always in BCNF by construction. Because internal uniqueness constraints represent functional dependency, Hartmann's result suggests that it might be possible to extend ORM$^-$ to overlapping internal uniqueness constraints, as conjectured in [20]. Zamperoni and Löhr-Richter [21] describe a method of checking consistency of an EER diagram by generating a system of linear inequalities that express constraints on the cardinalities of the sets in the diagram. However, the systems of inequalities that they generate cannot be solved efficiently in general. Berardi et al. [2] prove that deciding the consistency of UML class diagrams is EXPTIME-complete, even when diagrams are restricted to include just binary associations, multiplicities of the form 0..* and 1..*, disjointness constraints and covering constraints on generalization relationships.

Existing work on generating sample populations from data models has looked at models described using EER diagrams or SQL. Neufeld *et al.* [19] generate finite models of EER diagrams that are subject to consistency constraints expressed as logical formulas. Their method is sound but not complete, and employs various heuristics, as well as interaction with the user. Chays *et al.* [5] and Houkjær *et al.* [14] present methods of generating test data for relational schemas expressed in SQL DDL. Both methods generate data that respect primary and foreign key constraints, uniqueness constraints, and not-null constraints in the schema. In addition, both methods may be customized by the user to generate data with statistical properties that reflect real-world data, a direction that we intend to explore for our work with ORM.

The algorithm for generating test data from an ORM$^-$ model is useful for black-box testing, in which the details of the program under test are not known. Alternatively, *query aware* methods for generating test databases take a white-box approach, generating test data for a program by analyzing the queries that a program executes. Query-aware methods use a variety of techniques, including constraint solving [17, 22], symbolic execution [4], and operationalization of query expressions [3].

9. SUMMARY AND FUTURE WORK

As ORM modeling tools mature, modelers are increasingly utilizing ORM features that are not covered by the ORM$^-$ subset identified in [20]. We have shown how to extend ORM$^-$ to support objectification, the most commonly used ORM feature outside ORM$^-$. Our extensions also support restricted forms of external uniqueness constraints, which are used by developers for defining compound reference schemes. Finally, we have shown that although a change in the semantics of value constraints in ORM2 renders the ORM$^-$ algorithms unsound, an efficient check guarantees they are sound for the vast majority of ORM$^-$ models created in practice. Our extensions preserve and improve the ability to effi-

ciently generate legal populations of large ORM models. In future work, we plan to identify conditions that also allow use of equality constraints and a broader class of frequency constraints.

All but two models in our study include small numbers of *exclusion* constraints, which are known to be NP-Hard, as well as *subset* and *ring* constraints, which are believed to be at least NP-Hard. To address these remaining constraints, we will attempt to integrate our previous approach based on constraint solving [18] with the ORM$^-$ algorithm. We can apply the constraint solving technique to find populations of minimal, disjoint sub-models that collectively contain all non-ORM$^-$ constraints. Our study suggests that these sub-models tend to be small, in which case, we can find such populations quickly. We hope to modify the algorithm in this paper to generate a population of the remainder, starting from the sub-populations.

A final direction for future research was alluded to earlier. To generate data sets that reflect real-world data, we plan to investigate approaches that attempt to fit a statistical profile supplied by a user when generating instances. This capability will also be useful for producing multiple data sets with different characteristics for testing a database application.

10. REFERENCES

[1] A. Artale et al. Reasoning over extended ER models. In *Proc. 26th International Conference on Conceptual Modeling*, pages 277–292. Springer-Verlag, 2007.

[2] D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

[3] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. *Proc. International Conference on Data Engineering*, pages 506–515, 2007.

[4] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: Generating query-aware test databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 341–352, 2007.

[5] D. Chays et al. An AGENDA for testing relational database applications. *Software Testing, Verification and Reliability*, 14(1):17–44, 2004.

[6] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[7] M. Curland and T. Halpin. Model driven development with NORMA. In *Proc. 40th Hawaii International Conference on System Sciences*, page 286a, 2007.

[8] G. Engels et al. Conceptual modeling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(2):157–204, 1992.

[9] M. Fowler and K. Scott. *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[10] T. Halpin. Objectification. In *Proc. 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design*, volume 5, pages 106–123, 2005.

[11] T. Halpin. ORM 2. In *On the Move to Meaningful Internet Systems: OTM Workshop*, volume 3762 of *LNCS*, pages 676–687. Springer, 2005.

[12] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann, 2nd edition, 2008.

[13] S. Hartmann. On interactions of cardinality constraints, key, and functional dependencies. In *Foundations of Information*

and Knowledge Systems, volume 1762, pages 136–155. Springer Berlin, 2000.

[14] K. Houkjær, K. Torp, and R. Wind. Simple and realistic data generation. In *Proc. 32nd International Conference on Very Large Data Bases*, pages 1243–1246, 2006.

[15] M. Jarrar. Towards automated reasoning on ORM schemes. In *Proc. 26th International Conference on Conceptual Modeling*, pages 181–197. Springer-Verlag, 2007.

[16] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[17] S. A. Khalek, B. Elkarablieh, Y. O. Laleye, and S. Khurshid. Query-aware test generation using a relational constraint solver. In *Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 238–247, 2008.

[18] M. J. McGill, R. E. K. Stirewalt, and L. K. Dillon. Automated test input generation for software that consumes ORM models. In *On the Move to Meaningful Internet Systems: OTM Workshop*, volume 5872, pages 704–713. 2009.

[19] A. Neufeld, G. Moerkotte, and P. C. Lockemann. Generating consistent test data: restricting the search space by a generator formula. *The VLDB Journal*, 2(2):173–214, 1993.

[20] Y. Smaragdakis, C. Csallner, and R. Subramanian. Scalable satisfiability checking and test data generation from modeling diagrams. *Automated Software Engineering*, 16:73–99, 2009.

[21] A. Zamperoni and P. Löhr-Richter. Enhancing the quality of conceptual database specifications through validation. In *Proc. Entity-Relationship Approach*, volume 823, pages 85–98. Springer Berlin, 1994.

[22] J. Zhang, C. Xu, and S. C. Cheung. Automatic generation of database instances for white-box testing. In *Proc. 25th International Computer Software and Applications Conference Invigorating Software Development*, pages 161–165, 2001.