Automated Validation of Conceptual Schema Constraints

T.A. Halpin and J.I. McCormack

Key Centre for Software Technology
Department of Computer Science
University of Queensland. Australia 4072
email: halpin@cs.uq.oz.au

Abstract. For a database application, conceptual design methods such as fact-oriented modelling and entity-relationship modelling are commonly used to specify a conceptual schema, which may then be mapped to a structure in a chosen data model (e.g. a relational database schema). Since conceptual data models support a rich variety of constraints, and these constraints may impact on one another, the task of ensuring that the constraints expressed in a conceptual schema are consistent is non-trivial. Moreover, because different constraint patterns may be equivalent, some optimization may be needed to select the best constraint pattern for explicit assertion. With reference to conceptual schemas expressed in FOrML (an enhanced version of NIAM) this paper discusses meta-rules for strong satisfiability and constraint preference, and outlines an efficient algorithm for validating four main types of constraints. Complexity analyses and benchmarks of the implemented algorithm are included.

1 Introduction

The use of workbenches to provide automated support for the development of database applications is becoming widespread. For the modelling phase, it is becoming increasingly common for the data-perspective to be first specified in a human-oriented conceptual notation, which is then mapped to the appropriate logical data model (typically relational). Although most workbenches support a variant of EER (Enhanced Entity Relationship modelling), fact-oriented modelling arguably has several advantages (stronger linguistic basis, more constraint types, and its conceptual schema diagrams are more stable and easier to populate). Fact-oriented modelling (FOrM) comes in various flavours, under various names (e.g. NIAM, Binary-Relationship Modelling), and is supported by various CASE tools; some of these tools are well known (e.g. RIDL* from IntelliBase) while others are due for release this year (e.g. ITI's Conceptual Designer, and ServerWare's InfoViews).

Research at the University of Queensland is extending the fact-oriented modelling method, including automated support via a prototype known as WISE (Workbench for Information System Engineering). A detailed overview of WISE is given in Halpin (1991b). To place the topic of this paper (constraint validation) in perspective, a brief sketch of this project is now given. Conceptual schema editors are used to enter or modify conceptual schemas in graphical or textual form (with automatic layout).

Most syntax errors in the schema are detected at the entry stage since the editors incorporate most of the knowledge in the meta-conceptual schema. The output from this stage is fed to the Quality Checker: this performs further constraint validation, then checks for derivability and splittability of fact types (Zhang & Orlowska 1991).

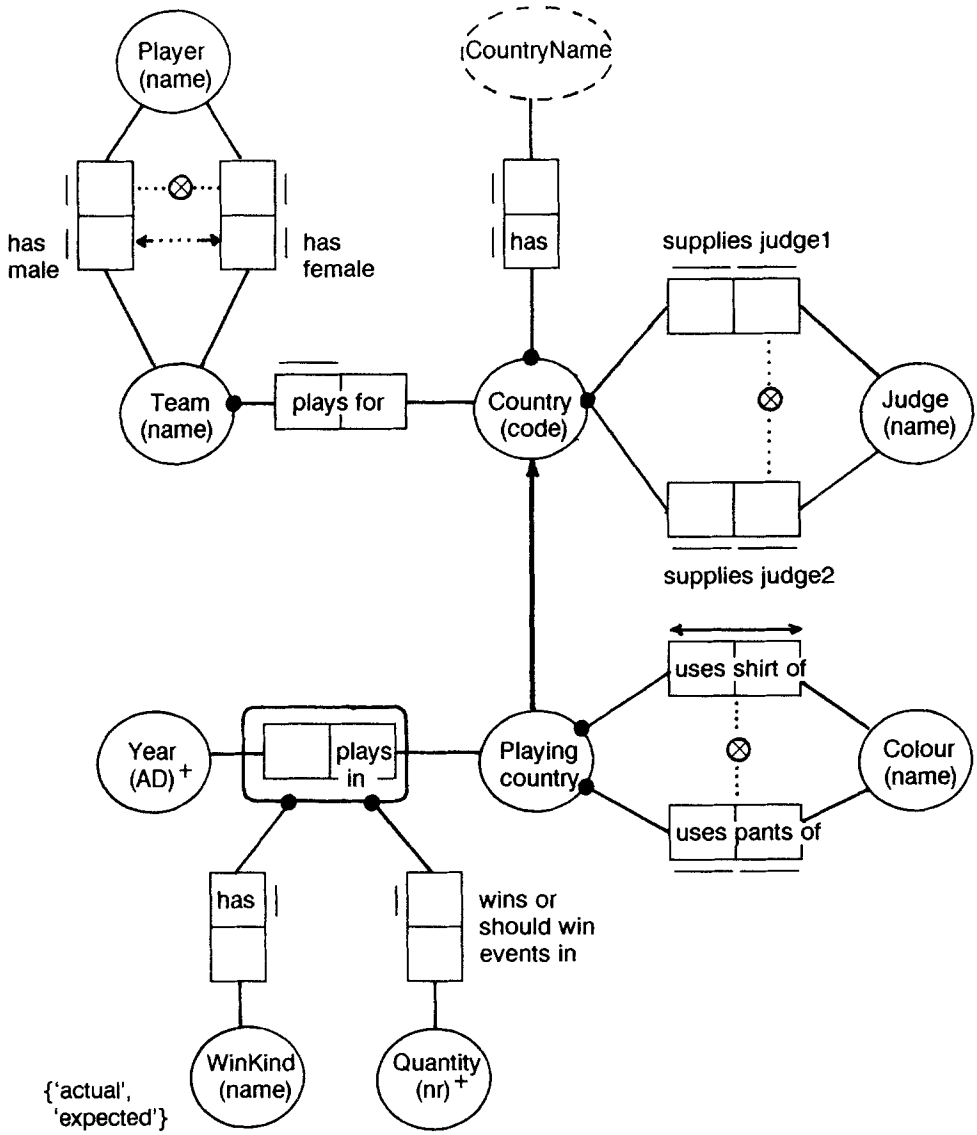
The checked schema is then passed to the conceptual schema optimizer, which transforms it to an optimal version using formal equivalence and implication theorems, and heuristic guidelines (Halpin 1989, 1991a, 1991c, 1992). The designer may interact with this module to over-ride defaults, provide better identifiers and allow information loss or gain. The optimized schema is then mapped to the appropriate data model (e.g. relational), generated, tested and tuned for a given DBMS. Other aspects under investigation include extending the schema languages and mapping algorithms, facilitating schema evolution and incorporating other object-oriented features. Only some of the phases just described have been implemented. This paper focusses on constraint validation.

Since conceptual data models support a rich variety of constraints, and these constraints may impact on one another, the task of ensuring that the constraints expressed in a conceptual schema are consistent is non-trivial. Moreover, because different constraint patterns may be equivalent, some optimization may be needed to select the best constraint pattern for explicit assertion. Section 2 defines the notion of strong satisfiability used for constraint consistency, and lists a number of results following from this definition. In section 3, various constraint implication theorems are cited, which indicate how some constraints may be implied by others, and guidelines are set out for explicit display of constraints. Section 4 specifies an algorithm for validating four kinds of constraint on lists of single roles. A similar algorithm is cited for lists of role-sequences. Complexity analyses and prototype benchmarks for these algorithms are included in section 5. The final section identifies some related problems for further research.

2 Strong Satisfiability of Conceptual Schemas

It is assumed that the reader has a basic grounding in logic and database theory. While much of our discussion can be translated into popular EER notions, we use fact-oriented modelling here since it facilitates work with constraints. For the reader who is unfamiliar with this method, we briefly discuss an example conceptual schema (see Figure 1). Entity types are denoted by named ellipses (e.g. Country). Value types (e.g. Number or CharString types) are shown as named broken ellipses (e.g. CountryName). Simple reference schemes for entity types are parentheisized (e.g. each Country is identified by its country code); a "+" on a reference mode indicates numeric reference (e.g. nr). Predicates are shown as named box-sequences (one box for each role); for example, the binary predicate plays_for has two roles. Predicates are ordered, with their name written in or beside their first role-box.

A bar across a sequence of one or more roles specifies a uniqueness constraint (instantiating object sequences may not be duplicated); arrow tips may be added to the bar (and must be if the roles are non-contiguous). For example, each team plays for only one country but a country may field many teams; the shirt relationship between Playing Country and Colour is many:many but the pants relationship is 1:1.



Playing_country $=_{df}$ Country **where exists** Team playing for Country

Fig. 1 An example of a fact-oriented conceptual schema diagram

Predicates which are completely spanned by a uniqueness constraint may be objectified; this nesting is shown as a frame (e.g. plays_in). A dot where *n* role-arcs connect to an object type indicates the disjunction of the *n* roles is *mandatory* or total (each object in the population of that type must play at least one of those roles). For example, each country must have a name, and supplies either one or two judges. An ⊗ symbol connecting role-sequences indicates mutual *exclusion* between the populations of these role-sequences. For example, a player cannot be both a male and female member of a team, and a country may not have a shirt colour which is the same as its pants colour. A dotted arrow from one role-sequence to another denotes a *subset* constraint (i.e. the population of the source is a subset of the target). A subset contraint in both directions is an equality constraint, and is shown as a dotted line with arrow-heads at both ends (e.g. a team has a male player if and only if it has a female player). A solid arrow from one object type to another indicates the former is a proper subtype of the latter; subtype definitions are specified at the bottom of the diagram (e.g. Playing_Country). Value-list constraints are shown in braces beside the relavent object type (e.g. WinKind). For a detailed background on fact-oriented modelling, see Nijssen & Halpin (1989); a recent overview is provided in Halpin & Orlowska (1992).

A conceptual schema diagram is mappable to a set of sentences in first order logic (Halpin 1989). An interpretation of a conceptual schema is then defined in the usual first-order way. An interpretation I of a conceptual schema CS is a *model* of CS iff each sentence of CS is true for I. A conceptual schema is *satisfiable* iff it has a model. In practice this notion of satisfiability is too weak, since it permits schemas with constraint patterns that are satisfiable only because these patterns are not populated. For example, a role with a uniqueness constraint and a frequency constraint of 2 generates a contradiction only if the role is populated: such constraint patterns are only *trivially satisfiable*.

The unsatisfactory nature of trivial models in relation to constraints has been noted in the literature. For example, Meyer, Weigand and Wieringa (1988, p. 13) attempt to avoid the problem by demanding that all models are non-empty. However, this is still too weak since it allows non-empty models with some empty predicates that are only trivially satisfiable. To demand that a CS must have a model in which all its predicates are non-empty is too strong, since legitimate exclusion and cardinality constraint patterns are rejected (Halpin 1989 pp. 6-3,4). We propose the following definition:

> A conceptual schema CS is *strongly satisfiable* (or *population-consistent*) if and only if: (a) for *each* of its predicates, there is a model of CS in which *that* predicate is instantiated; and (b) for *each* inter-predicate role-sequence which is an argument to an explicit constraint, there is a model in which *that* role-sequence is instantiated.

An inter-predicate role-sequence is an ordered list of roles, at least two of which occur in different predicates. Halpin (1989) proved various metarules using part (a) of this definition to avoid various cases of trivial satisfiability. We extend this work by considering some new cases which underpin the constraint validation algorithms discussed later.

The first rule is *NXS* (*N*o e*X*clusion with a *S*ubset constraint). In the diagram, rs1 and rs2 are each sequences of n roles ($n \geq 1$). The constraint arguments are rs1 and rs2 (not just subsequences). The exclusion constraint (denoted by ⊗) means there is no model in which both rs1 and rs2 are populated. On the left, a subset constraint from rs2 to rs1 is shown as a broken arrow from rs2 to rs1: in all models, each instance in the population of rs2 is also an instance in the population of rs1. Similarly, a subset constraint from rs1 to rs2 is shown on the right.

NXS Any schema with both an exclusion constraint and a subset constraint between the same two role-sequences is population-inconsistent (i.e. not strongly satisfiable).

Illegal combinations (constraints apply between whole role-sequences)

The proof of NXS is trivial. Consider the left-hand version. Assume strong satisfiability and both constraints hold. By strong satisfiability there is a model in which rs2 is populated. Let an instance in its population be a. The subset constraint implies that a occurs in the population of rs1. So a populates both rs1 and rs2, which contradicts the exclusion constraint. So the original assumption is wrong, i.e. the constraints are not strongly satisfiable. By swapping rs1 and rs2 the right-hand version follows. The constraint pattern is trivially satisfiable (there in a model where both rs1 and rs2 are empty, both constraints do hold), i.e. although the pattern is consistent it is population-inconsistent.

In the unlikely event that a designer explicitly enters both exclusion and subset constraints between the same role-sequences, this will be rejected by the editor. However, as discussed later, it is still necessary to check whether such a constraint combination is implied by other constraints on the schema.

As a related issue, the editor should be provided with knowledge as to where constraints may be meaningfully asserted on the schema. Apart from the obvious restrictions captured by graphic constraints on the meta-conceptual schema, further textual constraints at the meta-level must be specified and enforced. In this paper we restrict our attention essentially to *MSEX* constraints: *M*andatory roles, *S*ubset constraints, *E*quality constraints and e*X*clusion constraints. Recall that a role is mandatory (or total) for an object type if and only if each population instance of that type must play that role. Our approach bears some similarities to the "set constraint consistency analysis" performed by RIDL* (De Troyer et al. 1988, p. 398), but there are some significant differences.

To begin with, the "total union" and exclusion constraints commonly asserted between subtypes in RIDL* are unlikely to be ever used explicitly in our approach, since they are typically implied by the subtype definitions in conjunction with constraints on the fact types used in these definitions. For example, if Person has a mandatory functional association with Sexcode {'m','f'), and Man and Woman are defined as having Sexcode 'm', 'f' respectively then exhaustion (total union) and exclusion constraints for Man and Woman are implied.

Our treatment of subtypes is somewhat stricter than that of RIDL*. We demand that subtypes be definable in terms of roles played by their supertype(s), and give these definitions formal significance. If the designer ever tried to explicitly assert an exclusion or exhaustion constraint between subtypes this would be checked for consistency with the definitions and relevant constraints (see Halpin 1989 pp. 6-14,15 for relevant theroems), and typically allowed only as an implied constraint. In addition, subtypes are introduced only if they have a specific role to play. We feel this is a safer approach, as well as leading to less cluttered diagrams.

In some cases one might vacillate over whether to introduce a subtype or not (e.g. see Nijssen & Halpin 1989, p. 178). We resolve such cases by the following recursive *subtype introduction procedure* (SIP):

- If an optional role is played only by a well-defined subtype, then specify the subtype definition.
- If the subtype definition is stronger than "[not] playing a role directly attached to [one of its] supertype[s]" then introduce the subtype (and apply SIP to it).
- If the subtype definition can be expressed instead as a subset or exclusion constraint then do so, unless there are several roles which bear equality or subset constriants to the candidate subtype role (in which case introduce the subtype and apply SIP to it).

It is clear that subset, equality and exclusion constraints are allowed between role-sequences only if these are compatible (same corresponding host object types). Also exclusion constraints between exclusive subtypes, as well as subset constraints from subtype roles to mandatory supertype roles, are implied and hence omitted. Mainly as a consequence of the SIP procedure, other meta-rules follow which further restrict the explicit depiction of such constraints. In particular:

Consider two different object types B and C, with the same host supertype, and let *r* and *s* be roles attached to B and C.

- An explicit subset constraint from *r* to *s* is allowed only if *r* and *s* are optional and B is (directly or indirectly) a subtype of C.
- An explicit exclusion constraint between *r* and *s* is allowed only if *r* and *s* are optional and either B is a subtype of C or C is a subtype of B.

The significance of such rules is twofold: they allow such rule violations to be rejected at the schema entry stage; knowing these rules are now obeyed simplifies the working of the constraint validation checking applied later.

3 Constraint Implication and Display

Let *CS* be any well formed conceptual schema, and *C* be any well formed static constraint to be added to *CS*. Then *CS implies C* iff *C* is true in all models of *CS*. Using formal logic, Halpin (1989) proved several constraint implication theorems for fact-oriented schemas. We cite without proof the following results:

- If roles *r* and *s* are mandatory and optional for the same object type, then a subset constraint is implied from *s* to *r*.

Let *rs1* and *rs2* be compatible role-sequences (of 1 or more roles). Then:

- A subset constraint from *rs2* to *rs1* implies a subset constraint from each subsequence in *rs2* to the corresponding subsequence of *rs1*.
- An exclusion constraint between *rs1* and *rs2* implies an exclusion constraint between all compatible supersequences of *rs1* and *rs2*.

- Let *rs1*, *rs2* and *rs3* be compatible role-sequences. A subset constraint from *rs1* to *rs2* combined with a subset constraint from *rs2* to *rs3* (transitively) implies a subset constraint from *rs1* to *rs3*.

- An exclusion constraint between *n* role-sequences is equivalent to the exhaustive conjunction of *n(n-1)/2* binary exclusion constraints between all possible pairs of these role-sequences.

- An equality constraint between two role-sequences is equivalent to subset constraints in both directions.

- If a single equality constraint between *n* role-sequences is allowed, it is equivalent to equivalent to the *n-1* binary equality constraints between adjacent role-sequences.

- If the disjunction of roles *r* and *s* is mandatory, and a subset constraint exists from *s* to *r*, then *r* is mandatory. In this case *r* should be displayed separately as mandatory.

Many other examples of constraint implication and display preferences are given in Halpin (1989), dealing with uniqueness, frequency, cardinality, subtype, asymmetry etc. constraints. However those results cited here provide sufficient background for the constraint validation algorithms which follow. We restrict ourselves to the following problem:

> Given a conceptual schema output from the CS editor, is the pattern of its MSEX constraints strongly satisfiable? If not, specify the violation(s) (and ideally interact with the designer to correct the schema). If it is strongly satisfiable, then optimize the display by hiding implied constraints except that mandatory roles are always to be depicted as mandatory.

For example, consider the constraint pattern of Figure 2. Each box denotes a single role. The exclusion constraint applies between the left-hand roles, but the subset

constraint (by connecting the role-junction points) applies between the role-pairs. The pair-subset constraint implies subset constraints between the left-hand roles (and between the right-hand roles). From theorem NXS, this conflicts with the exclusion constraint and the schema is not strongly satisfiable. Note that because subset contraints are implied through all subsequences, we merely had to look for an implied subset constraint on the explicit exclusion constraint: we did not have to derive the supersequence exclusion constraints.

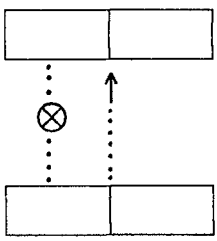


Fig. 2 This constraint pattern is not strongly satisfiable

As a simple example of optimizing constraint display, consider the left-hand diagram in Figure 3. Object types are depicted by ellipses. A dot connected to one or more roles means the disjunction of these roles is mandatory for the dotted object type. The pair-subset constraint implies subset constraints from the lower roles to the roles directly above them, which combined with the disjunctive mandatory role constraints imply that the top roles are mandatory and hence must be displayed as shown on the right.

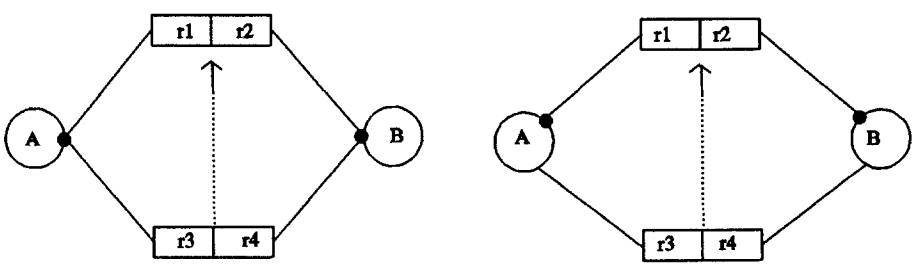


Fig. 3 The left-hand version should be converted to the right-hand version

The basic ideas underlying *MSEX constraint validation (checking for constraint satisfiability, and optimizing the constraint display)* have now been covered, so that an intelligent designer would normally be able to carry out this procedure manually. However, with large schemas this is both tedious and open to error (because of the large numbers of implied constraints). Hence it is desirable to have this validation performed automatically. The rest of the paper specifies algorithms for doing this efficiently.

4 MSEX Constraint Validation

We believe the algorithms presented here for validating MSEX constraints offer advantages such as efficiency and completeness compared with those used by other systems. For portability the algorithms have been coded in C. Basically, we perform a "populability check" of conceptual schemas, which focusses on potential schema populations, indicating any predicates which cannot be populated as well as any redundant subset constraints.

The basic principle of the algorithm is to build a series of graphs such that each graph represents a "population scenario" (possible world). A directed arc between two vertices *u, v* in a graph is equivalent to the subset constraint: every instance of an object-type which plays role *u* in this scenario also plays role *v*.

The population graphs are a series of directed graphs which represent possible populations of the schema, and are defined as either unary (compare single roles) or n-ary (compare sequences of at least 2 roles). Although, the unary case is just a special case of the n-ary, the algorithms are treated differently as they typically have to be implemented separately.

For simplicity, the main algorithms assume that exclusion and subset constraints span only two role-sequences. All constraints not of this form are pre-processed to give a series of binary constraints. All equality constraints are then pre-processed to form two subset constraints. The details of the pre-processing are as follows.

Pre-processing:

If n-ary equality constraints are allowed, each equality constraint of the form =(rs1,...,rsn) is transformed into (n-1) binary equality constraints between adjacent pairs: rs1 = rs2, .. , rs_{n-1} = rsn. For example, =(rs1,rs2,rs3) becomes: rs1 = rs2, rs2 = rs3.

Each n-ary exclusion constraint of the form ⊗(rs1,rs2,...,rsn) is transformed into n(n-1)/2 binary constraints by exhaustive pairing: rs1 ⊗ rs2,..., rs1 ⊗ rsn, rs2 ⊗ rs3,... , rs2 ⊗ rsn, ... , rs_{n-1} x rsn. For example, ⊗(rs1,rs2,rs3) becomes: rs1 ⊗ rs2, rs1 ⊗ rs3, rs2 ⊗ rs3.

All equality constraints of the form rs1 = rs2 are transformed into two subset constraints of the form rs1 → rs2, rs2 → rs1 (using "→" to denote "is a subset of"). For example, rs1 = rs2 becomes: rs1 → rs2, rs2 → rs1.

Finally, all subset constraints between sequences of *n* roles (*n* > 1) have the implied subset constraints generated for each of their subsequences. Thus a single such constraint is expanded to 2^n-1 subset constraints. For example, (rs11, rs12,rs13) → (rs21,rs22,rs23) is expanded to the 7 constraints:

unary:	rs11 → rs21; rs12 → rs22; rs13 → rs23
binary:	(rs11,rs12) → (rs21,rs22); (rs11,rs13) → (rs21,rs23); (rs12,rs13) → (rs22,rs23)
ternary:	(rs11,rs12,rs13) → (rs21,rs22,rs23)

This final expansion is used to optimize constraint display. If only a consistency check is needed, subset constraint generation can be limited by existing exclusion constraints.

Unary validation *(constraints which compare single roles):*

We construct a directed graph G = (V,E), where V is a set of vertices and E is a set of edges. All v ∈ V are roles whose connecting object-types have the same root in their subtype-graph, i.e. a population graph around some object-type O (with no super-types) has as its vertices all of the roles played by O (directly connected to O) or any subtypes of O. All e ∈ E are edges of the form (u ∈ V -> v ∈ V) where it can be shown that any instance of O playing role u must play role v.

The algorithm must be run for each subtype graph (possibly trivial) in the schema. A sub-type graph is identified by its root. From this point onwards, a role refers not only to the role within the schema but also to that role's corresponding vertex in the graph. An overview of the algorithm is now given.

Step 1:

The initial step in the algorithm is to add the vertices to the graph. Each vertex in the graph corresponds to a role which is directly connected to an object-type in the subtype graph of interest. All such roles are represented in the graph.

All edges in the graph are directed and have an extra attribute which records the constraint from which the edge was derived. This is useful for reporting redundant subset and exclusion constraints.

Step 2:

The next step adds to the graph the edges which are derived from non-disjunctive mandatory role constraints. Put simply this is: "For all such constraints which span a role *r* in the graph where *r* is connected to some object-type O, add directed edges to *r* from all other roles connected to either O or subtypes of O. Mark these edges as being derived from the relevant mandatory role constraint." At this point, the base-graph has been formed. Before any checking can be done, edges produced by subset and disjunctive mandatory role constraints are added to the graph.

Next a sequence of possible non-empty populations is built up from the disjunctive mandatory role constraints which span roles in the graph. Basically, we form the power-set and subtract the null set.

For example, suppose there are two disjunctive mandatory role constraints c1 and c2 which impact on the graph, where c1 spans roles r1, r2 and c2 spans roles r3, r4. The possible (non-empty) populations of roles 1 and 2 are p1 = {{1},{2},{1,2}}, i.e. either role 1 may be populated and not role 2, role 2 may be populated and not role 1, or both may be populated (there are 2^n-1 of these "population scenarios").

Similarly, the possible populations of roles 3 and 4 are p2 = {{3},{4}, {3,4}}. The total possible set of schema populations around the roles influenced by c1 and c2 is M = { {1,3}, {1,4}, {1,3,4}, {2,3}, {2,4}, {2,3,4}, {1,2,3}, {1,2,4}, {1,2,3,4} }. The size of this set is #p1 * #p2, the product of the cardinalities of p1 and p2.

Step 3.

Using the base-graph as a starting point, apply the following algorithm to each m ∈ M.

Step 3.1 (add disjunctive-mandatory role constraints):

For all roles $r \in$ m, add edges to the graph using the same procedure as that used for non-disjunctive mandatory roles, i.e. if r is connected to some object-type O, add directed edges from all other roles connected to either O or subtypes of O to r. Mark these edges as being derived from the relevant mandatory role constraint.

Step 3.2 (add the subset constraints):

For each subset constraint $r \rightarrow s$ such that $r \in$ V and $s \in$ V, if no path from r to s exists in the graph, add an edge from r to s to the graph and mark it as being derived from the relevant subset constraint.

Step 3.3 (check the exclusion constraints):

For each exclusion constraint of the form $r \otimes s$ such that $r \in$ V and $s \in$ V if a path exists from r to s in the graph, the constraint is invalid since role r cannot be populated; if a path exists from s to r in the graph, the constraint is invalid since role s cannot be populated.

Step 3.4 (check the subset constraints):

For each subset constraint $r \rightarrow s$ such that $r \in$ V and $s \in$ V, if the path from r to s in the graph is not a single edge which has been derived from the subset constraint currently being checked, the constraint is redundant in this population scenario. If a subset constraint is redundant in all population scenarios (for all m in M) the constraint is redundant in the schema.

Step 3.5 (check for implied mandatory role constraints)

For each role r in the graph where it is possible to get from all other roles in the graph to r, r is mandatory for this population scenario. If a role is mandatory in all population scenarios, the role is mandatory in the schema. Note that the information needed for this check can be built up while adding the subset and mandatory role constraints.

As an example of the unary check, consider the incomplete schema fragment depicted in Figure 4. In this graph there are 13 roles labelled "r1".."r13" connected directly (or indirectly via subtypes) to the object type named "Target OT".

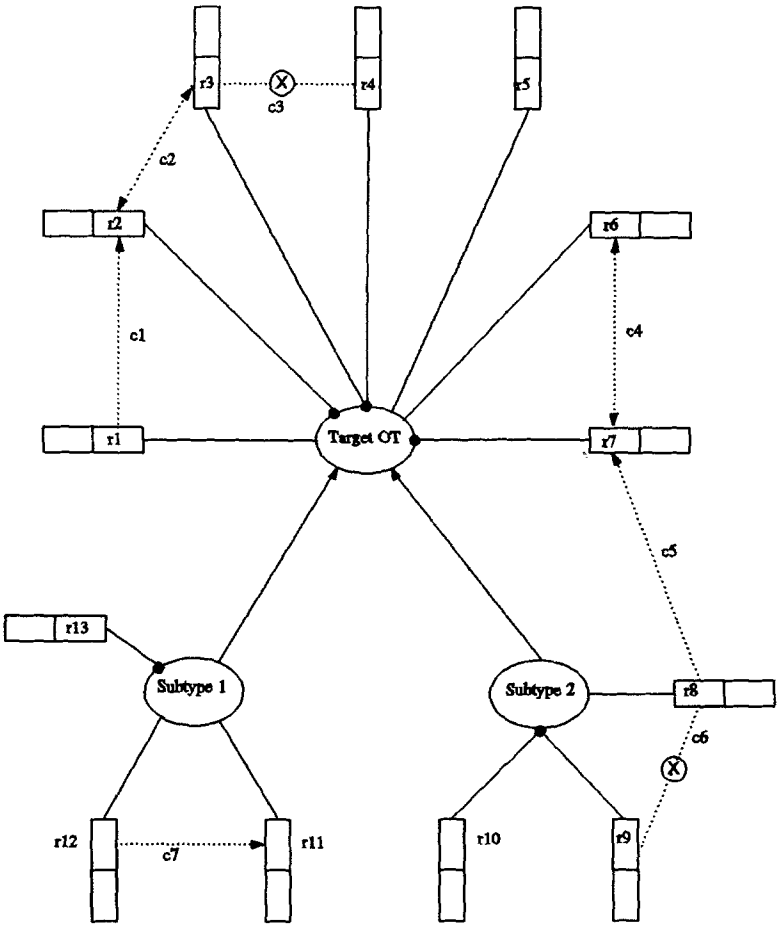


Fig. 4 A schema fragment used to illustrate unary validation

For this example, the algorithm is executed thus. The vertices V = {r1 .. r13}. Add directed edges added for non-disjunctive mandatory roles: r1 → r2, r3 → r2, r4 → r2, r5 → r2, r6 → r2, r7 → r2, r8 → r2, r9 → r2, r10 → r2, r11 → r2, r12 → r2, r13 → r2, r1 → r7, r2 → r7, r3 → r7, r4 → r7, r5 → r7, r6 → r7, r8 → r7, r9 → r7, r10 → r7, r11 → r7, r12 → r7, r13 → r7, r12 → r13, r11 → r13. Hence in the base graph G, all vertices have edges to r2 and r7, and there are edges from r12 to r13 and r11 to r13.

M = { {r3, r9}, {r3, r10}, {r3, r9, r10 }, {r4, r9},
{r4, r10}, {r4, r9, r10 }, {r3, r4, r9}, {r3, r4, r10},
{r3, r4, r9, r10 } }

Now consider iteration 1 of step 3. Add disjunctive mandatory roles. All vertices now have edges to r2, r3, r7, and r9. Adding subset constraints causes the following edges to be added: r2 → r3, r3 → r2, r6 → r7, r7 → r6, r1 → r2, r8 → r7, r12 → r11, r2 → r3, r3 → r2, r6 → r7, r7 → r6, r12 → r13, r11 → r13. The results of iteration 1 are: c1 marked as redundant; c2 can be replaced by a subset constraint to R2; c4 can be replaced by a subset constraint; c5 is redundant

The results of the complete execution of the algorithm are: c1 tagged as redundant; c2 can be replaced by a subset constraint to R2; c3 tagged as invalid since R4 cannot be populated; c4 can be replaced by a subset constraint; c5 is redundant; c6 is valid; c7 is valid.

Note that the algorithm detects all errors but makes a default assumption regarding the best way to remove the error. An improvement would be to interact with the designer to determine which error correction option is best (for example, another possible error correction with the current example is that r3 should have been mandatory, leaving c2 as implied.

N-ary validation *(constraints comparing sequences of 2 or more roles):*

An n-ary check checks all n-ary subset, equality and exclusion constraints in the schema (see Figure 4). Again G = (V,E) is a directed graph. A "population node" is derived from a role-sequence of two or more roles in a constraint. For example, in Figure 5 the pair-subset constraint from the role-pair (r1,r2) to the role-pair (r3,r4) has the population nodes r1#r2 and r3#r4.

A population node *n* is "population equivalent" to role-sequence *rs* iff all roles occurring in *n* occur in the same order in *rs*. A population node *n* is the "corresponding" population node to population node *m* within constraint *c* iff *c* is made up of *rs1* and *rs2*, and *n* is population equivalent to *rs1* and *m* is population equivalent to *rs2*, or vice-versa.

Given a binary constraint made up of rs1 = (r1,r2) and rs2 = (r4,r5) the population node n = r1#r2 is population equivalent to rs1, and the corresponding population node in rs2 of r1#r2 in the constraint is r4#r5.

Having described the n-ary form of the population graph, we now give an overview of the n-ary validation algorithm. This is similar in principle to the unary algorithm. Vertices in the graph used in the n-ary case represent sequences of n roles. For a given N, the algorithm must be run on every n-ary constraint in the schema.

For each n-ary constraint cc of the form p1 subset p2 or p1 exclude p2, where p1 and p2 are the n-ary population nodes which represent the n-length role-sequences in cc, add the vertices p1 and p2 to the graph (V = { p1,p2}) and proceed as follows:

```
Changed = true;
C = {}; ( C is the set of constraints which have been processed )

while changed do
      for all subset constraints c where c spans some v ∈ V and c is not in C
            c is of the form pa subset pb
            C = C + c
            if pa is not in V
                  V = V + pa;
                  Changed = true
            if pb is not in V
                  V = V + pb;
                  Changed = true
            If no path form pa to pb exists in the graph
                  Add an edge from pa to pb
                  Mark the edge as being derived from c
```

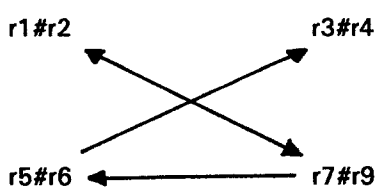
if cc is a subset constraint and the path from p1 to p2 in the graph is not a single edge which is derived from cc, cc is redundant.

if cc is an exclusion constraint and there is a path from p1 to p2 in the graph, cc is invalid since p1 cannot be populated.

if cc is an exclusion constraint and there is a path from p2 to p1 in the graph, cc is invalid since p2 cannot be populated

As an example, consider the schema fragment shown in Figure 5. Intuitively, there is a problem since a pair-subset constraint is implied from (r1,r2) to (r3,r4), which is population-inconsistent with the exclusion constraint c1.

The algorithm checks the exclusion constraint c1 as follows. Initially V = {r1#r2,r3#r4} and E = {}. Adding c2 adds the vertex r5#r6 and the edge r5#r6 → r3#r4. Adding c3 adds the vertex r7#r9 and the edge r7#r9 → r5#r6. Adding c4 adds the edges r1#r2 → r7#r9 and r7#r9 → r1#r2. The final graph for the check of the exclusion constraint is as shown. Since there is a path from r1#r2 to r3#r4, the exclusion constraint is invalid because r1#r2 cannot be populated (by theorem NXS).



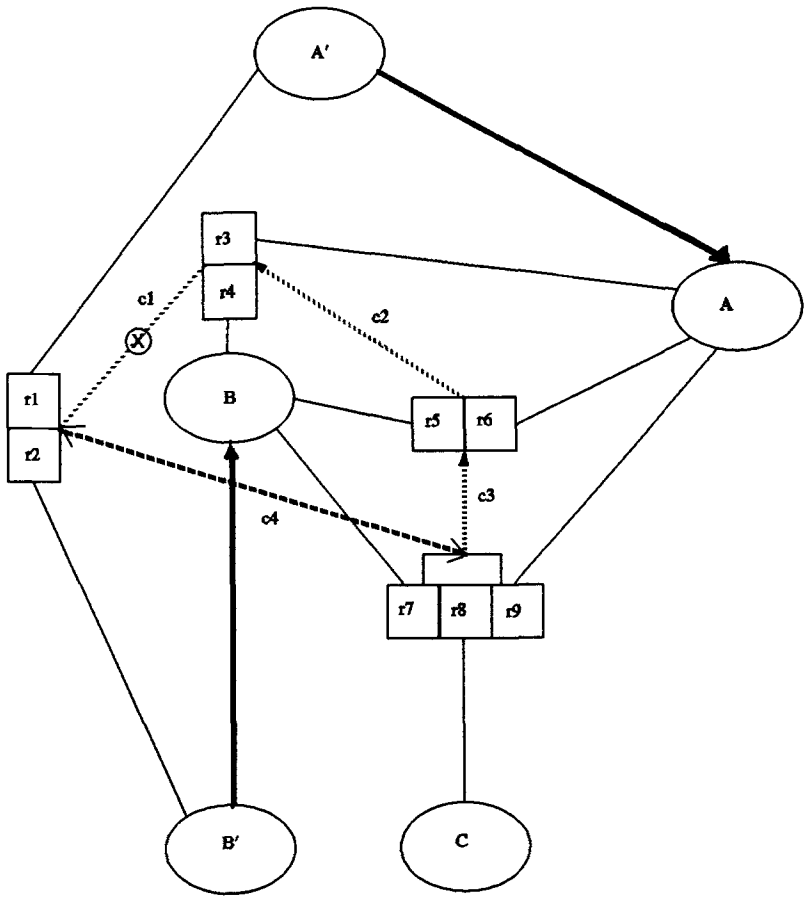


Fig. 5 A schema fragment used to illustrate n-ary validation

As a final example consider the first schema shown in Figure 3. The set of population scenarios for the (trivial) subtype graph with root A is M = { {r1}, {r3}, {r1, r3} }. Each of the 3 resultant graphs has an arc from r3 to r1 added due to the subset constraint.

Since r1 is mandatory in all 3 population scenarios, r1 is mandatory in the schema and the disjunctive mandatory role constraint over roles r1 and r3 can be replaced by a single mandatory role conatraint over r1.

Similarly the disjunctive mandatory role constraint over r2 and r4 can be replaced by a single mandatory role constraint over r2.

The n-ary case has two nodes - r1#r2 and r3#r4 and one edge from r3#r4 to r1#r2. There are no redundancies or errors.

When the schema is re-assembled with the new mandatory role constraints the schema takes the more correct form of the second schema shown in Figure 3.

5 Complexity Analysis

The complexity analysis ignores the pre-processing step. Complexity of a *unary* check on a single subtype graph is determined as follows. Let R be the number of roles in the subtype graph. Let C be the number of subset, equality, exclusion and non-disjunctive mandatory role constraints constraints which impact on the subtype graph. Let NM be the number of non-disjunctive mandatory role constraints which impact on the subtype graph. Let P be the number of elements of the set M (the set of possible schema populations). P is exponential in the arity of the largest disjunctive mandatory role constraint which impacts on the subtype graph.

Step 1 is clearly O(R). Step 2 has a worst case complexity of O(R * NM). Step 3 is performed P times. Step 3.1 takes O(R * the number of elements in m). Step 3.2 takes O(R * R * the number of subset constraints involved in the graph) if an algorithm such as Dijkstra's is used to check for the path. Step 3.3 takes O (R * R * the number of subset constraints involved in the graph) if an algorithm such as Dijkstra's is used to check for paths. Step 3.4 also takes O(R * R * the number of subset constraints involved in the graph). Since the number of elements in m is never greater than the number of vertices in the graph, Steps 3.1 - 3.4 are O(#constraints * R * R). Hence the complexity of checking one subtype graph is O(R * R * P * C).

Let OTS = the number of object-types in the schema. Let RS = the number of roles in the schema. Let CS = the number of subset, equaliity, exclusion and non-disjunctive mandatory role constraints in the schema. Let DMS = the number of disjunctive mandatory role constraints in the schema.

Let SLDM = the size (arity) of the largest disjunctive mandatory role constraint in the schema. Let PS = the number of elements involved in the largest "possible world set " (M). This is O(DMS * 2^{SLDM}).

Each object-type may only occur in one subtype graph. Each role may only occur in one subtype-graph. Hence the complexity of the entire schema is O(RS * RS * CS * PS).

We now analyse the complexity of the n-ary case. Let C = the number of n-ary subset and exclusion constraints involved in the current graph. The complexity of checking a single n-ary constraint is is $O(C^3)$ since testing for an existing path between nodes is is $O(C^2)$. Hence, for a complete schema the complexity is: $O(CS^4)$ where CS = the number of n-ary subset, and exclusion constraints in the schema.

Although the complexity analysis for the unary algorithm yeilds a result which is exponential in the size of the disjunctive mandatory roles within the schema, the number of large disjunctive mandatory roles constraints is typically small. As a result of this the performance of the algorithm is not adverseley affected by the exponential complexity.

Furthermore, if the schema is only to be checked for strong-satisfiability, only those subtype graphs containing exclusion constraints need be checked for the n-ary case and only exclusion constraints need be checked for the nary case.

The preceding algorithms have been implemented, and produced the following results when run on a schema containing in excess of 250 object-types, 300 predicates, 400 mandatory role constraints, 6 non-trivial subtype graphs, 30 subtype & equality constraints and 10 exclusion constraints. On a SUN SparcStation 2 the full check (unary and n-ary) completed in 3 seconds. On an IBM compatable 16 MHz 80386-SX the same check completed in 68 seconds.

6 Conclusion

This paper has examined the notions of constraint satisfiability and implication for four important classes of constraints in fact-oriented modelling, and specified efficient algorithms for their checking and display optimization. Since these constraints are at least partially supported by various versions of EER modelling, the work has wider implications. While validation procedures have been developed for other classes of constraints (e.g. uniqueness), there are several other constraints in fact-oriented modelling which require a similar set of validation algorithms (e.g. frequency, asymmetry). The development and implementation of efficient validation algorithms for such constraints is a topic for future research.

References

Bry, F. & Manthey, R. 1986, 'Checking Consistency of Database Constraints: a Logical Basic', *Proc. Twelfth Int. Conf. on Very Large Data Bases*, VLDB, Kyoto, pp. 13-20.

De Troyer, O., Meersman, R. & Verlinden, P. 1988, 'RIDL* on the CRIS Case: a Workbench for NIAM', *Computerized Assistance during the Information Systems Life Cycle: Proc. CRIS88*, eds T.W.Olle, A.A. Verrijn-Stuart & L. Bhabuta, North-Holland, Amsterdam.

De Troyer, O. 1989, 'RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints', *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Oregon.

De Troyer, O. 1991, 'The OO-Binary Relationship Model: a truly object-oriented conceptual model', *Advanced Information Systems Engineering: Proc. CAiSE-91*, Springer-Verlag Lecture Notes in Computer Science, no. 498, Trondheim.

Halpin, T.A. 1989, 'A Logical Analysis of Information Systems: static aspects of the data-oriented perspective', PhD thesis, University of Queensland.

Halpin, T.A. 1991a, 'Optimizing Global Conceptual Schemas', *Databases in the 1990s: 2*, eds B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.

Halpin, T.A. 1991b, 'WISE: a Workbench for Information Systems Engineering', *Proc. 2nd Workshop on Next Generation of CASE Tools*, Trondheim.

Halpin, T.A. 1991c, 'A Fact-Oriented Approach to Schema Transformation', *Proc. MFDBS-91*, Springer-Verlag Lec. Notes in Computer Science, no. 495, Rostock.

Halpin, T.A. 1992, 'Fact-oriented schema optimization', to appear in *Proc. CISMOD-92*, India, July 1992.

Halpin, T.A. & Orlowska, M. E. 1992, 'Fact-Oriented Modelling for Data Analysis', *Journal of Information Systems*, vol. 2, no. 2, Blackwell Scientific, Oxford.

Halpin, T.A. & Ritson, P.R. 1992, 'Fact-Oriented Modelling and Null Values', *Research and Practical Issues in Databases: Proc. 3rd Australian Database Conf.*, eds B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.

Lundberg, B. 1983, 'On Correctness of Information Models', *Information Systems*, vol. 8, no. 2, pp. 87-93, Pergamon Press.

Meyer, J., Weigand, H. & Wieringa, R. 1988, 'Specifying Dynamic and Deontic Integrity Constraints', Rapport IR-175, Vrije Universiteit, Amsterdam.

Nijssen, G.M. & Halpin, T.A. 1989, *Conceptual Schema and Relational Database Design: a fact-oriented approach*, Prentice Hall, Sydney.

Qian, X. & Wiederhold, G. 1986, 'Knowledge-based Integrity Constraint Validation', *Proc. Twelfth Int. Conf. on Very Large Data Bases*, Kyoto, pp. 3-12.

Rajagopalan, P. & Ling, T.W. 1987, 'A method for semantic validation of a class of integrity constraints', *Tech. Report*, Uni. of Singapore.

Zhang, Y. & Orlowska, M.E. 1991, 'Synthesizer+: an automatic tool for relational database design', *Proc. 14th Australian Computer Science Conf.*, Sydney.