ON THE PROPERTIES OF EXTENDED INCLUSION DEPENDENCIES

Hiroshi ARISAWA
Yokohama National University
Tokiwadai, Hodogaya-ku
Yokohama, Japan

Takao MIURA
Mitui Engineering & Shipbuilding
Tsukiji, Chuo-ku,
Tokyo, Japan

abstract

In this paper, we propose new classes of Inclusion Dependencies as an extension of "Generalization" based on the Entity-Association model. Various kinds of extensions are discussed, and four classes (IND, IXG, UXG and co-EXD) are evaluated from the viewpoint of database design. We present the complete inference axioms for each class and the polynomial complexity of inference problems.

1. BACKGROUND

One of the most important issues in database design is certainly the specification of integrity constraints. Since databases must be kept consistent, and some external checking mechanisms must be provided. The constraints detected at database design should be embedded in the database schemes by some way such as "normal" forms so that the consistency is preserved. Those constraints could be also used for query optimization.

A typical example is found in the relational database theory. That theory [23] has been considerably developed, with the help of functional dependency or join dependency which have been much discussed by many investigators. However, the theory is heavily based on the mathematical properties of the dependencies and it is doubtful for database designers to construct their databases in accordance with the theory, since those properties are neither intuitive nor easily understandable.

Originally information model is considered to allow us to capture "real world", and the design methodology based on the model must be intuitive enough to reflect the world, powerful to describe this world and easy to construct on it information structure. In this sense, database designers must start their work with the initial image and they are encouraged to capture the information structure. Naturally, designing database means structuring information. More precisely, designing databases consists of two phases [4]; information modelling to construct information structure and data modelling to change the information structure into the ones adaptable for computer processing. Entity Relationship (E-R) model [10], Entity Association (E-A) model [18] and Navathe-Schkolnick model [17] are well-known information models. As data models, relational model, network model or hierarchical model are the examples. We believe that both modellings should be closely related so that no concept translation is required.

Despite of these proposals, the enumeration of design problems still comes to us [14]. For example, in the first stage of database design, both of predicates (or schemes) and constraints among attributes should be simultaniously detected, but the design methodology to reflect them is not fully developed [24].

In this paper, we will pay attention on the structures among sets (or "types") of entities, because they are sometimes regarded as design primitives, or sometimes as constraints. Some investigators also discussed this problem; Generalization [22] concerns inclusion relationship among entity sets. [10] proposes the notion of existency dependency which considers generalizations as constraints. And [7] discusses this topic from the viewpoint of relational model. Here we regard Generalization as constraints over sets of entities, and subsequent sections discuss the extension and the properties of the constraints.

Section 2 gives the definition of databases and the data model AIS. In section 3, we introduce Inclusion Dependency (IND) and develop the straightforward extension with the polynomial membership algorithms. Section 4 states the other kind of extension, Exclusion Dependency (EXD), and the interaction with INDs. However, the EXD class will be shown to be rather inappropriate since many "unrelated" entity types exist in the database. Alternatively we will propose, in section 5, the notion of co-Exclusion Dependency (co-EXD) and its interaction with INDs. Also the polynomial time membership algorithms will be stated.

In any model, rigorous treatment should be provided, for we believe that the theory has the expressive power only when unambiguous concepts and effective operations are provided. This brings us to discuss database theory on mathematical framework. Especially axiomatization combines the intuitive correctness and the conceptual derivability. Sound and complete axioms for interesting universe are the ones we want. Also, testing membership algorithm and its (time) complexity is another importance since our aim is to decide wether a particular member can be derived or not.

Throughout the paper, we assume the basic concepts in mathematical logic [16]. For

Proceedings of the Twelfth International Conference on Very Large Data Bases **Kyoto, August, 1986**

example, propositional logic and the first-order predicate logic are referred without explanation.

2 AIS data model and the database

In this section, we present the definition of Associative information structure (AIS) and the databses on which we will state our theory. The model stands on five prmitives: entity, entity type, association, predicate and constraint.

Informally, an entity is a logical object in the database which corresponds to a distinguishable thing in the real world. For example, a manager or a secretary is an entity.

Collections of entities can be often grouped together to perform a semantic unit, in turn, this must be corresponded to an entity called entity type. Each entitiy type is denoted by a spelling such as Manager or Secretary.

On the other hand, the set of entitties itself is called an entity-set or E-set. Conceptually entity type represents "intension" and E-sets "extension". For a type, the E-set means "active" domain [11] because of extension.

E-sets may be mutually overlapped, that is, an entity can be included in two or more entity sets. For instance, an office-worker may be both a manager and a secretary.

A predicate expresses n-ary relationship among entity sets. In other words, the predicate corresponds to a "relationship type". This is explicitly specified by the designers. The occurrence of the predicate describes a particular information among entities which are in the corresponding sets. Each occurrence is being called an association.

To keep all the information consistent, the designers must specify constraints over types explicitly. For example, "active" domain property is a constraint such that every entity in associations of a predicate must be in the appropriate entity sets and that no other entity exists in the entity sets. As another example, in this paper, we discuss Inclusion Dependencies which says "an entity set A is always a subset of another entity set B" (i.e. if an entity e is an element of A, then e must be an element of B).

As stated before, AIS data model represents the information model and the data model; the actual collection of entities, entity types, predicates and associations constitute the AIS database. The materialization of the database scheme is provided by an AIS diagram. An Entity-set is represented by ovals, a predicate by ◆ linked to entity sets, an entity by o and an association by •. In the following, lower case letters a,b,c mean entities; upper case letters A,B,C mean entity types; $\lambda(A),\lambda(B),\lambda(C)$ mean entity sets of type A,B,C respectively. $[E_1 \ldots E_n]$ represents the predicate defined on entity types $E_1,\ldots,E_n$. As shown in this paper, INDs and the extensions can be captured in the diagram by set inclusion notation. For more detail, see [1],[2] and [4].

[Example] Throughout this paper we refer the same example, modified version of [6]. We assume there are three predicates and eight entity types like:

[O_Worker Floor] says Office Worker w is located on Floor f.

[Manager Secretary Day] says Manager m works on Day d with Secretary s.

[Director Limousine Driver] says Director r uses Limousine l drived by Driver e.

All these are drawn by Fig.1,(a)-(c).

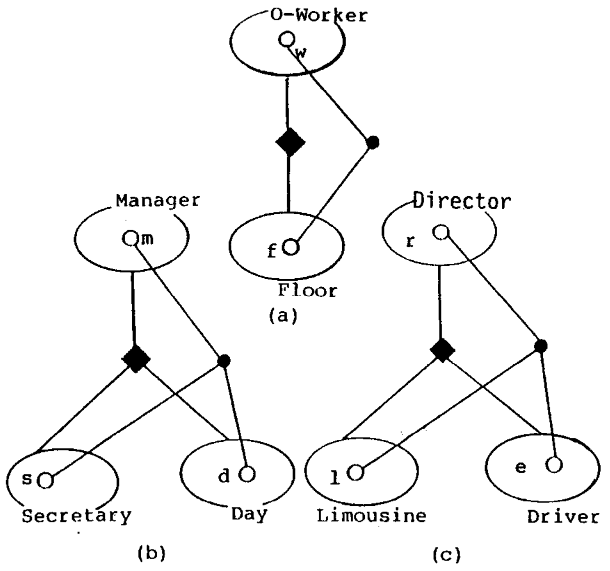

Fig. 1 AIS diagram

3 EXTENDED GENERALIZATION CONSTRAINTS

3.1 Inclusion Dependency

To each object in the real world one and only one entity is associated in the database so that several entity sets may share entities. Moreover, very often there are several inclusion relationship among entity sets, and they must be considered as constraints.

Given two entity types A and B, A is called inclusive to B, denoted $A<B$, if $\lambda(A)$ is always a subset of $\lambda(B)$. Such kind of constraints is said Inclusion Dependency (IND).

On the other hand, Generalization [22] is a constraint such that every entity in $\lambda(T_1)$ has the same relationship involved by T_2 if T_2 is a generalization of T_1. T_1 is sometimes said "subtype" of T_2. This is naturally embedded in AIS databses; if an entity in $\lambda(T_1)$ is in $\lambda(T_2)$, then the entity has to be related to an association on T_2, for, all entity sets are active in AIS.

Semantically IND is based on this concept. In fact, as we show later, extended INDs can describe more sophisticated classes of Generalization. [4] discusses the design methodology using extended INDs. [7] discusses INDs in the framework of relational model. [25] presents the similar discussion from the viewpoint of knowledge representation. Also, [21] shows that "classical" relational model plus allowable INDs is equivalent to a subclass of Universal Instance model. Note that the inference problem of INDs is PSPACE complete and we need powerful subclasses. (PSPACE complete problems are problems that can be solved using only polynomial space and are hard as any problem that can be solved using plynomial space. It's believed that this problem cannot be solved in polynomial time [15].) In this paper, our major concern is to give new subclasses of INDs and to characterize them.

[Example] In our example, we assume the following INDs:

Manager<O_Worker
Secretary<O_Worker
Director<Manager
Director<O_Worker

Figure 2 shows the AIS diagram which visualizes our database environment.

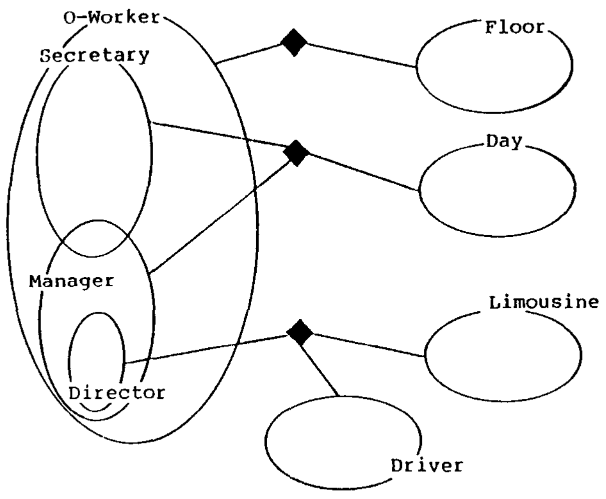

Fig. 2 AIS diagram with Inclusion Dependencies

Note that there may be an office worker who is both secretary and manager.

According to [7], INDs have the following inference axioms:

[I1] A<A for every entity type A
[I2] A<B and B<C imply A<C

Note INDs in [13] are more general than ours so that some computing intractabilty happens. Our axioms are restricted to entity sets and therefore much simpler. Similar approach is in [13] which discusses "unary" INDs and the interaction with FDs. Intuitively, these are the axioms on set inclusion and they can be illustrated using AIS diagram. Here we have our first result.

[Theorem 1] [I1] and [I2] are sound and complete with respect to INDs.

(Proof) We show here the theorem by propositional logic, that is, by relating IND A<B to a logical formula **A=>B**("=>" means implication); such technique is in [20]. We denote **A=>B** by **p** if p is A<B and the set by **X** if X is a set of formulas. For logical formula **A=>B**, we can assign "usual" boolean truth values; **A=>B** is defined true if **A** is false or **B** is true.

We have similar propositional axioms for [I1] and [I2] corresponded.

Suppose that X is a set of INDs and p an IND A<B. To prove the theorem, it is enough to show:

(i) p is a logical consequence of X iff **p** is a logical consequence of **X** (Equivalence Theorem)

(ii) **p** is a logical consequence of **X** iff **p** is derived from **X**

(iii) p is derived from X iff **p** is derived from **X**

Proof of (i): If p is not a logical consequence of X, there can be an entity x of type A but not of type B while X holds in the database. Now consider the truth assignment:

C is true if the entity is in $\lambda(C)$, false otherwise. Clearly **A=>B** is false. Assume some **C=>D** in **X** is false. **C** is true and **D** is false. The entity x is in $\lambda(C)$ by definition. But X holds in the database, then x must be in $\lambda(D)$, or **D** is true. Or, **p** is not the logical consequence.

Conversely, assume **p** is false, or **A** is true and **B** is false. Now we consider the database whose entity types appear exactly in X and p such that:

Given one entity x, for every entity type W, x is in $\lambda(W)$ if **W** is true under the assignment.

For each C<D in X, if x is in $\lambda(C)$, it must be in $\lambda(D)$ since **C=>D** is true and **C** is true. By assumption, x is in $\lambda(A)$ and not in $\lambda(B)$. That means p is not the logical consequence.

Proof of (ii): If-part (soundness) is clear, and we show the converse. It is sufficient to show the assignment which satisfies **X** but not **p**, when **p** is not derived from **X**.

Consider the assignment as follows:

- Assign false to **C** if **A=>C** is not derived from **X**.
- Assign true otherwise.

A=>B is false since **A** is true (**A=>A** is always obtained by [I1]) and **B** is false by assumption.

Suppose **C=>D** in **X** is false. **C** is true and **D** is false. That is, **A=>C** can be derived. As **C=>D** is in **X**, by [I2], **A=>D** can be also derived, or **D** must be true, contradiction.

Proof of (iii): Syntactical translation of the proof procedure shows the correctness. □

Note that [25] presents another proof.

[Example] In the above example, Director <O_Worker is redundant since it can be induced.

3.2 Extended Generalization

Consider the constraint {A*B<C}. This means that each entity both in $\lambda(A)$ and $\lambda(B)$ must be in $\lambda(C)$. Note this cannot be expressed by INDs in section 3.1. Similarly, {A<B+C} presents a new type of constraint which says that every entity in $\lambda(A)$ must be in $\lambda(B)$ or $\lambda(C)$. As this example shows, more than two types participate in new classes of the constraints. In this section, we give the rigorous definitions and the characterization [2].

INDs or Generalization may concern the "vertical" relationship, IS_A hierarchy, between two types. On the other hand, "horizontal" relationship may interact with vertical ones. For example, in our example, every one who is a secretary "or" a manager, must be an office worker.

First, the notion of expression is to define the above "horizontal" relationship. The expression of Intersection (Union respectively) is defined recursively:

- For entity type A, A is the expression.
- If W1 and W2 are both expressions, W1*W2 (W1+W2) is also an expression.
- There is no other expression except those using the above rules.

Using these expressions, two kinds of constraints are defined;

Intersection Extended Generalization (IXG) is a class of INDs which can involve the intersection expression instead of single types,

Union Extended Generalization (UXG) is similarly defined using the union expression.

We say that IXG $A_1*..*A_n<B_1*..*B_m$ (UXG $A_1+..+A_n<B_1+..+B_m$ respectively) holds if the intersection (union) of $\lambda(A_1),...,\lambda(A_n)$ is always a subset of the intersection (union) of $\lambda(B_1),...,\lambda(B_m)$.

For example, the comment above is specified by

Manager+Secretary<O_Worker.

Note [19] discusses updating on IXG and UXG framework, but doesn't show the characterization of these constraints. In this section we prove the existence of sound and complete inference axioms of IXG and UXG.

Now let us turn to consider some properties.

[IX1] $A_1*..*A_n<A_i$ for every i=1,..,n

[IX2] $A<B_1*..*B_m$ if and only if $A<B_i$ for every i=1,..,m

[UX1] $A_i<A_1+..+A_n$ for every i=1,..,n

[UX2] $A_1+..+A_n<B$ if and only if $A_i<B$ for every i=1,..,n

Also we extend the meaning of [I1] and [I2] to allow the introduction of expressions.

[lemma 1] Every formula of IXG (UXG respectively) can be transformed into a form such that the intersection appears only on left side (the union appears only on right side). In fact, [IX2] [UX2] can be applied for this "normalization". This can be done using one pass compiling technique, the space complexity is $O(n^2)$ where n is the description length. []

For example, {Manager+Secretary<O_Worker} is equivalent to {Manager<O_Worker, Secretary<O_Worker}. For another example, the set {A*B<C} and {A<B+C} are already normal.

[Theorem 2]
(1) [I1][I2][IX1][IX2] are sound and complete with respect to IXGs.
(2) [I1][I2][UX1][UX2] are sound and complete with respect to UXGs.
(Proof) see Appendix □.

The membership complexity problem consists of determining how fast a member can be derived from the given set of formulas. We have the following.

[Theorem 3] Testing membership of INDs, IXGs and UXGs takes time $O(k)$, $O(k)$ and $O(k^4)$ respectively where k is the description length of the given set.

(Proof) see Appendix □.

[Example] We add more constraints; Every one who is a Manager and a Secretary, should be a project leader. Leader may be an office worker or a driver. The new predicate is :

[Leader Project] says Leader a is affected to Project p.

The constraints are:

Manager*Secretary<Leader
Leader<O_Worker+Driver

Here we calculate non-redundant set of the constraints in our example (Fig. 3).

Manager<O_Worker
Secretary<O_Worker
Leader<O_Worker+Driver
Director<Manager
Manager*Secretary<Leader

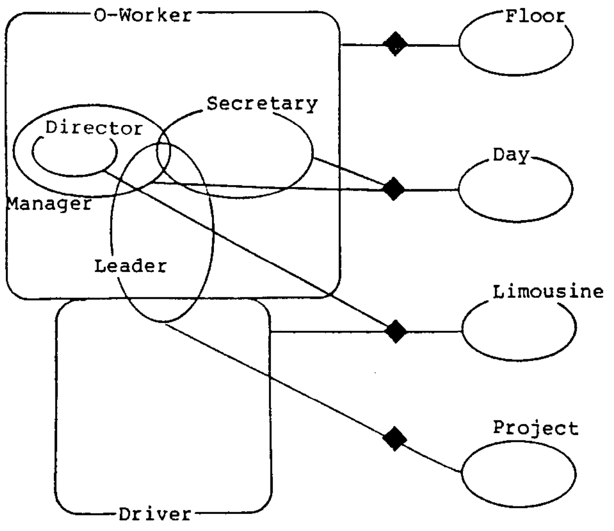

Fig. 3 AIS diagram with IXGs

4 EXCLUSION DEPEDENCIES

Natural extension of INDs covers the exclusion relationship. Here we address this problem.

We sometimes find exclusive relationship during the database design phase. Partitioning and categorizing entity sets [17] can treat this kind of relationships. On the other hand, in E-R model, this is out of conceren. Substantially, the problem here is, at best, to produce several design alternatives depending on how such constraints are captured. Moreover, redundancy or exclusion management is required. Unlike them, our initial approach is to model this kind of relationships and to give mathematical framework.

Exclusive relationship among entity types says that the entity sets never intersect. It is on the opposite side of INDs, and explicit specification is necessary.

Formally, given two entity types A and B, A is called exclusive to B, denoted by A||B, if and only if no common entity can exist in both A and B. This sort of constraints is called Exclusive Dependency or EXD [8]. [12] presents the detection mechanism for invalidity of a query using an exclusive relationship graph, but no characterization is found. ([8] gives the axiomatization.) Also [25] presents IND and EXD with the complete axioms.

Here we go back to the initial problem: consider how effective the EXD class is at database design. The following example shows the class is useless.

[Example] In our example (Figure 2), implicitly we assume that O_Worker, Driver, Day, Floor and Limousine are mutually exclusive. However, as we have the tool for the specification of these facts, following constraints must be stated explicitly:

O_Worker||Driver
O_Worker||Floor
O_Worker||Day
O_Worker||Limousine
Driver||Floor
Driver||Day
Driver||Limousine
Floor||Day
Floor||Limousine
Day || Limousine

In addition, we assume that a secretary can also be a manager;however no secretary is a director. This fact is expressed by:

Secretary || Director

Note the above set of descriptions are non-redundant.

[Example] In the case of Figure 3, descriptions about Project are added to the above:

Project||O_Worker
Project||Driver
Project||Limousine
Project||Day
Project||Floor

As easily seen, even by the help of the complete axioms, the designers must be annoyed to specify EXDs, not only because there are a lot of "unrelated" entity types, but also because they should look for vain relationship.

We find one more problem about EXD. Unlike the constraints discussed so far, there exists a set of EXDs which is unstatisfiable by any entity sets. For example, {A||B, A<B}. [8] calls this class of EXD "vacuous" EXDs. This means testing satisfiability should be performed.

We will stop to investigate EXDs any more, for, by the reasons above, the database designers must prefer co-EXDs to EXDs as shown in the next section.

5 CO-EXCLUSIVE DEPENDENCIES AND INTERACTION WITH IXG/UXG

5.1 Co-Exclusive Dependencies

We introduced and discussed several classes of extended INDs so far, and evaluated them from the view point of database design. Practically EXDs are less intuitive and effective than others, and are hard to be detected easily and naturally.

Here we propose as an extension of INDs which designers can get familiarized more easily with. We simply consider the negation of EXD as a constraint, and we call it co-EXD; that is, two entity sets "can" share entities. Such constraints are, in fact, extension of INDs. We assume that two entity sets must be exclusive if and only if the co-EXD cannot be implied. This idea is similar to Closed World Assumption in mathematical logic.

Given two entity types A and B, A is said to be coexclusive to B, denoted A#B, if not(A||B) holds, or i.e., two sets $\lambda(A)$ and $\lambda(B)$ can intersect.

We have to remark on co-EXD. In the case of INDs (IXGs, UXGs), they constrain databases in such a way that the only databases satisfying the constraints are said consistent. As for co-EXDs, they mean there can be databases satisfying the constraints and do not mean the entity sets must intersect. What co-EXDs mean is to constrain database when updating and sharing entities.

Let us compare co-EXD with EXD. The set {A<B, A||B} is never satisfied for any assignment, that is, there is no set assignment by which both formulas are "true". On the other hand, {A<B,A#B} is satisfiable, that is, there exists a set assignment by which both are true; $\lambda(A)$ and $\lambda(B)$ are not exclusive and $\lambda(A)$ is a subset of $\lambda(B)$. In this sense, we want to put an emphasis on the fact that local satisfaction of co-EXDs is always reflected to global view.

Note A#B is said true under the assignment if and only if there exists an element both in $\lambda(A)$ and $\lambda(B)$. Then A#B holds if and only if A#B can be true for some assignment. A#B is said to be a logical consequence of a set of co-EXDs and (extended) INDs, X, if and only if for every assignment which makes every formula of X true, A#B is true.

[Example] Again we show our example. In Figure 2, only the following description must be specified:

Manager#Secretary

Note that there is no need to describe "unrelated" relationship, compared to section 4.

Informative description is that there is no intersection between O_Worker and Driver. Here we don't describe this fact with certainty. In general, the more constraints exist, the more descriptions are required.

Some properties are easily proved (note A,B,C are all entity types):

[C1] A#A for every A
[C2] A#B implies B#A

[C3] A#B and B<C imply A#C
Using them we have:

[lemma 2]
(i) A<B implies A#B. In fact, since A#A holds ([C1]), we have A#B by [C3].
(ii) A<B and A<C imply B#C. In fact, A<B implies A#B by (i), and by [C3] B#C holds. □

[Theorem 4] [I1][I2][C1][C2][C3] are sound and complete with respect to INDs and co-EXDs.
(Proof) see Appendix □.

5.2 Co-EXD and the interaction with intersection expression

Co-EXD involving more than two entity types is not equal to a number of co-EXDs of two entity types. For instance, {A#B, B#C, C#A} does not say there exists a common entity in $\lambda(A)$, $\lambda(B)$ and $\lambda(C)$. Now we extend co-EXD to multiple entity types environment, denoted $\#(A_1,..,A_n)$, which says the intersection of $\lambda(A_1),..,\lambda(A_n)$ can not be empty. For example, {#(A,B,C)} says there can be an entity common in the three sets. Note that if #(A,B,C) then {A#B, B#C, C#A} holds but the reverse does not hold true.

More precisely, entity types $A_1.,,A_n$ are mutually co-exclusive if the following recursive conditions hold:
(i) $A_1\#A_2$ if n=2
(ii) $\#(A_1,..,A_{n-1})$; and $(A_1*..*A_{n-1}, A_n)$ when n>2.

[Example] In the case of Figure 3, we change Manager*Secretary<Leader into the two cases as follows:
X={#(Manager,Secretary,Leader)}
Y={#(Manager,Secretary), #(Manager,Leader)}
The former says that there can be a leader who is a manager and a secretary, the latter says that there is no such person, although there can be a leader who is a manager (see Fig.4). Hereafter we assume X.
To avoid the notational complexity, we define "equivalence" symbol (≡): we say E≡F if and only if E<F and F<E.

[lemma 3] Extended co-EXD can be expressed by binary co-EXDs and IXGs with an introduction of a new type. In fact, $\{\#(A_1,..,A_n)\}$ is equal to $\{B\#A_n, \#(A_1,..,A_{n-1}), B\equiv A_1*..*A_{n-1}\}$. As easily seen, the description length is still O(n). □

As for the above example, the additional constraint X is expressed as follows:
MgrSec≡Manager*Secretary
MgrSec#Leader
Manager#Secretary

The following properties clearly hold:
[M1] #(E) for every E, E is non-empty (this is rather a definition)
[M2] $\#(E_1,..,E_n)$ implies $\#(F_1,..,F_n)$ where $\{F_1,..,F_n\}$ is a permutation of $\{E_1,..,E_n\}$.
[M3] $\#(E_1,..,E_n)$ implies $\#(E_1,..,E_n,E_i)$ where i=1,..,n
[M4] $\#(E_1,...,E_n,F_1,...,F_m)$ and $F_1*..*F_m<G$ imply $\#(E_1,..,E_m,G)$ where n>=0, m>0

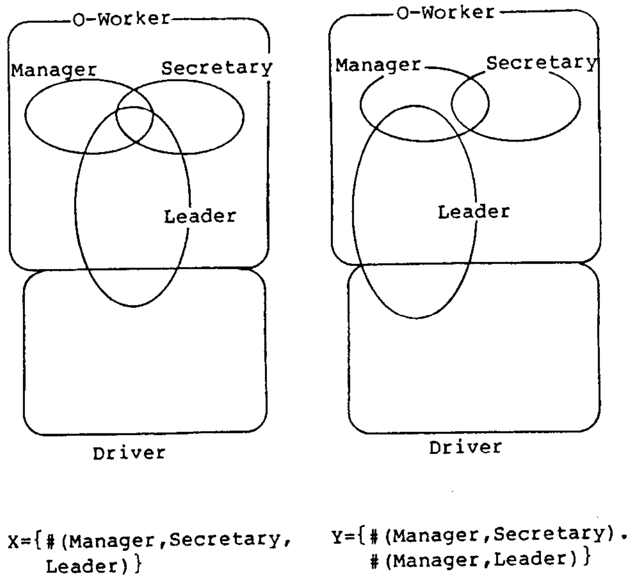
O-Worker
Manager
Secretary
Leader
Driver
O-Worker
Manager
Secretary
Leader
Driver
X={#(Manager,Secretary, Leader)}
Y={#(Manager,Secretary). #(Manager,Leader)}

Fig. 4 AIS diagram with co-EXDs

[M5] $\#(E_1,..,E_n)$ implies $\#(F_1,..,F_m)$ where $\{F_1,..F_m\}$ is a subset of $\{E_1,..E_n\}$ where all the symbols are intersection expressions.
Note [M2] corresponds to [C2], [M4] to [C3] and [M1][M3][M5] to [C1].

[Theorem 5] [M1]-[M5][I1][I2][IX1][IX2] are sound and complete with respect to IXGs and co-EXDs under intersection expressions.
(Proof) see Appendix □.

5.3 Interaction with union expression

Now we go back to binary co-EXDs in order to discuss UXGs. Our extended co-EXDs can involve union expression.

[Example] In the Figure 4, we again change our environment such that
(O_Worker+Driver)#Leader
instead of
Leader<O_Worker+Driver.
That means some other person, say Bob, may be a leader. The constraint is the above one. (Fig 5)
Let us consider co-EXDs and UXGs. Following example shows us how hard to draw the difference on the diagram.

[Example]
X={A#(B+C)}
Y={A#B, A#C}
Z={A#B, A#(B+C)}
Clearly X and Y are not equivalent though Y implies X. This is because every assignment satisfying Y makes all the elements of X, but the reverse direction is not. Also Z is redundant since the first implies the second in a sense that every assignment satisfying the first formula should satisfy the second. Note X cannot be expressed straightforward by the diagram though Y can.

[lemma 4] An extended co-EXD $(A_1+..+A_n)$

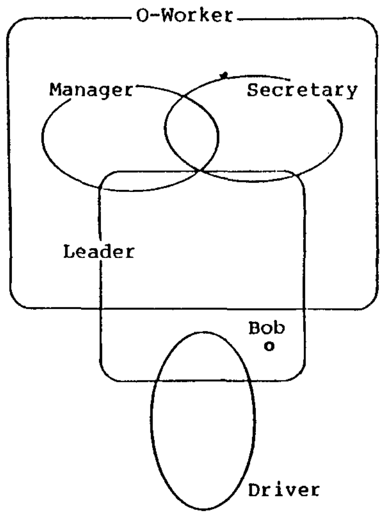

Fig. 5 AIS diagram with co-EXDs

$\#(B_1+..+B_m)$ is equivalent to the set $\{A\#B, A\equiv A_1+..+A_n, B\equiv B_1+..+B_m\}$. Also note the description length is still $O(n+m)$.□

For example, the above constraint is
$OD\equiv O\_Worker+Driver$
$Leader\#OD$

One more definition is needed. A normal UXG $A<B_1+..+B_m$ is called minimal if and only if $m=1$ and $A\equiv B_1$, or any proper subset $\{B_{j_1},..,B_{j_k}\}$ of $\{B_1,..,B_m\}$ does not satisfy $A<B_{j_1}+..+B_{j_k}$. Note that a given UXG may have several minimal forms.

[Theorem 6] Allowing union expression, [I1][I2][UX1][UX2] [C1]-[C3] are sound and complete with respect to co-EXDs and UXGs.
(Proof) see [2].□

[Example] The constraints in the figure 5 are non-redundant.

[Theorem 7]
(1) Testing membership of INDs and co-EXDs takes time $O(n^2)$ where n is the description length of the given set and the candidate.
(2) Testing membership of IXGs and co-EXDs takes time $O(n^2)$
(3) Testing membership of UXGs and co-EXDs takes time $O(n^5)$
(Proof) see [2].□

6 CONCLUSION

In this paper, we proposed new classes of Inclusion Dependencies as extension of Generalization based on entity and association concept. Various kinds of extensions were discussed, and four classes (IND, IXG, UXG and co-EXD) were evaluated from the viewpoint of database design. We presented the complete inference axioms for each class and the polynomial complexity of inference problems.

In order to construct databases according to a given scheme, it is helpful to consider the data structure on logical data storage as well as abstract data modelling. For, redundancy reduction of entity sets or association sets, and query optimization generally require fairly uniform and logical treatment on data storage. We call this level of database abstraction "realization structure". Non-First normal form Relation (NFR) [3], for example, is addressed for this purpose. Using the theory, domain can be organized systematically from more than one entity set using inter entity sets structure.

Several problems remain unsolved. First, we must relate IXGs, UXGs and co-EXDs to predicates in AIS and to compound value association [2][4]. Second, decomposing domains helps us to distribute data onto several sites in order to achieve parallel processing as in database machines. The new criteria for design may be presented.

REFERENCES

[1] H.Arisawa: A conceptual design of a database machine based on a new data model, E-R approach to system analysis and design, Proc. 1st E-R Conference, North-Holland, 523-540, (1979)
[2] H.Arisawa: Formal Approach to Database Design Based on Entity-Association Model, doctoral dissertation, Kyoto University, (1986)
[3] H.Arisawa, K.Moriya and T.Miura: Operations and the properties on non first normal form relational databases, Proc. VLDB, 197-204, (1983)
[4] H.Arisawa and T.Miura: Formal approach to database description, Proc. IEEE-COMPCON, 463-470, (1984)
[5] C.Beeri et al: Computational problems related to the design of normal form relational schemes, ACM-TODS, 4-1, 30-59, (1979)
[6] C.Beeri: Compatible attributes in a universal relation, Proc.ACM-PODS, 55-62, (1982)
[7] M.Casanova et al: Inclusion Dependencies and their interaction with functional dependencies, proc. ACM-PODS, 171-176, (1982)
[8] M.Casanova et al: Towards a sound view integration methodology, proc. ACM-PODS, 36-47, (1983)
[9] M.Casanova et al: Mapping uninterpreted schemes into E-R diagram, IBM J.Res.Develop., 28-1, 82-94, (1984)
[10] P.Chen: The entity-relationship model - towards unified view of data, ACM-TODS, 1-1, 9-36, (1976)
[11] E.F.Codd: A relational model of data for large shared data banks, CACM, 13-6, 337-387, (1970)
[12] Demolombe: Assigning meaning to ill-defined queries expressed in predicate calculus language, Advances in Databases, Plenum Press, 367-395, (1980)
[13] P.C.Kanellakis et al: Unary INDs have polynomial time inference problems, ACM STOC, 264-277, (1983)
[14] W.Kent: Choices in practical database

design, proc.VLDB, 165-180, (1982)

[15] M.R.Garey et al: Computers and Intractability - A guide to the theory of NP-completeness, Freeman, (1979)

[16] E.Mendelson: Introduction to mathematical logic (2nd), Van Nostrand, (1979)

[17] S.Navathe et al: View Representation in logical database design, proc. ACM-SIGMOD, 144-156, (1978)

[18] A.Pirotte: The Entity Association model, International Computing Symposium, North-Holland, 581-597, (1977)

[19] R.Reiter: On the integrity of typed first order databases, Advances in Databases, Plenum Press, 137-157, (1980)

[20] Y.Sagiv et al.: An equivalence between relational database dependencies and a fragment of propositional logic, JACM, 28-3, 435-453, (1981)

[21] E.Sciore: Inclusion dependencies and the universal instance, proc. ACM-PODS, 36-47, (1983)

[22] J.Smith et al: Database abstraction - Aggregation and Generalization, ACM TODS, 2-2, 105-133, (1977)

[23] J.D.Ullman: Principles of Database systems (2nd), Computer Science Press, (1982)

[24] C.Beeri et al: Comprehensive approach to design of relational database schemes, proc. VLDB, 196-207, (1984)

[25] P. Atzeni et al: Formal properties of net-based knowledge representation schemes, IEEE COMPDEC (1986)

APPENDIX

Proof of [Theorem 2]

By [lemma 1] we assume every formula is normal. The proof strategy is similar to [Theorem 1]. Moreover, we outline the proof of (1) and a similar proof holds for the case of union.

Care should be taken that we extend [I1] [I2] to the ones involving expressions. Relating $A_1*..*A_n<B$ to $\mathbf{A_1*..*A_n{=}{>}B}$, we have the corresponding set of logical formulas. The logical formula above is defined true if some $\mathbf{A_i}$ is false or **B** is true. Assume X is the given set of IXGs and p an IXG. Equivalence Theorem is proved similarly in [Theorem 1]. In order to prove completeness, we assume **p** is not derived from X, and show that there is an assignment which satisfies all **X** but not **p**.

Assign false to every **C** if $A_1*..*A_n<C$ cannot be derived from X. By [IX1], $\mathbf{A_1,..,A_n}$ are all true and **B** is false by assumption.

When $\mathbf{C_1*..*C_m{=}{>}D}$ is false, $\mathbf{C_1,..,C_m}$ are true and **D** false, that is, $A_1*..*A_n<C_i$ can be derived for $i=1,..,m$. By [IX2], $A_1*..*A_n<D$ is proved from X since $C_1*..*C_m<D$ is in X, **D** must be true, contradiction. □

Proof of [Theorem 3]

INDs and IXGs are much similar to functional dependencies, and [5] can be applied to our problem. In the case of UXGs, when deriving $A_1+..+A_n<B_1+..+B_m$, we want to calculate all the union expressions of the UXG which have $B_1+..+B_m$ on the right side. Then we will test whether the set contains $A_1+..+A_n$.

```
(1)  Y:={B_1,..,B_m}
(2)  repeat
(3)     for each C_1+..+C_l<D_1+..+D_h in X
(4)        if all of D_1,..,D_h are in Y
                then add C_1,..,C_l to Y
(5)  until (Y is not modified) or (Y contains all the elements of
           A_1,..,A_n)
(6)  if Y is not modified then return(FALSE) else return(TRUE)
```

The above algorithm calculates the set desired, since (1) represents [I1], (4) represents [I2] and [UX2], the latter of (5) means [UX1]. Also this halts in finite steps because the set Y is increasing monotonously and X is finite. (3) and (5) take O(k), and (4) takes $O(k^2)$. The loop (2)-(5) halts in O(k) times, for, Y has the limit length. In total, this takes time $O(k^4)$. □

Proof of [Theorem 4]

Soundness holds clearly and we show the completeness. Assume X is a set of INDs and co-EXDs, p an IND or co-EXD. In order to prove the completeness, we construct the set assignment which satisfies all of X but not p if p is not derived from X.

When p is an IND, [C1]-[C3] cannot derive new INDs and the theorem is reduced to [Theorem 1].

Assume p is a co-EXD A#B where A,B are both entity types.

Let $C_1,..,C_n$ be entity types in X and p. Consider the assignment as follows:

(i) Add x_{ij} to C_i, C_j if $C_i \# C_j$ can be derived from X.

(ii) Add x_{ij} to C_k if $C_i \# C_j$, $C_i<C_k$ are derived from X.

(iii) Repeat (i) and (ii) as many times as possible.

where x_{ij} are all distinct and $i \leq j$.

[i] The assignment is well-defined: The number of $\{x_{ij}\}$ are $O(n^2)$, and every C_i is increased monotonously. Then generating procedure halts in finite steps. It is easy to show the assignment is Finite Church Rosser, that is, there is the finite length procedure and the result is unique, not dependent on the applying sequence.

[ii] p is not satisfied: By (i), $\lambda(A)$ and $\lambda(B)$ are both nonempty. Since p is not derived from X, rule (i) cannot applied directly to A and B. Assume there is an element x_{ij} both in A and B. Then certainly we have $C_i \# C_j$, $C_i<A$ and $C_j<B$ (or $C_i<B$) derived from X. Then, we must have A#B by [C3] or [lemma 2].

[iii] Every formula in X holds: Co-EXD in X must hold because of (i). Assume $C_i<C_j$ is in X. Every element x_{ik} in C_i must be in C_j by (ii). Element x_{pq} in C_i must be C_p and C_q with $C_p \# C_q$, $C_p<C_i$ (or $C_q<C_i$). Then, by [I2], $C_p<C_j$ and $C_p \# C_q$ adds x_{pq} to C(j) by (ii). □