A Boolean Algebra Approach for Class Hierarchy Normalization

Yanchun Zhang *Jitian Xiao*

Department of Mathematics and Computing
The University of Southern Queensland
Toowoomba, QLD 4350, Australia
Email: {yan, jitian}@usq.edu.au

Abstract

Normalization in object-oriented design is much different from that in relational database design. Not only are the conceptual data model of object-oriented (o-o) models integrating richer structuring capabilities than that of relational models, but also the dependency constraints, attribute ranges and access paths in o-o models are more complex than that in relational models. In o-o models, inheritance semantics is expressed mainly by class hierarchy, and it is important to ensure and maintain an appropriate class hierarchy. In this paper, we propose a Boolean algebra approach for class hierarchy normalization. A class hierarchy normal form(CHNF) and an indexing model for class hierarchy are defined respectively. Some methods and algorithms, such as transformation from a non-CHNF class hierarchy to a CHNF one, are given.

Keywords object oriented databases, class hierarchy, normal forms, Boolean algebra.

1 Introduction

In traditional databases, a relational schema design begins with an initial schema, and ends with an equivalent one that is better in some respects. This better schema implies the fact that a schema with less *redundancy* and less *update anomaly* problem is preferable. The aim of solving the redundancy problem is to minimize the duplication of information stored in relations. Redundancies in relations cause update problems, such as insertion/deletion anomalies, which can also affect the performance of a database processing. In the literature, these redundancy and update problems have been solved to some extent by proposing a series of *relational normal forms* (such as 3NF, BCNF, 4NF and 5NF) and normalization procedures.

In object-oriented databases, it is an important issue to design a good and correct *conceptual schema*. These conceptual data models integrate richer structuring capabilities than the flat relational model by allowing the reuse of data and processes through inheritance, the construction of complex objects, and the identification of objects independently of their values(using the identity mechanism).

One can translate an existing schema from traditional databases into OODB [5], but a good relational schema does not necessarily result in a good schema in the object-oriented sense. Some concepts of relational formalization method can be used to describe object-oriented database design [8], but most of them are different.

In the context of normal forms and normalization, there are many differences between R models (flat relational and extended-relational data models) and O-O models (object-oriented data models). First of all, in R models, dependency constraints are restricted to attribute-attribute relationships (e.g., functional and multivalued dependencies). In O-O models it is augmented to include the concepts of identifier and other atomic attributes (such as pointers, etc.). Secondly, object attributes may be complex (or multivalued), allowing reference to instances of other objects. The attribute types can be sets, collections, lists and any other structured type. However, relational attributes can only be simple or multivalued or other nested sub-relations in R models. And thirdly, objects are uniquely identified by an object identifier that is assigned by the system and can not be changed during its life time. There is no notion of key attributes in O-O models, whereas key attributes in R model are necessary to normal forms. In addition, the concepts of aggregation, generalization and multi-inheritance should be considered in object-oriented normal forms.

Due to the above differences, the normal forms and normalization procedures for object-oriented models should be discussed independently from to that of R models. In the following sections, we propose a Boolean algebra approach for class hierarchy normalization. In Section 2, we first review the related concepts and properties of Boolean algebra, then in Section 3, discuss the normalization of class hierarchy which deals with the redundancies of the

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1–4, 1997.

schema definition (inter-object relationship). The conclusions will be included in Section 4.

2 Review on Boolean Algebra

Boolean algebra, in honour of George Boole who first set up a logic algebra of this type in 1854, is a algebra structure with very good properties. It is used in many areas of information processing and switch theory. Here, as a review, we list Boolean algebra and related concepts and properties [4, 10, 13].

Definition 1 [*Boolean algebra*] *A Boolean algebra is a six-tuple* $< B, +, *, ', 0, 1 >$, *where* B *is a non-empty set,* $+$ *and* $*$ *are binary operations on* B, $'$ *is a unary operation on* B *such that for all* $a, b \in B$, *we have* $a+b \in B$, $a*b \in B$, *and* $a' \in B$. *And the following axioms hold:*

1. *For all* a , $b \in B$, $a+b = b+a$; $a*b = b*a$
2. *For all* $a, b, c \in B$, $(a+b)+c = a+(b+c)$; $(a*b)*c = a*(b*c)$
3. *For all* $a, b, c \in B$, $a+(b*c) = (a+b)*(a+c)$; $a*(b+c) = (a*b)+(a*c)$
4. *For all* $a \in B$, $a+0 = a, a*1 = a$
5. *For each* $a \in B$, *there exists an element denoted* a' *and called the complement or negation of* a *in* B *such that* $a+a' = 1$ *and* $a*a' = 0$

Elements $0, 1 \in B$ *are called the smallest element and biggest element, respectively.*

When there is no danger of confusion, we refer to the Boolean algebra simply as B. The set B must contain at least the two elements 0, 1. The symbols 0 and 1 may be considered as nullary operations mapping from set B to 0, 1 in B.

Property 1 *Let* $< B, +, *, ', 0, 1>$ *be a Boolean algebra,* $x, y \in B$, *then*

(a). $(x')' = x$.

(b). $x*(x+y) = x$, $x+(x*y) = x$.

(c). $(x*y)' = x'+y'$, $(x+y)' = x'*y'$.

(d). $x+0 = x$, $x*1 = x$.

(e). $0 \neq 1$.

Definition 2 *In a Boolean algebra* $< B, +, *, ', 0, 1 >$, *we define a binary relation* $\preceq$ *on* B *by stipulating that* $x \preceq y$ *if and only if* $x*y = x$.

Property 2 *Let* $< B, +, *, ', 0, 1 >$ *be a Boolean algebra, and* $\preceq$ *be a binary relation on* B *defined as Definition 2, then*

(a). For all $x, y \in B$, $x \preceq y$ *if and only if* $x+y = y$

(b). For all $x \in B$, $x \preceq x$. *(Reflexivity)*

(c). For all x, y *and* z, $(x \preceq y)$ *&* $(y \preceq z) \rightarrow x \preceq z$. *(Transitivity)*

(d). $(x \preceq y)$ *&* $(y \preceq x) \rightarrow x = y$. *(Anti-symmetry)*

Definition 3 *A binary relation* R *on set* A *satisfying the following conditions*

(1). $(xRy$ *&* $yRz) \rightarrow xRz$.

(2). $(xRy$ *&* $yRx) \rightarrow x = y$.

is called as a partial order *on* A. *A partial order* R *on* A *is said to be* reflective *if and only if* xRx *for all* $x \in A$. *For a reflective partial order* $\preceq$ *on set* A, *we define* $x \prec y$ *if and only if* $x \preceq y$ *and* $x \neq y$.

Definition 4 *Let* $\preceq$ *be a partial order on set* A. $z \in A$ *is said to be an* upper bound *of a subset* $Y \subseteq A$ *if* $y \preceq z$ *for all* $y \in Y$. $z \in A$ *is said to be a* least upper bound (lub) *of a subset* $Y \subseteq A$ *if and only if*

(1) z *is an upper bound of* Y.

(2) $z \preceq w$ *for every upper bound* w *of* Y.

Similarly, $z \in A$ *is said to be a* lower bound *of a subset* $Y \subseteq A$ *if* $z \preceq y$ *for all* $y \in Y$. $z \in A$ *is said to be a* greatest lower bound (glb) *of a subset* $Y \subseteq A$ *if and only if*

(3) z *is a lower bound of* Y.

(4) $w \preceq z$ *for every lower bound* w *of* Y.

Property 3 *(1) Let* $\preceq$ *be a partial order on set* A. *For any subset* $Y \subseteq A$, Y *has at most one lub (glb).*

(2) Let $< B, +, *, ', 0, 1 >$ *be a Boolean algebra. For any* $Y \subseteq B$, Y *has exactly one lub and one glb.*

Example 1. Let A be a non-empty set. The power set (i.e., a set containing all subsets) of A is denoted as P(A). Under the usual operations of union, intersection, and complementation, and with ϕ and A as the distinguished elements 0 and 1, we have a Boolean algebra $< P$(A), $\cup, \cap, ', \phi$, A $>$, where $\cup, \cap$ are two binary operations, $'$ is a unary operation on P(A).

As a special case, we consider that A={a, b, c}. In this case, P(A) = { ϕ, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, A } . The smallest and biggest elements are ϕ and A, respectively. The binary relation $\preceq$ on P(A) defined as in Definition 2 is the conventional set relation $\subseteq$. Obviously, $\subseteq$ is a reflective partial order. For any subset Y of P(A), lub(Y) = $\cup_{S \in Y}$S, and glb(Y) = $\cap_{S \in Y}$S. For example, lub({ {a, b }, {a, c } }) = {a, b, c }, and glb({ {a, b }, {a, c } }) ={ a }. []

Usually, the Boolean algebra $< P(A), \cup, \cap, ', \phi, A >$ is called as the subset Boolean algebra of set A.

Definition 5 *Let $< B, +, *, ', 0, 1 >$ be a Boolean algebra and $\preceq$ a partial relation on B as defined in Definition 2.*

(1) An element $b \in B$, $b \neq 0$, is said to be an atom *of B if and only if, for all $x \in B$, the condition $x \preceq b$ implies that $x = b$ or $x = 0$.*

(2) Let $x, b \in B$, x is said to be a predecessor (descendant) of b if $x \preceq b (b \preceq x)$. x is said to be a direct predecessor of b if $x \prec b$, and there is not any other element $c \in B$ such that both $x \prec c$ and $c \prec b$ hold. If x is a direct predecessor of b, then we say that b is a direct descendant of x.

Definition 6 [*Boolean algebra graph*]
Let $< B, +, *, ', 0, 1 >$ be a Boolean algebra, B a finite non-empty set, and $\preceq$ a partial order relation on B as defined in Definition 2. A directed graph $< V, E >$ is said to be a Boolean algebra graph *of B, if*

(1) $V = B$,

(2) $E = \{ <x,y> \mid x, y \in B, y$ is a direct predecessor of $x \}$.

An element $x \in V$ is called a node, and an element $<x, y> \in E$ is called as a edge (from x pointing to y) of the Boolean algebra. If there exist an edge sequence $<x_1, x_2>, <x_2, x_3>, \ldots, <x_n, x_{n+1}>$ $(n \geq 1)$ such that $<x_i, x_{i+1}> \in E$ for all $1 \leq i \leq n$, we say that there exists a path *from x_1 to x_{n+1}.*

For simplicity, an undirected graph is usually used to replace the directed one.

Example 2. Consider the Boolean algebra in Example 1. Let A={ a, b, c}, then P(A) have three atoms {a}, {b} and {c}. A directed Boolean algebra graph of P(A) is shown in Fig.1(a), whereas a undirected Boolean algebra graph of P(A) is shown in Fig.1(b).

In Fig.1(a), an edge is expressed by an arrowed line which links nodes from one node to one of its direct predecessor nodes. {b, c}, for example, have three predecessors ϕ, { b } and { c }, two direct predecessors { b } and { c }, and one (direct) descendant { a, b, c }, and there are two arrowed line from {b, c} to nodes { b } and { c }. As mentioned before, we usually use Fig.1(b) for Fig.1(a). But we will keep in maid that Fig.1(b) is a directed graph and all lines are arrowed lines pointing upward from the bottom. []

Property 4 *Let $< B, +, *, ', 0, 1 >$ be a Boolean algebra, then*

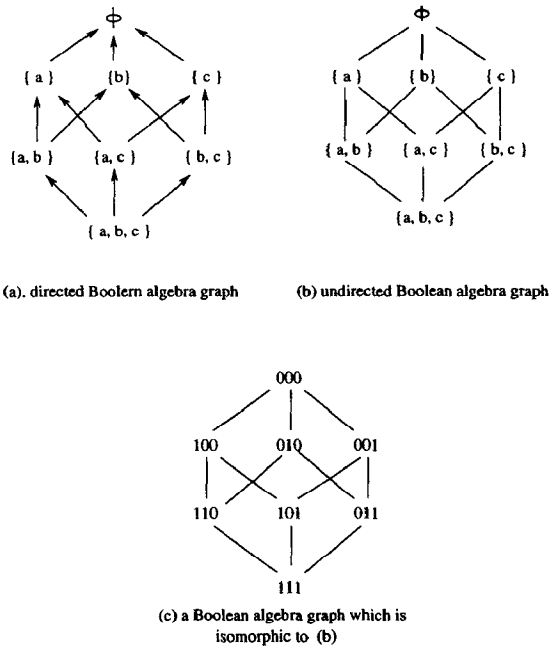


Figure 1: Boolean algebra graph

(1) $\#B = 2^k$, for some $k \geq 1$, and B has exactly k atoms.

(2) If x is a predecessor of y, there must be a path from y to x in B.

where $\#B$ indicates the number of elements in B.

Definition 7 *Given two algebras $< A, +, *, ', 0, 1 >$ and $< B, \oplus, \otimes, \bar{}, \theta, \zeta >$. If there exists a* one-to-one *function* f *from A to B such that*

(1) $f(x + y) = f(x) \oplus f(y)$;

(2) $f(x') = \overline{f(x)}$;

we say that $< A, +, *, ', 0, 1 >$ and $< B, \oplus, \otimes, \bar{}, \theta, \zeta >$ are isomorphic, f *is an isomorphism (isomorphic mapping) between A and B.*

Property 5 *If $< A, +, *, ', 0, 1 >$ and $< B, \oplus, \otimes, \bar{}, \theta, \zeta >$ are isomorphic, and* f *is the isomorphism between A and B, then*

(1) $f(x * y) = f(x) \otimes f(y)$;

(2) $f(0) = \theta$ and $f(1) = \zeta$.

Property 6 *All Boolean algebras with same number of elements are isomorphic.*

If two Boolean algebras are isomorphic, we may take them as the same Boolean algebra. The differences between them are only the symbol differences. All properties are the same.

Example 3. Let B_N be a set of all binary numbers whose word-length is N (N $\geq$ 1). That is, $B_N = \{a_1a_2...a_N | a_i \in \{0,1\}, 1 \leq i \leq N\}$. Then, $< B_N, \vee, \wedge, \sim, 0_N, 1_N >$ is a Boolean algebra, where $\vee, \wedge$ are conventional logical operation on binary numbers, i.e. logical *bit or* and logical *bit and*, $\sim$ is conventional logical 1-complement operation, 0_N is the number 00...0, and 1_N is the number 11...1.

Obviously, $\#B_N = 2^N$. For this Boolean algebra, the following properties hold:

(1) For every two elements x = $a_1a_2...a_N$ and y = $b_1b_2...b_N$,
lub({x,y}) = x $\vee$ y = $(a_1 \vee b_1)(a_2 \vee b_2)...(a_N \vee b_N)$,
glb({x,y}) = x $\wedge$ y = $(a_1 \wedge b_1)(a_2 \wedge b_2)...(a_N \wedge b_N)$.

(2) B_N is isomorphic with P(A) in Example 2 if #A = N.

(3) If we define partial order $\preceq$ on B_N as following: For every two elements x=$a_1a_2...a_N$ and y = $b_1b_2...b_N$, x $\preceq$y if and only if $a_i \leq b_i$ for 1 $\leq i \leq$ N, then all of the atoms of B_N are as following: 10...00, 010...00, ..., 00..10, 00...01.

In fact, property (1) can be easily extended to obtain *lub* and *glu* for any subset of B_N. When N = 3, the Boolean algebra graph of B_N is shown in Fig.1(c). []

We usually call the Boolean algebra of Example 3 as *bit Boolean algebra.*

3 Normal Form of Class Hierarchy

In O-O models, *inheritance semantics (isa semantics)* is expressed mainly by class hierarchy. It is important to ensure and maintain an appropriate class hierarchy structure to fully express the inheritance semantics. An improper class hierarchy structure will function initially but, when a class schema evolution occurs , may display an unreasonable structure that may cause loss of information, confusions in semantics and storage anomalies in implementation.

Here we describe an object-oriented model that will be used for the development of object-oriented normalization theory. Similar to relational models, we call the component set of a class definition an *attribute set.* In the following discussion, we use capital letters for attribute sets, and some constant attribute sets are given below where the brackets denote that the attribute is a complex one (multivalued attribute). A hypothetical class hierarchy graph is shown below in Fig.2, where the meanings of the capital letters are :

P–represents the attribute set of Person, such as name, [address];

E–represents the special attribute set of Employee, such as job, salary;

S–represents the special attribute set of Student, such as S-No, major;

R–represents the special attribute set of Researcher, such as [laboratory];

T–represents the special attribute set of teacher, such as title;

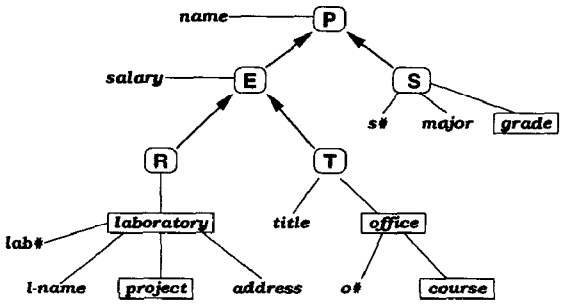
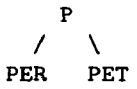


Figure 2: Class hierarchy

Here the special attribute set is the attribute set differing from that of its superclass's. We assume that a person may have several addresses, a researcher may work in several laboratories.

In this diagram, P, E, R, T and S are perceived as separate classes. They are graphically represented as large round squares. Their possible attribute sets are drawn beside them, and are linked to the class with a single line. When an attribute is a complex one, it is drawn inside a small square. For instance, the attribute salary of the class E is a simple attribute, whereas attribute laboratory of class R is a complex (multivalued) one. The relationship between superclass and subclasses is graphically represented by a directed line with an arrow point from subclass to its direct superclass. For instance, both R and T are subclasses of class E, and from each of them an arrowed line is drawn linking to E. It means that both R and T inherit all information of class E.

Example 4. We now extract the class hierarchy from Fig.2 (we are not concerned with the component attributes), and discuss the redundancy problem of schema definition. In Fig.2, if we only consider information from two kinds of persons, such as researcher and teacher, we may get following class hierarchy

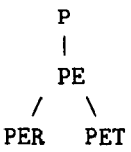


where node PER represents Researcher, meaning that it is composed of the attribute sets of P, E and

R, i.e. PER =P ∪ E ∪ R. In this representation, a node labeled ABC represents a class frame whose attribute set is composed of

- special attribute set C, and inherited attributes AB from its superclass if its superclass attribute set is AB; or
- special attribute set BC, and inherited attribute set A if its superclass's attribute set is A; or
- special attribute set ABC which inherits nothing from its direct superclass's attribute set (or, say, its superclass's attribute set is empty).

Similarly, PET represents Teacher. In the above representation, the attribute set P in PER and PET is inherited from its parent node (superclass), but E appeared in both PER and PET is redundantly defined. If we use following class hierarchy



then we get no redundancies in this schema definition, where PE denotes the attribute set composed of P and E. E can be defined for PE class, and will be inherited by PER and PET classes. E appeared in PER and PET is defined only once and inherited from PE. []

Generally, in object-oriented database applications, it is hard to design a perfect class hierarchy. Practically, it is difficult to design a class schema with no redundancy in class attribute definition. A redundant class schema will result in redundancy in object definition and object storage. A non-redundant class hierarchy is easy to maintain, even when in schema evolution time. On the other hand, for a given application, we can construct different class inheritances in different ways. How do we judge if a given class hierarchy is reasonable or not? To keep a clear class inheritance, it is necessary to develop a rule to detect whether a given class hierarchy is redundantly defined, and if so, how to transfer it into a non-redundant one.

3.1 CHNF and its related concepts

Based on the discussion above, we will present a set of definitions for normal form of class hierarchies. Later we will propose some procedures to detect and deal with redundant class schema definitions so that a better class schema with no such redundancy can be obtained. For convenience, we assume that any class hierarchy corresponds to a (directed, acyclic) rooted graph and several graph terms, such as node, predecessor etc., are used without declaration.

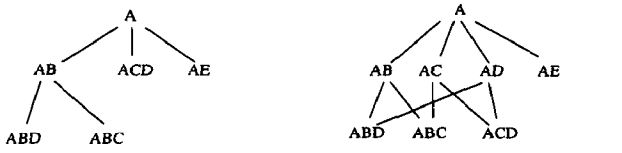
Definition 8 *A* local attribute set *of a node contains those attributes of the node which are not inherited from its superclass. A* full attribute set *of a node is a attribute set which includes attributes in its local attribute set, and attributes in its direct superclass's full attribute set . An* inherited attribute set *of a node is one which include attributes in its full attribute set but not in the local attribute set of the node.*

a full attribute set of a node in a class hierarchy includes not only the local attribute set of the node but also the local attribute sets of all its predecessors.

Definition 9 *Let $N_1, N_2, ..., N_n$ be all nodes of a given class hierarchy, and L_i be the local attribute sets corresponding to node N_i (i=1,2, ..., n). If $L_i \cap L_j = \phi$ for any i, $j(1 \leq i, j \leq n, i \neq j)$, then we say that the class hierarchy is in* class hierarchy normal form,*or say, in CHNF.*

Obviously, if a class hierarchy is in CHNF, there are no repeated components of the local attribute set of classes in the class hierarchy. A class hierarchy in CHNF has the least redundancy in class attribute definition, and can fully express the semantics of inheritance. However, it is an interesting task to detect the redundancy in a large class hierarchy, and to transfer a non-normal form class to a normal form one. We use a Boolean algebra model of class hierarchy to deal with the transformation.

Example 5. Two class hierarchies are given below in Fig.3. Different capital letters stand for different attribute sets, and they are not overlapping in attributes.



(a). A class hierarchy with redundancy

(b). A class hierarchy with no redundancy

Figure 3: Class hierarchies

In Fig.3(a), the intersection of the local attributes in node ABD and ACD (also ABC and ACD) are not empty (note that the local attribute set of node ABD is D, and the local attribute set of node ACD is CD. According to the definition, this class hierarchy is not in CHNF. In Fig.3(b), all of the intersection of local attribute sets between any two nodes are empty. Therefore, it is in CHNF. In fact, the class hierarchy in Fig.3(b) can be considered as a further normalization of that in Fig.3(a). []

In the above example, the class hierarchy in Fig.3(b) differs from that in Fig.3(a) in two respects. First, two nodes , AC and AD are added to the class hierarchy in Fig.3(b). Secondly, not only the node ACD but also nodes ABD and ABC become child nodes of AC and AD. In some applications, the object set of newly added class nodes may be empty, but the repeated definition of class attributes and the possible inconsistence resulting from it are avoided.

The transformation from a non-CHNF class hierarchy to a CHNF one is a three-step process. We first map the input class hierarchy to a Boolean algebra graph, then find nodes of its corresponding Boolean algebra, and at last remove the unnecessary nodes from the Boolean algebra graph to form a CHNF class hierarchy.

3.2 Mapping a class hierarchy to a Boolean algebra

Given a class hierarchy H, let $N_0, N_1, N_2, ..., N_n$ be all of the nodes of the class hierarchy with N_0 its root node. Let F = { $F_0, F_1, F_2, ..., F_n$} be the set of the full attribute sets of H, with F_i corresponding to node N_i (i=0,1, ... ,n). For any i, j, i≠ j, if N_i is a predecessor of N_j, then $F_i \subseteq F_j$ for F_i is included in the inherited attribute set of N_j. F_0 is the smallest attribute set because $F_0 \subseteq F_i$ for i=1,2,...,n. Now Let $L = \{L_1, L_2, ..., L_m\}$, for some m, be a set of possible local attribute sets of H excluding F_0 (note that $F_0 = L_0$ is the full attribute set as well as the local attribute set of N_0). If a class have two or more direct superclass, i.e. the case of multi-inheritance, its local attribute sets will have two or more element sets in L, each differs from others according to one of its direct superclasses. For example, in Fig.3(b), node ABD has two direct predecessor nodes AB and AD. Therefore, class ABD has two different local attribute sets, D, relative to its superclass node AB, and B, relative to its superclass AD.

Assume that $L_i \cap L_j = \phi$ for every i, j, i ≠ j,(1 ≤ i, j ≤ m), and let P(L) be the power set of L. Then we get a Boolean Algebra $\beta = \{P(L), \cap, \cup, \sim, \phi, \overline{A}\}$, with $\overline{A} = L_1 \cup L_2 \cup ... \cup L_m$, here ∩, ∪, and ∼ are conventional operators of sets, i.e. union, intersection and complement.

In the above Boolean algebra β, if we add the attribute set $F_0 = L_0$ to every element of P(L),then we can get another Boolean algebra **B** that is isomorphic to β. **B**= {P'(L),∩, ∪, ⋄, $F_0, \overline{L}$ }, with P'(L)={ X'| $X' = F_0 \cup X, X \in P(L)$}, $\overline{L} = F_0 \cup \overline{A}$, F_0 is the smallest element of this Boolean algebra, $\overline{L}$ is the biggest one, $\diamond(X) = \sim (X) \cup F_0$, and ∩, ∪ are conventional operators of sets.

Because all local attribute sets (excluding F_0) of the class hierarchy are in L, we conclude that for any i ($i \leq n$), there must be a set L' ∈P'(L) such that $F_i = \bigcup_{S \in L'} S$. It means that each of the full attribute sets of H will correspond to one of the nodes in Boolean algebra **B**. We can create a mapping from F of H to P'(L) of Boolean algebra **B** as following:

a). F_0 corresponds to $\{F_0\}$;

b). for every i (i=1, 2, ...,n), let $F_i = F_0 L_{i_1} L_{i_2} ... L_{i_k}$, then F_i corresponds to set $\{F_0, L_{i_1}, L_{i_2}, ..., L_{i_k}\}$ in P'(L), where L_{i_j} be a local attribute set of node N_i or some predecessor node of N_i.

Example 6. Consider the example in Fig.3(b). The class hierarchy has 8 nodes. The full attribute sets corresponding to those nodes are A, AB, AC, AD, AE, ABD, ABC, ACD. The local attribute sets corresponding to those nodes are A, B, C, D, E, (B,D), (B,C), (C,D), where attribute sets in parentheses indicate that that node have multi-parent nodes (expressing multi-inheritance). Different local attribute sets in this example are A, B, C, D, E, and A corresponds to root node of H. The set of P'(L) in Boolean algebra **B** is {{A}, {A,B}, {A,C}, {A,D}, {A,E}, {A, B, C}, {A, B, D}, {A, B, E}, {A, C, D}, {A, C,E}, {A, D, E} ,{A, B, C, D}, {A, B, C, E}, {A, B, D, E}, {A, C, D, E}, {A, B, C, D,E}}.

The mapping f from the full attribute sets of the class hierarchy to P'(L) is defined as following: f(A)={A}, f(AB)={A, B}, f(AC)={A, C}, f(AD)={A, D}, f(AE)={A, E}, f(ABD)={A, B, D}, f(ABC)={A, B, C}, f(ACD)={A, C, D}. Note that every full attribute set of a node in H corresponds uniquely to a node in Boolean algebra **B**, the mapping f is a *one-to-one* mapping. []

If $L_i \cap L_j \neq \phi$ for some i, j, i ≠ j,(1 ≤ i, j ≤ m), we can get the full attribute sets as before but the local attribute sets of H need to be changed so that the resultant P'(L) can be mapped as before. Actually, the new local attribute sets can be obtained by splitting the original local attribute sets. Let $L = \{L_1, L_2, ..., L_m\}$ be the original local attribute sets excluding F_0. We draw L_i, L_j out of L, let $L_{m+1} = L_i \cap L_j$, $L_{m+2} = L_i - L_{m+1}$, $L_{m+3} = L_j - L_{m+1}$. Then we replace L with $L' = (L - \{L_i, L_j\}) \cup \{L_{m+1}, L_{m+2}, L_{m+3}\}$. If there exists $L'_k, L'_l \in L'$, and $L'_i \cap L'_j \neq \phi$ for some k, l, k ≠ l, we can repeat the same process until a new set $L^{new} = \{L_1^{new}, L_2^{new}, ..., L_{q'}^{new}\}$ appears, with $L_i^{new} \cap L_j^{new} = \phi$, for every i, j, i≠j. Note that all of the attributes of H are maintained, and all are included in L^{new}. The following algorithm PNL will produce the non-intersected sets L^{new} from L.

The mapping process from F to P'(L^{new}) can be used without any change.

Algorithm 1. PNL:Producing a non-intersected local attribute sets.

Input: A class hierarchy H, with local attribute

sets L=$\{L_1, L_2, ..., L_m\}$, $F_0 \bar{\in}$ L is
the local attribute set of the root node of H.

Output: non-intersected local attribute sets L^{new} = $\{L_1^{new}, L_2^{new}, ..., L_q^{new}\}$, $\cup_{l \in L^{new}} l = \cup_{l \in L} l$. $L_i^{new} \cap L_j^{new} = \phi$ for any i, j, i≠ j ($1 \leq i, j \leq q$).

Begin
if $L_i \cap L_j = \phi$ for every i, j ($1 \leq$i, j≤q)
then let L^{new} = L; **return** ;
end_if
L' = L;
let X=$L_1 \cap L_2 \cap ... \cap L_m$;
if X≠ ϕ
then $F_0 = F_0 \cup$ X; L' = ϕ;
for i=1 **to** m **do**
$L_i' = L_i$ - X; $L' = L' \cup \{L_i'\}$;
end_for
end_if .
do while exist i, j, i≠j such that $L_i' \cap L_j' \neq \phi$
$L_{m+1}' = L_i' \cap L_j'$; $L_i' = L_i' - L_{m+1}'$;
$L_j' = L_j' - L_{m+1}'$; m=m+1;
end_do
let L^{new} =L'; **return** ;
End

Example 7. Consider the class hierarchy H in Fig.3(a), and L={B, D, C, CD, E}. L^{new}= {B, C, D, E}, same as the local attribute sets in Fig.3(b) (note that in this example, F_0 =A $\bar{\in}$L). Therefore, the corresponding Boolean algebra **B** is the same as that in Example 6, and the mapping from H to **B** can be similarly constructed. []

Definition 10 *Given a class hierarchy H. Let F be the set of all full attribute sets of H, L = $\{L_1, L_1, ..., L_m\}$ be the set of all local attribute sets of H, $L_0 = F_0 \bar{\in}$ L the local (and full) attribute set of root node of H, L^{new} the non-intersected local attribute set generated by Algorithm PNL. We call* **B**= $\{P'(L^{new}), \cap, \cup, \diamond, L_0, L^{new} \cup L_0\}$ *as the Boolean algebra corresponding to class hierarchy H.*

Theorem 1 *Given a class hierarchy H and the corresponding Boolean algebra* **B**= $\{P'(L^{new}), \cap, \cup, \diamond, L_0, L_{new} \cup L_0\}$. *Let $F = \{F_0, F_1, ..., F_n\}$ be the set of all full attribute sets of H. For any $F_i \in F$, there is one node $N_{i'} \in P'(L^{new})$ such that $F_i = \{F_0 L_{i_1} L_{i_2} ... L_{i_k}\}$, and $N_{i'} = \{F_0, L_{i_1}, L_{i_2}, ..., L_{i_k}\}$ and F_i corresponds to $N_{i'}$.*

Proof : *Let $L = \{L_1, L_2, ..., L_m\}$ be the set of all local attribute sets generated by Algorithm PNL ($L_i \cap L_j = \Phi$ for every i, j, $1 \leq i, j \leq m, i \neq j$). $L_0 = F_0 \bar{\in}$ L. If $L_{i_j} \in L$ for every j ($1 \leq j \leq k$), then, according to the definition of a power set, there exists a set $\{L_{i_1}, L_{i_2}, ..., L_{i_k}\} \in P(L^{new})$. Therefore, node $N_{i'} = F_0 \cup \{L_{i_1}, L_{i_2}, ..., L_{i_k}\} = \{F_0, L_{i_1}, L_{i_2}, ..., L_{i_k}\} \in P'(L^{new})$ and F_i corresponds $N_{i'}$.*

Assume that $L_{i_j} \bar{\in} L$ for some j. According to the Algorithm PNL, there must be local attribute sets $L_{i_j 1}, L_{i_j 2}, ..., L_{i_j q} \in L$ such that $L_{i_j} = L_{i_j 1} \cup L_{i_j 2} \cup ... \cup L_{i_j q}$, that is, L_{i_j} was split into q sub-attribute sets ($q \geq 1$). Re-number all of those local attribute sets we can get local attribute sets $L_{i_1'}$, $L_{i_2'}$, ..., $L_{i_p'} \in L$, satisfying $L_{i_1'} \cup L_{i_2'} \cup ..., \cup L_{i_p'} = L_{i_1} \cup L_{i_2} \cup ... \cup L_{i_k}$ and $\{L_{i_1'}, L_{i_2'}, ..., L_{i_p'}\} \in P(L^{new})$. Thus $N_{i'} = \{F_0\} \cup \{L_{i_1'}, L_{i_2'}, ..., L_{i_p'}\} = \{F_0, L_{i_1'}, L_{i_2'}, ..., L_{i_p'}\} \in P'(L^{new})$ and F_i corresponds to $N_{i'}$. []

In the following discussion, we will use L for L^{new}.

3.3 Indexing and Operations on Class Hierarchy

In constructing a class hierarchy, we need to know the current level of a node, and answer the following questions: How can we find all or some of the predecessor nodes for a given node in the class hierarchy? How many direct predecessor nodes does the current node have? And which node is the direct predecessor of a given node? Which one is the predecessor before the direct predecessor of a given node? Is there any easy way to find them? We will use the Boolean algebra model to answer the questions.

3.3.1 Indexing on Class Hierarchy

Logical operations on binary numbers are much easier than those on the attributes of a class hierarchy. So, in many cases, people seek to map sets to some binary numbers and transfer the operations on sets to those on binary numbers. In the following, we will discuss mapping from a class hierarchy to a proper bit Boolean algebra, and to some extent, transfer the operations on a class hierarchy to those on binary numbers of the bit Boolean algebra.

For a given class hierarchy H, by using the Algorithm PNL, we can obtain a non-intersected local attribute set L (here we use L for L^{new}). Let F = $\{F_0, F_1, ..., F_n\}$ be the set of all full attribute sets in H, L = $\{L_1, L_2, ..., L_N\}$ be the set of all local attribute sets excluding that of the root node of the class hierarchy, $F_0 = L_0 \bar{\in}$ L be the full attribute set as well as the local one. Now create the mapping f_N from F to the bit Boolean algebra $< B_N, \vee, \wedge, \sim, 0_N, 1_N >$ as follows:

(1) define $f_N(L_0) = f_N(L_0)$= 00...0;

(2) For every $L_i \in$ L, $L_0 \cup L_i \subseteq F$, define $f_N(L_0 \cup L_i) = 2^{i-1} = \underbrace{00\ldots0}_{N-i} 1 \underbrace{00\ldots0}_{i-1}$, (i=1, 2, ..., N).

(3) For $F_i = L_0 L_{i_1} L_{i_2} ... L_{i_k}$, define $f_N(F_i) = f_N(L_0 \cup L_{i_1}) \vee f_N(L_0 \cup L_{i_2}) \vee ... \vee f_N(L_0 \cup L_{i_k})$, (i=1, 2, ..., N).

We call the 2-tuple $< B_N, f_N >$ the *Indexing Model* of the given class hierarchy, where B_N indicates the bit Boolean algebra. The number value of $f_N(F_i)$ is called the *index* of the node whose full attribute set is F_i. It seems that for a given class hierarchy, by rearranging the order of the local attribute sets of H, we can obtain many different indexing models for H. But we have the following theorem:

Theorem 2 *For a given class hierarchy H, the Indexing Model of H is unique in the sense of isomorphism.*
Proof : *By using Algorithm PNL, we can get a non-intersected local attribute set* $L = \{L_1, L_2, ..., L_N\}$. *L is determined uniquely by given class hierarchy H. Therefore, N is determined uniquely by H. According to Property 6, the indexing model of H is unique in the isomorphism sense. []*

For convenience, we can permute the local attribute sets of L in *alphabetical order*. In this way, every class hierarchy will have a unique indexing model.

Example 8. Consider the class hierarchy H in Fig.3(a), where F = { A, AB, ACD, AE, ABD, ABC}, L = {B, CD, E, D, C}, L^{new} = {B, C, D, E}, and N = 4. According to the method described above, we have:
$f_4(A) = 0000$,
$f_4(AB) = 0001$, $f_4(AC) = 0010$, $f_4(AD) = 0100$, $f_4(AE) = 1000$,
$f_4(ABD) = f_4(AB) \vee f_4(AD) = 0001 \vee 0100 = 0101$, $f_4(ABC) = f_4(AB) \vee f_4(AC) = 0001 \vee 0010 = 0011$, $f_4(ACD) = f_4(AC) \vee f_4(AD) = 0010 \vee 0100 = 0110$.
so node AB has a index 0001, and node ABD has a index 0101, and so on. Note that nodes AC and AD do not exist in the class hierarchy, and we will add these nodes into the class hierarchy later. In this way, every node in H has an unique *index* which belongs to B_N of indexing model of given class hierarchy. It is clear that the class hierarchy in Fig3.(b) has the same indexing model in the sense of isomorphism. In fact, for any binary number $a_1a_2a_3a_4 \in B_4$, it will determine one node through this index, no matter if the node exists in H. For example, 1110 will be the index of node ABCD, although it is not existed in H. []

3.3.2 Operations on Class Hierarchy

To maintain a proper class hierarchy, some operations on class hierarchy need to be considered. The indexing model of class hierarchy is useful and convenient for expressing the following operation.

Let F_i be the full attribute set of node N_i, $f_N(F_i) = a_1a_2...a_N$ be the index of N_i. $f_N(F_j) = b_1b_2...b_N$ be the index of N_j. Following are some operations on class hierarchies:

(1). $\mathbf{Gl}(N_i)$: get the level of a given node N_i in H.
$\mathbf{Gl}(N_i) = \sum_{i=1}^{N} a_i$;

(2). $\mathbf{Gcp}(N_i, N_j)$: get index of the common predecessor of any two given nodes.
$\mathbf{Gcp}(N_i, N_j) = f_N(F_i) \wedge f_N(F_j) = (a_1 \wedge b_1)(a_2 \wedge b_2)...(a_N \wedge b_N)$.

(3). $\mathbf{Ccp}(N_i, N_j)$: construct an index of the common predecessor for two nodes.
$\mathbf{Ccp}(N_i, N_j) = f_N(F_i) \vee f_N(F_j) = (a_1 \vee b_1)(a_2 \vee b_2)...(a_N \vee b_N)$.

These operations provide some easy ways to deal with locating classes (and objects) in class navigation, especially when the class hierarchy is in CHNF. For example, consider the class hierarchy in Fig.3(b). According to Example 8, the common predecessor of node ABD and ACD is AD. We can easily get the common predecessor by using the operation Gcp (ABD, ACD) = f_4(ABD) $\wedge f_4$(ACD) = 0101 $\wedge$ 0110 = 0100, that is the index of AD. The method can be used similarly to get a common descendant of any two nodes. In constructing a CHNF of a given class hierarchy, when we need to add new nodes to the CHNF of given class hierarchy, these operations will work efficiently.
Other similar operations are:
(4). Get the number of the predecessor nodes for a given node in H.
(5). Get index of all the predecessors for a given node in H.
(6). Get the common descendant of any two or more given nodes.
The detailed expressions for the operations are omitted. By using the indexing model of class hierarchy, operations are all transferred to binary operations instead of those on attribute sets. It is clear that all (except operation 5) of these operations have constant complexities, that is, O(1).

3.4 Construction of CHNF

In Section 3.2, we discussed the mapping from a given class hierarchy to a proper Boolean algebra. We know that any boolean algebra has 2^k nodes, for some k. Our idea is to find a boolean algebra with proper nodes and with its nodes information rich enough to be mapped to from given class hierarchy using the method discussed. In this sense, any class hierarchy H will corresponds to one Boolean algebra B_H uniquely.

After the mapping between a class hierarchy H and its corresponding algebra B_H has been created, the nodes in B_H can be divided to three groups:

$P_1(L)=\{n \mid \exists\, l, l \in L \wedge n \in P'(L) \wedge f(l)=n\}$

$P_2(L)=\{m \mid \exists\, l, l \in L \wedge m \in P_1(L) \wedge \exists\, n(\, n \in P'(L) \wedge m=pre(n)) \wedge f(l)=n\,\}$

$P_3(L)=P'(L) - P_1(L) - P_2(L)$

where L, $P'(L)$ and f have the same meanings as in Section 3.2, $m=pre(n)$ means that m is one of the predecessors of n. For any node in $P_2(L)$, there exists at least one of its descendant nodes in given class hierarchy. Nodes in $P_3(L)$ are not useful any more, and we need not to care about them.

If a given class hierarchy H is in CHNF, $P_2(L)$ in its B_H will be empty. In the case that $P_2(L)$ is not empty, we need to transfer H to a new class hierarchy so that $P_2(L)$ become empty. In fact, for H, we only need to add nodes whose local attribute sets corresponding to the nodes in $P_2(L)$. In this way, we will complete the transformation from H to CHNF of H. The algorithm MARK and TRANSFER will complete this transformation.

Algorithm 2. MARK: Create mapping from given class hierarchy H to its corresponding Boolean algebra **B**, and mark two class of nodes of **B**.

Input : A class hierarchy H; non-intersected local attribute sets $L=\{L_1, L_2, ..., L_m\}$; $F = \{F_0, F_1, ..., F_n\}$; $F_0 \in L$ is the local attribute set of the root node of H, $n \leq m$.

Output : a Boolean algebra **B** whose nodes corresponding to full attribute sets of H are marked as one category(mark1); and all their predecessors not marked by mark1 belong to second category(mark2).

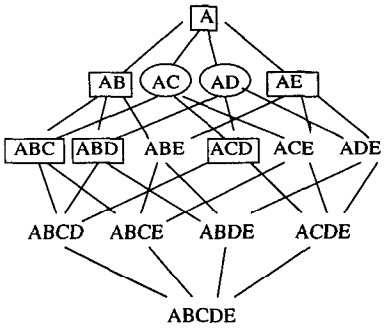
Begin

Generate P(L), the power set of L;
let $P'(L)=\{ X' \mid X' = F_0 \cup X, X \in P(L) \}$;
let $\overline{L} = \{ Y' \mid Y' = F_0 \cup Y, Y \in L\}$;
Create Boolean algebra $\mathbf{B}=\{P'(L), \cap, \cup, \diamond, F_0, \overline{L}\}$;
for i=0 **to** n **do** /* generate set $P_1(L)$ */
  let $F_i = L_{i_1} L_{i_2} ... L_{i_k}$;
  Find the node $N \in P'(L)$ of **B** whose value is $V_N=\{ L_{i_1}, L_{i_2}, ..., L_{i_k}\}$, and mark1(N);
end_for
for any node N' in **B** **do** /*generate set $P_2(L)$*/
  if N' not marked $\wedge \exists\, N'_d \in \mathbf{B}$ (N'_d marked $\wedge$ N'_d is a descendant of N')
    then mark2(N');
  end_if ;
end_for ;
return (**B**);

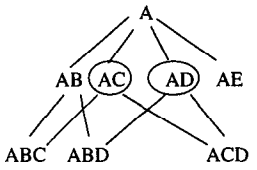
End

In the above algorithm, the sets in $P_1(L)$ of **B** are marked by function *mark1()*, and sets in $P_2(L)$ by function *mark2()*. They are useful in generating CHNF of the given class hierarchy H.

Example 9. Consider the class hierarchy in Fig.3(a) again. L={B, D, C, CD, E}, and L^{new}= {B, C, D, E} (note that F_0 =A $\in$L).



(a)



(b)

Figure 4: Boolean algebra graph of a class hierarchy

We can get that $P_1(L^{new})$={{A}, {A, B},{A, B, C}, {A, B, D}, {A, C, D}, {A, E}}, $P_2(L^{new})$={{A, C}, {A, D}}. A Boolean algebra graph is given in Fig.4(a), in which we marked the sets in $P_1(L^{new})$ with square boxes , and the sets in $P_2(L^{new})$ with round boxes, respectively. For simplicity, a set of attributes is expressed by a continuous string of those attributes in the nodes of **B** , e.g. the set of {A, B, D} is expressed by string ABD. All of the sets in $P_3(L^{new})$ are not marked. After removing sets of $P_3(L^{new})$ from the Boolean algebra graph we get a sub-graph of the Boolean algebra showed in Fig.4(b). Actually, adding two nodes whose attribute sets are AC and AD to a proper position within class hierarchy we can get a new class hierarchy. This new class hierarchy must be the CHNF of the class hierarchy given in Fig.3(a) because it will result in an empty $P_2(L^{new})$. The CHNF of the class hierarchy will be same as that in Fig.4(b), where sets in round boxes are newly added. []

The following algorithm will create CHNF of any class hierarchy, thus finishing the normalization of a class hierarchy.

Algorithm 3. TRANSFER: Transfer a given class hierarchy to one in CHNF

Input : A class hierarchy H.

Output : CHNF of the given class hierarchy H.

Begin

Generate the local attribute sets L (excluding F_0, the local attribute set of the root);
Generate the non-intersected local attribute sets L^{new} of L (by **PNL**), and use L for L^{new}.
Mapping nodes from H to its corresponding boolean algebra B_H and mark the related nodes of **B** (by Algorithm MARK).
let CHNF=ϕ;
for any node N in B_H **do**
if N marked by *mark1()* **or** *mark2()* in B_H
then create a new node H_N; copy full attribute set of N to H_N.
CHNF = CHNF $\cup$ $\{H_N\}$;
end_if
end_for
for any node H_N in CHNF **do**
find the parent node N' of H_N in CHNF (according to N in B_H), and index from H_N to N'; /*create link between nodes*/
end_for
End

A careful reader will find that the algorithms given above can be improved greatly, e.g. partially generate the Boolean algebra graph and so on. Here we are just concerned with the behaviour of the algorithms. The evaluation of the performance and the improvement of the algorithms go beyond the range of this paper.

4 Conclusions

In this paper, a formal method for normalization of class hierarchy is presented. This could lay a foundation for constructing and maintaining class hierarchies. A class hierarchy in CHNF has no redundant attribute definition, and is better than that of a non-CHNF one in that it can fully and clearly express inheritance semantics. The Boolean algebra model is a powerful tool for class hierarchy normalization. By creating the mapping from a given class hierarchy to a proper Boolean algebra, we conclude that any class hierarchy corresponds uniquely to a subset Boolean algebra. On the basis of Boolean algebra, we give three algorithms to complete the transformation from a non-CHNF class hierarchy to a CHNF one. The indexing model of a class hierarchy is another Boolean algebra which can be easily used to determine the predecessors or descendants of classes in a class hierarchy. It will be helpful for class navigation. Boolean algebra can also be used to determine where a new class will be laid when it is added to the class hierarchy.

Acknowledgement

We would like to thank Ms Anne Fuller for her comments and suggestions on an earlier version of this paper.

References

[1] C. Beeri. A Formal Approach to Object-oriented Databases *Data & Knowledge Engineering* , Vol.5, 1990, pp.353-382.

[2] S. Ceri and G. Gottlob. Normalization of relations and Prolog. *CACM*, pages 524–545, June 1986.

[3] J. Diederich and J. Milton. New methods and fast algorithms for database normalization. *ACM Transactions on Database Systems*, 13(3):339–365, September 1988.

[4] S. S. Epp. *Discrete Mathematics with Applications.* PWS Publishing Company, 1995, second edition.

[5] J. Fong. Mapping Extended Entity Relationship Model to Object Modeling Technique. *SIGMOD RECORD*, 18(3), Sept. 1995.

[6] S. Khoshafian. Object-oriented Databases. John Wiley & Sons, Inc., 1993.

[7] W. Kim. Objected-Oriented Database: Definition and Research Directions *IEEE Transection on Knowledge and Data Engineering*, Vol.3, September 1990, pp.327-341.

[8] B. S. Lee. Normalization in OODB Design. *SIGMOD RECORD*, 23(3), Sept. 1995.

[9] D. .Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[10] E. Mendelson. *Theory and Problems of Boolean Algebra & Switching Circuits.* McGRAW-HALL BOOK COMPANY, 1970.

[11] W.Y. Mok, Y. Ng and D.W. Embley. A Normal Form for Precisely Characterizing Redundancy in Nested Relations *ACM Transection on Database systems*, Vol.21, No.1 March 1996, pp.77-106.

[12] Z.M. Ozsoyoglu and L. Yuan. A New Normal Form for Nested Relations *ACM Transections on Database Systems*, Vol.12, No.1 1987, pp.111-136.

[13] R. R. Stoll. *Set Theory and Logic.* W. H. Freeman and Company, 1963.

[14] Y. Zhang and M.E. Orlowska. An improvement on the automated tool for relational database design. *Information Systems*, 15(6), 1990.