データベース・システム 82－8
（1991. 3. 11）

木構造複合オブジェクトの形式的モデル：SCORE

姜 世杰 大保信夫 山口和紀 北川博之 鈴木 功
筑波大学

木構造複合オブジェクトを記述するための形式モデルとして、文法的データモデルSCOREが提案されている。SCOREは、複合オブジェクトの記述に文脈自由文法を用い、文脈自由文法により定義されたスキーマに従った木構造をもつ複合オブジェクトを扱っている。この結果、一つの共通な枠組みの下で汎化、統合と再帰的な構造を表現することが可能となる。複合オブジェクトを操作するため、代数的データ操作言語ARESが定義されている。ARESの持つ四つのオペレータは強力なデータ操作機能を保証している。SCOREは一般的な木構造複合オブジェクトを扱うための理論的なベースとなることが期待できる。

SCORE: A FORMAL DATA MODEL FOR HIERARCHICAL COMPLEX OBJECTS

Shi-Jie Jiang Nobuo Ohbo Kazunori Yamaguchi Hiroyuki Kitagawa Isao Suzuki
University of Tsukuba
Tennodai 1-1-1, Tsukuba, Ibaraki 305, Japan

Hierarchical complex object is a very significant concept in CAD applications. In this paper a formal model called SCORE and its associated algebra named ARES for manipulating the hierarchial complex objects are proposed. SCORE model is based on trees that are generated by CFG. As a result, the important mechanisms such as generalization, aggregation, and recursion can be specified in a common frame. ARES is composed of four operators which serve as a primitive set of operators for constructing application oriented operations for hierarchical complex objects.

1. Introduction

The significance of the concept of grammars for formal languages has long been acknowledged in varied fields. Recently, database researchers have recognized its utility in representing complex objects [Gonnet88, Gyssens89, Jiang90]. In this paper a formal model called SCORE (short for Structured Complex Objects defined REcursively) and its associated algebra named ARES for representing the hierarchical complex objects are proposed. This model is based on *trees* that are generated by CFG (Context Free Grammar).

Semantic data models, such as ER [Chen76], FDM [Shipman81], SDM [Hammer81], Format [Hull84] and IFO [Abiteboul87] provide a rich set of design tools for representing complex interrelationships of data not present in the relational model, such as aggregation, generalization and set constructs. Logic based models, such as Datalog [Ullman88], LDM [Kuper84] and LDL [Beeri87] aim at the limited expressiveness of data manipulation languages of the relational model, i.e. they generalize the relational calculus to express recursively specified queries. Finally, relational extensions of the standard relational model, such as RT/M [Codd79] and the nested model [Makinouchi77, Kitagawa89], try to strike a balance between generalizing the data modeling and the data manipulation parts of the relational model. All these models share the property that they recognize the most fundamental characteristic of data to be its hierarchical structure. However, it is not quite clear whether they can effectively model all data applications which exhibit a hierarchical nature. Good examples are textbases [Gonnet88] and CSG data [Jiang90] as shown in Figure 1, which in addition to having a hierarchical structure, are constructed out of rules that follow a grammatical structure.

Tree structure is very popular as a data structuring technique in CAD applications. To adequately support such applications, the data model must reflect this tree structure. In SCORE, we define a hierarchical complex object as a tree using CFG. We present an algebra called ARES which can extract and manipulate data at all levels of a tree without having to navigate in the tree in order to access data that is stored at various levels in the tree.

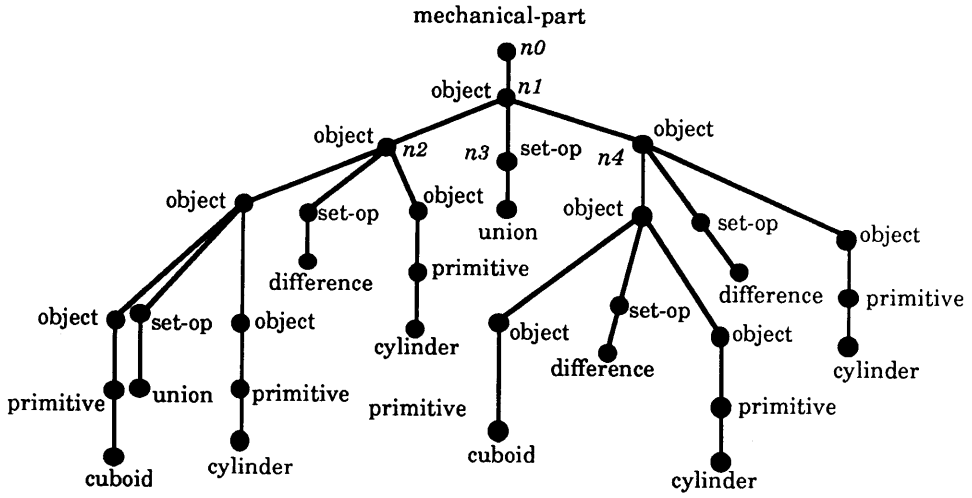

Figure 1 The tree representation of a solid object

2. Basic Terminology

Formally, a tree can be defined recursively in the following manner. Here, N* is a set of lists of elements in N. A projection π_i to the i-th element is used to extract an element in the list.

Definition 1 [Tree]

A tree T is an ordered pair (N, ρ: N -> N*), where $N=\{n_1,..., n_r\}$ is a finite set of data items called nodes and ρ is a mapping from N to N* where the following condition should be satisfied.

$$\exists!n\in N,\ \forall n'\in N,\ \exists!n_1, n_2,..., n_m\in N$$
$$\pi_{i0}(\rho(n))=n_1,$$
$$\pi_{i1}(\rho(n_1))=n_2,$$
$$....,$$
$$\pi_{im}(\rho(n_m))=n'.$$

Definition 2 [Labeled Tree]

A labeled tree L is a tree in which each of its nodes is associated with a label, that is, L=(N, ρ, ψ: N -> Label) with a mapping ψ from N to Label which is a set of labels.

Hereon, a tree is a labeled tree unless otherwise specified. A node n of a tree T=(N, ρ, ψ) is an internal node iff ρ(n)≠(); otherwise it is called a leaf node. The set of internal node is defined as internal(T)={n | n∈N, ρ(n)≠()}. Obviously, leaf(T)=N-internal(T). A tree T=(ϕ, ρ:ϕ->ϕ*, ψ:ϕ->Label) is called *an empty tree* represented as E.

Definition 3 [Subtree]

Given a tree T=(N, ρ, ψ). A subtree of T is a tree T'=(N', ρ': N' -> N'*, ψ':N' -> Label), where $N\subseteq N'$, $\psi'=\psi\,|\,N'$. Here, $\psi\,|\,N'$ is a function ψ with a domain restricted to N'. This subtree is described by subtree(T, N').

Suppose that we are given a tree T. Then rt(T) denotes the root of T. The existence and uniqueness of the root is guaranteed by the condition, if the tree T is not the empty tree, E. For a node n of T, pt(n) denotes its parent. Namely, $pt(n)=n' \Leftrightarrow \pi_i(\rho(n'))=n$. cdll(n) is the sequence of the labels of all children of n. Formally, $\forall i(\pi_i(cdll(n))=\psi(\pi_i(\rho(n))))$.

For disjoint trees $T_i=(N_i, \rho_i, \psi_i)$ (i=1,..., n), a tree T=(N, ρ, ψ) is constructed as follows.

$N=\{n_r\}\cup N_1\cup...\cup N_n$ for $n_r\notin N_i$ (i=1,..., n).

$$\rho(n)=\begin{cases}(rt(T_1),...,\ rt(T_n)) & n=n_r\\ \rho_i(n) & n\in N_i\end{cases},\qquad \psi(n)=\begin{cases}l & n=n_r\\ \psi_i(n) & n\in N_i\end{cases}$$

This tree is denoted as $<n_r, l: T_1,..., T_n>$.

If $n_1, n_2,..., n_k$ is a sequence of nodes in a tree such that n_i is the parent of n_{i+1} for i=1,..., k-1, then this sequence is called a *path* from node n_1 to node n_k with the length k-1. If there is a path from node a to node b with length larger than zero, then a is called an *ancestor* of b, and b is called a *descendant* of a. For example, in Figure 1, the ancestor of n_1 is n_0, while its descendants are n_2, n_3, and n_4. In a tree, the root is the only node with no ancestor. A set of descendents of the node a is specified by descendent(a), the parent of the node a is specified by pt(a).

3. SCORE Model

We first give a few definitions for the SCORE model and then introduce the algebra ARES. The terms *schema* and *instance* are borrowed from the relational model. In the SCORE model, the former is associated to the grammar and the latter is associated to the tree.

Definition 4 [Database Schema]

A database schema is a formal grammar g=(V, T, S, P) with V a finite set of attributes, T a finite set of constants, S a set of axioms, $S \subseteq V$, and P a finite set of production rules of the form A->s where $A \in V$, $s \in (V \cup T)^+$.

Another notation r=(A, s) is used interchangeablely with r=A->s. Namely, $P \subseteq V \times (V \cup T)^+$. Quite different from the CFG for the language theory, we cannot replace production rules A->B, B->C by A->C, because the derivation tree structure is of significance.

Definition 5 [instance]

Let g=(V, T, S, P) be a database schema. A D-tree D over g is called an instance of g iff $\psi(rt(D)) \in S$ and for each internal node n there exists a production rule $\psi(n)$->cdll(n) in P, and for each leaf node n there does not exist a production rule $\psi(n)$->s in P. Formally, $\forall n \in internal(D)$ $(\psi(n), cdll(n)) \in P$, and $\forall n \in leaf(D)$ $\psi(n) \notin \pi_1(P)$.

Definition 6 [isomorphism]

Given two instances D_1 and D_2 of some g, we say D_1 and D_2 are isomorphic (denoted by $D_1 \cong D_2$), if there exists a bijection θ: D_1 -> D_2 satisfying the following condition:

$$\forall n \in D_1(\psi_1(n)=\psi_2(\theta(n)) \wedge (\rho_1(n)=(n_1,\ldots, n_m) \Rightarrow \rho_2(\theta(n))=(\theta(n_1),\ldots, \theta(n_m)))).$$

g=(V, T, S, P)
V={mechanical-part, object, primitive, motion-op, set-op}
T={rotate, scale, union, intersection, difference, cuboid, cylinder}
S={mechanical-part}
P={<mechanical-part>-><object>
<object>-><primitive>
<object>-><object><motion-op>
<object>-><object><set-op><object>
<primitive>->cuboid
<primitive>->cylinder
<motion-op>->rotate
<motion-op>->scale
<set-op>->union
<set-op>->intersection
<set-op>->difference}

Figure 2 Example of a database scheme

R={(υ, <object>)->(α, <object>)(η, <set-op>)(β, <object>),
(α, <object>)->(γ, <primitive>),
(γ, <primitive>)->cuboid,
(η, <set-op>)->union,
(β, <object>)->(ω, primitive),
(ω, primitive)->cylinder}

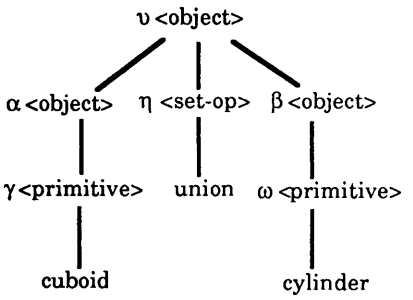

Figure 3 Example of pattern

Figure 2 shows an example of a database schema which describes the CSG data. The D-tree shown in Figure 1 represents an instance over g.

4. ARES: An Algebra for SCORE

The main motivation for the algebra is to provide a query language which can extract and manipulate data at all levels of a tree without having to navigate in the tree in order to access data that is stored at various levels in the tree. We present an algebra called ARES (short for Algebra for REcursive complex objects in SCORE) which allows to formulate queries on both schema and instance level. The operators in this algebra are all recursively defined so that each operator can be applied to subtrees at all levels thus eliminating the need for a special operator to serve as a "navigator". This is achieved by manipulating trees by means of pattern matching.

4.1 Pattern Matching

Basically, a pattern R is a set of production rules. When a part T' of a given tree T over g=(V, T, S, P) is the D-tree over (V, T, S, R), then we say the pattern R matches the tree T at rt(T'). Two extensions on the pattern specification are required for the pattern to be effectively used for specifying operations.

1. Suppose that the pattern is R={A->B, B->C}, then it is ambiguous whether B in A->B should correspond to the same node with B in B->C. In order to make it clear, we prefix tags before all attributes (even though some of them may be of trivial in some case).

2. It is sometimes necessary to represent an isomorphism among subtrees. To represent the isomorphism between a subtree with its root corresponding to λB and that with the root corresponding to ωB, we use the equivalence relation $\lambda \sim \omega$. This relation Q is a subset of Tag×Tag. Notice here that the nodes correspond to λB and ωB are distinct.

A set of tags is named Tag and tags are represented by Greek letters α, β, γ and so on. TagLabel stands for Tag×Label. A *tagged production* P' over P is a subset of TagLabel×TagLabel* which satisfies the following condition.

$$\forall r'=(\varphi_1', \ldots, \varphi_n') \in P', (\pi_2(\varphi_1'), \ldots, \pi_2(\varphi_n')) \in P.$$

Some of the tags in the tagged production have no significance. However, we assume that each label in the tagged production is given a tag for the sake of simplicity. Formally, we define the pattern as follows:

Definition 7 [pattern]
Let g=(V, T, S, P) be a database schema. An ordered pair R=(P', Q) is called *pattern over g* if the following conditions are satisfied.

Condition 1: Connected, Rooted and Loop-free

$\exists! r(\in P'), \forall r'(\in P'), \exists! r_1, r_2, \ldots, r_n \in P', \pi_{i1}(r)=\pi_1(r_1), \pi_{i2}(r_1)=\pi_1(r_2), \ldots, \pi_{in+1}(r_n)=\pi_1(r')$
Hereon, r is denoted as rt(R).

Condition 2: Tag Consistency

$\forall r, r' \in P'(\pi_1(\pi_i(r))=\pi_1(\pi_j(r')) \Rightarrow \pi_2(\pi_i(r))=\pi_2(\pi_j(r')))$.

Due to the tag consistency, we can define $att(\lambda)=\{\pi_2(\pi_i(r)) \mid \pi_1(\pi_i(r))=\lambda, r \in P'\}$ to represent the attribute tagged by λ in r. Given a pattern and an instance of some database schema, we can find out the occurrences of the pattern in the instance, namely, perform the *pattern matching*.

Definition 8 [pattern matching]

Let g=(V, T, S, P) be a database schema and let D=(N, ρ, ψ) be an instance over g. Let R=(P', Q) be a pattern over g. *R matches the tree D at node n* iff there is a mapping f: TagLabel->N which satisfies the following conditions:

1. $\psi(f((\alpha,x)))=x$,
2. $(\alpha, \beta)\in Q \Rightarrow$ subtree(T, decedent(f((α, x)))) ≅ subtree(T, decedent(f((β, x))))
3. $\alpha\neq\beta \Rightarrow f((\alpha, x))*f((\beta, x))$
4. $\forall r'=(\varphi_1', \varphi_2',..., \varphi_n')\in P' \Rightarrow \rho(f(\varphi_1'))=(f(\varphi_2'),..., f(\varphi_n'))$
5. f(rt(R))=n.

Since Q is an equivalence relation, for any tag α, $(\alpha, \alpha)\in Q$ holds. If α=β in the above condition 2, then the condition is trivially satisfied. If α≠β in the above condition 2, then the condition 4 also holds and the subtrees which are isomorphic to each other should be disjoint.

Figure 3 describes a pattern over the schema shown in Figure 2 with the *object* and *primitive* to be tagged by α, β and γ, ω, respectively. If we consider the tree in the Figure 1 as an instance of the schema shown in Figure 2, the leftmost part of the tree matches this pattern.

When using the operators to manipulate trees, all the operators have the same process of *evaluation* as follows:

Definition 9 [Evaluation]

Let D and R be an instance and pattern over some schema g respectively. σ_i stands for a tag. Let Op[R, σ_1,..., σ_n](D) be any operation Op defined below. The evaluation strategy of Op over D is defined recursively as follows:

Step 1. If rt(D) is not a leaf, then apply the same operation Op[R, σ_1,..., σ_n] on all child subtrees D_1,..., D_m of rt(D). Formally,

$$D'=\begin{cases} \langle rt(D),\ \psi(rt(D)):\ Op[\sigma_1,..., \sigma_n](D_1),..., Op[\sigma_1,..., \sigma_n](D_m)\rangle & rt(D)\in internal(T) \\ D \quad otherwise \end{cases}$$

Step 2. If R matches D' at rt(D'), perform the specified operation on D', and return the resultant tree. Otherwise return D'.

The evaluation strategy of Op over g is defined in a similar manner.

4.2 Operators in ARES

We now formally define the four operators in ARES. In the definition, s, s_i (i=1,2,...) denote arbitrary strings composed of attributes and constants.

Definition 10 [Substitution]

Let g=(V, T, S, P) be a database schema and A be an attribute (A does not have to be in V). Let R be a pattern over g and λ a tag in R. The substitution U[R, λ, A] is defined as follows:

U[R, λ, A](g)=g'=(V', T, S', P')

where V'=V∪{A}, S'=S∪{A} if att(λ)∈S, otherwise S'=S, P'=P''∪{C->s_1As$_2$ | C->$s_1$att(λ)s_2∈P''}, for P''=P∪{A->s}.

U[R, λ, A](D)=D' where if R matches the tree D at rt(D), then D'=(N, ρ, ψ') is defined as follows for the mapping f.

$$\psi'(n)=\begin{cases} A & \text{if } n=f((\lambda, x)) \\ \psi(n) & \text{otherwise} \end{cases}$$

If the match fails, then D'=D.

Let's assume a database schema g=(V, T, S, P) with V={A, B, C}, T={a, b, c}, S={A} and P={A->BC, B->A, B->b, C->a, C->c} and let D be the instance shown in Figure 4(a). Then the substitution U[{(α, A)->(λ, B), (λ, B)->(β, A)}, λ, D] yields the database schema g'=(V', T, S, P') with V'={A, B, C, D}, P'={A->DC, A->BC, D->A, B->A, D->b, B->b, C->a, C->c} and the instance D' as shown in Figure 4(b).

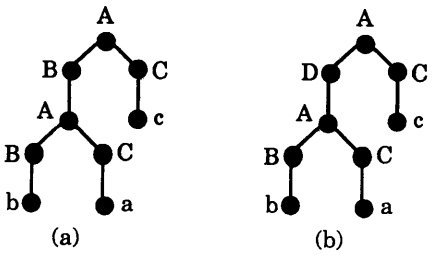

Figure 4 Example of substitution

We also need an operator that move data from one place to another. This operator named *move* is defined as follows:

Definition 11 [Move]

Let g=(V, T, S, P) be a database schema. Let R be a pattern over g and two tags α and β occur in the rule R. The move M[R, α, β] is defined as follows:

M[R, α, β](g)=g'=(V, T, S, P')

where P'=P∪{A->$s_1$att(β)s_2 | A->$s_1$att(α)s_2∈P},

M[R, α, β](D)=D'=(N', ρ', ψ')

where N'=(N-descendent(f((α, att(α)))))∪N_0, subtree(D', N_0)=(N', ρ'', ψ'')≅ subtree(D, descendent(f((β, att(β))))),

$$\rho'(n)=\begin{cases} \rho(n) & n\in N\text{-descendent}(f(\alpha, att(\alpha))) \\ (n_1,\ldots, rt(subtree(N', N_0)),\ldots, n_m) & n=pt(f(\alpha, att(\alpha))), \\ & \rho(n)=(n_1,\ldots, n_i,\ldots, n_m), \\ & n_i=f(\alpha, att(\alpha)) \\ \rho''(n) & \text{otherwise} \end{cases}$$

$$\psi'=\psi \mid N'$$

Next, we define two operators that allow the introduction of new node and the removal of existing one.

Definition 12 [Deletion]

Let g=(V, T, S, P) be a database schema. Let R be a pattern over g and α a tag in R. The deletion is defined as follows:

D[R, α](g)=g'=(V, T, S, P')

where P'=P∪{B->s_1s_2 | B->$s_1$att(α)s_2∈P}.

D[R, α](D)=D'=(N', ρ', ψ')

where N'=N-subtree(D, f(α, att(α))),

ψ'=ψ | N'

$$\rho'(n)=\begin{cases}\rho(n) & n\neq pt(f(\alpha, att(\alpha)))\\ (n_1,\ldots, n_{i-1}, n_{i+1},\ldots, n_m) & n=pt(f(\alpha, att(\alpha))),\ \rho(n)=(n_1,\ldots, n_i,\ldots, n_m),\ n_i=f(\alpha, att(\alpha))\end{cases}$$

Definition 13 [Insertion]

Let g=(V, T, S, P) be a database schema and A be an attribute that does not occur in V. Let R be a pattern over g and α a tag in R. The insertion is defined as follows:

I[R, α, A](g)=g'=(V', T, S, P')

where V'=V$\cup${A}, P'=P$\cup${att(α)->sA}.

I[R, α, A](D)=D'=(N', ρ', ψ')

where N'=N$\cup${n_A} for $n_A\notin N$,

$$\rho'(n)=\begin{cases}\rho(n) & n\neq f(\alpha, att(\alpha))\\ (n_1,\ldots, n_m, n_A) & n=f(\alpha, att(\alpha)),\ \rho(n)=(n_1,\ldots, n_m)\end{cases}$$

$$\psi'(n)=\begin{cases}\psi(n) & n\in N\\ A & n=n_A\end{cases}$$

4.3 Data Manipulation

The four operators in ARES define the algebra of SCORE. Many conceivable operations upon trees can be expressed in terms of these four operators, that is, there exist sequences of instance dependent algebra operations that returns the same result at the instance level.

As an example, we reconsider the addition of primitive node into the tree as shown in Figure 6. The transformation can be accomplished by consecutively performing the following sequence of operations written in ARES:

1. I[{(α, A)->(β, B_1)(γ, B_2)(υ, B_3)}, α, V],
2. M[{(α, A)->(β, B_1)(γ, B_2)(υ, B_3)(ω, V)}, ω, α],
3. D[{(λ, A)->(β, B_1)(γ, B_2)(υ, B_3)(ω, A), (ω, A)->(ξ, B_1)(χ, B_2)(v, B_3)(ε, V)}, ε],
4. D[{(ω, A)->(α, B_1)(γ, B_2)(υ, B_3)(β, A)}, α],
5. D[{(β, A)->(α, B_2)(γ, B_3)(ω, A)}, α],
6. D[{(β, A)->(α, B_3)(γ, A)}, α],
7. U[{(α, A)->(β, A)}, α, R],
8. I[{(α, R)->(ω, A)}, α, N].

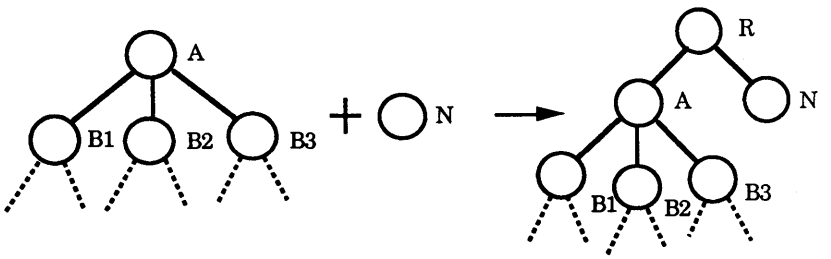

Figure 6 Example of node addition

Another example is the permutation of two subtrees from A as shown in the figure 7. This operation in our algebra can be expressed by

1. I[{(α, A)->s_1}, α, B]

2. M[{A->s_{11}(α, C)s_{12}(β, B)), β, α]
3. M[{A->s_{11}(ρ, C)s_{12}(ω, D)s_{13}C), ρ, ω]
4. M[{A->s_{11}Ds_{12}(α, D)s_{13}(γ, C)), α, γ]
5. D[{A->s_1(δ, C)), δ]

Here s_i, $s_{ij} \in (V \cup T)^*$ represent some finite strings.

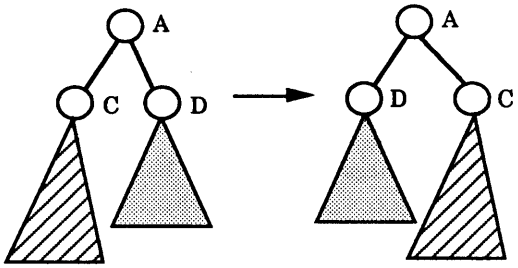

Figure 7 Example of the permutation

5. Expressive Power

Gyssens and others proposed a hierarchical data model based on the context free grammar and defined a set of operations to manipulate the hierarchical structures [Gyssens89]. However, there is one crucial difference between the Gyssens' model and SCORE. The former, in an attempt to support the concept of complex objects, imposes a keen restriction onto the definition of complex object, that is, it forces a complex object to obey a conditional CFG grammar whose rules must have different symbols in the right-hand. SCORE expresses objects using CFG without any restriction, and results in a much more natural syntax and semantics for expressing the operations on a complex object. The following theorem guarantees that the algebra of Gyssens' model can be expressed in the ARES. The proof of this theorem is given in [Jiang90].

Theorem [expressive power]
ARES has the expressive power of Gyssens's algebra [Gyssens89].

Because the expressive power of the algebra in Gyssens' model has been shown to be the same as the calculus in the model [Gyssens89], we have the following as a natural result.

Corollary
ARES has the expressive power of Gyssens's calculus.

6. Conclusion

In this paper a formal model called SCORE and its associated algebra named ARES are proposed for the purpose of representing the hierarchical complex objects existing in CAD applications. This model is based on *trees* that are generated by CFG. The operators in ARES are all recursively defined so that each operator can be applied to subtrees at all levels thus eliminating the need for a special operator to serve as a "navigator". This is achieved by manipulating trees by means of pattern matching.

ARES is characterized by tree transformation accompanied schema change. In general, the schema returned by the algebra sequence defines a larger language than the schema returned by the original. Sometimes, changing the database schema in arbitrary ways is risky and not desirable. Our general approach to the problem of changing database schema adopts the philosophy of restricting the evolution of the database schema to a controllable degree. This can

be achieved by imposing the condition that the set of production rules in the database schema to remain unchanged in all operations with an exception to the substitution operation. The substitution is correspond to the rename operation in the relational model, and does not destruct the structure of database. So if we restrict the other three operators in ARES to abide to the above rule, the evolution of database schema can be easily controlled.

References

[Abiteboul87] Abiteboul, S. and Hull, R., "IFO: A Formal Semantic Database Model", *ACM TODS.*, Vol.12, No.4, 1987, pp.525-565.

[Beeri87] Beeri, C. et al., "Sets and Negation in a Logic Database Language (LDL1)", *Proc. 6th PODS*, San Diego, 1987, pp.21-37.

[Chen76] Chen, P. P., "The Entity-Relationship Model: Towards a Unified View of Data", *ACM TODS*, Vol.1, No.1, March 1976.

[Codd79] Codd, E. F., "Extending the Relational Database Model to Capture More Meaning", *ACM TODS.*, Vol.4, No.4, Dec. 1979, pp.397-434.

[Gonnet88] Gonnet, G. H. and Tompa, F. W., "Mind Your Grammar: a New Approach to Modelling Text", Techn. Rep., Univ. Waterloo, 1988.

[Gyssens89] Gyssens and M. Paredaens, J., "A Grammar-Based Approach towards Unifying Hierarchical Data Models", *Proc. of SIGMOD*, 1989, pp.263-272.

[Hammer81] Hammer, M. and McLeod, D., "Database description with SDM: A Semantic Database Model", *ACM TODS*, Vol.6, No.3, 1981, pp.351-386.

[Hull84] Hull, R. and Yap, C., "The Format Model: A Theory of Database Organization", *Journ. ACM*, Vol.31, No.3, 1984, pp.518-537.

[Jiang90] Jiang, S. J., "Incorporating Abstract Data Types and Complex Objects in Relational Database Environments", Ph.D thesis, University of Tsukuba, Tsukuba, 1990.

[Kitagawa89] Kitagawa, H. and Kunii, T., "The Unnormalized Relational Data Model For Office Form Processor Design", in *Computer Science Workbench*, Kunii, T., Ed., Springer-Verlag, 1989.

[Kuper84] Kuper, G. M. and Vardi, M. Y., "A New Approach to Database Logic", *3rd PODS*, Waterloo, Ont., 1984, pp.124-138.

[Makinouchi77] Makinouchi, A., "A Consideration of Normal Form of Not-necessarily Normalized Relations in the Relational Data Model", in *Proc. VLDB*, 1977, pp.447-453.

[Shipman81] Shipman, D., "The Functional Data Model and the Data Language DAPLEX", *ACM TODS*, Vol.6, No.1, 1981.

[Ullman88] Ullman, J. D., "Principles of Database and Knowledge-Base Systems, Vol I", Computer Science Pr., 1988.