A Logic-Based Approach for Managing Structured Document Data

Norio Katayama Atsuhiro Takasu Jun Adachi

Graduate School of Engineering, University of Tokyo

Research and Development Department, National Center for Science Information Systems

{katayama, takasu, ja}@nacsis.ac.jp

Abstract

This paper presents a logic-based approach for managing structured document data such as SGML and TEX documents. The difficulty in managing such document data is the variety of document structures. We have proposed an S-object model as the data model for this purpose, which integrates the concepts of hypertext and deductive databases. In this paper, we formalize the query and consistency management mechanism of the S-object model. It uses an extended version of a function-free definite program. The extension is made by the concept of a functional predicate which provides a general way to introduce arbitrary computation into definite programs. It increases the expressive power without losing the semantics of a function-free definite program.

1 Introduction

This paper presents a logic-based approach for managing structured document data such as SGML (ISO8879) and TEX documents. We have proposed an "S-object model (semantic object model)"[2] as a data model for this approach. The contribution of this paper is to formalize the query and consistency management mechanism of the S-object model. It is based on predicate logic and uses an extended version of a function-free definite program which is used in deductive databases. The extension is made by the concept of a "functional predicate" which is the key of this paper.

Section 2 discusses background of our approach, and Section 3 outlines the S-object model. Section 4 presents the functional predicate and its evaluation algorithm. Section 5 discusses characteristics of functional predicates and Section 6 illustrates the logic-based management of structured document data. Conclusions are contained in Section 7.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ADTI copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of ADTI Organizing Committee. To copy otherwise, or to republish, requires a fee and/or special permission.

International Symposium on Advanced Database Technologies and Their Integration (ADTI'94)
Nara, Japan, October 1994

2 Background

The motivation of this study is to develop a flexible model for storing documents in the form of SGML or TEX. These documents are called structured documents because they contain the information of document structures, such as titles, abstracts, chapters, sections, and figures. But they are quite different from standardized document structures such as ODA (ISO8613). ODA documents have well-defined structures and semantics. Thus, they are said to be appropriate for being stored in object-oriented databases. The situation differs for SGML or TEX documents because they have the variety of document structures. It is impractical to define a stable database schema for SGML or TEX documents; thus making it very difficult to apply conventional approaches such as relational or object-oriented databases.

One promising approach is a hypertext database[4, 5], which integrates hypertexts and database techniques. It can store hypertexts effectively and sometimes has a virtual link mechanism. Structured documents can be regarded as hypertexts since they have a network structure among components in a logical sense. Therefore, structured documents can be stored into a hypertext database.

However, we concluded that hypertexts do not have enough power to represent SGML or TEX documents because such documents contain structured semantics which is more complex than the one hypertexts have. When such documents are modeled only by nodes and links, their semantics of structures are not preserved. Then, we selected a logic-based approach as the basis of the query and consistency management. By using

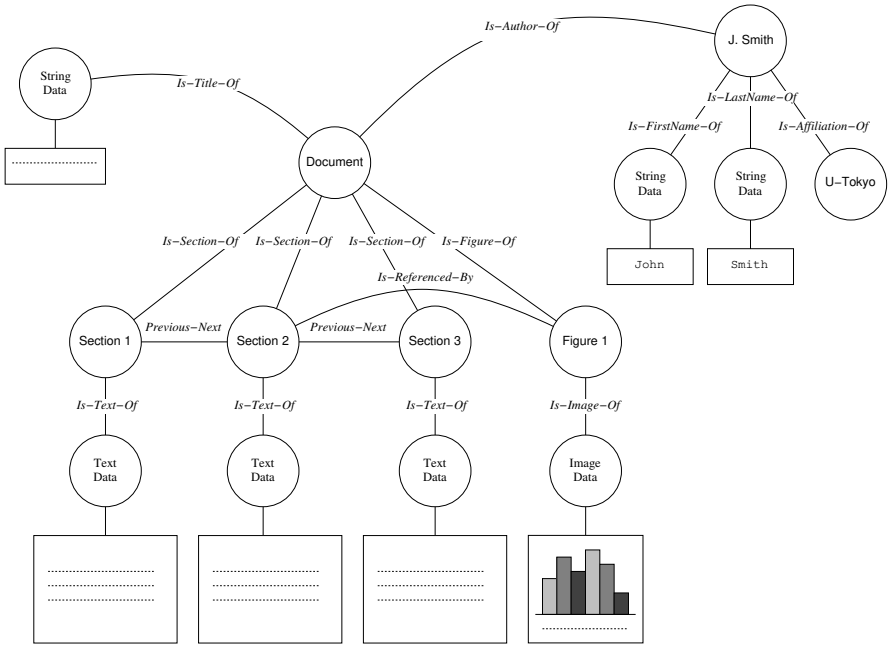


Figure 1: An Example of a Network Structure Composed by S-objects and S-links

this approach, queries and integrity constraints are described with predicate logic; thereby making it possible to describe recursive queries, derivation rules, consistency rules, etc.

We have developed an S-object model as the data model for this approach, which integrates the concepts of hypertexts and logic-based management mechanisms. It specifies how to represent and manage the semantics of data. It uses a network structure as a semantic representation, and first-order predicate logic to manage the representation.

The predicate logic used in the S-object model is close to the one used in deductive databases. The S-object model is also based on a function-free definite program with stratified negation. That class of the predicate logic is well studied in the literature on deductive databases. However, we found that the class is too simple to be applied to practical use, because it has no mechanism to introduce arithmetic operators in a general fashion. For example, although Datalog[6, 1] has binary comparison operators such as $<$ (less), $>$ (greater), $=$ (equal), etc., the way to introduce other operators is not provided. In practical applications, more operators are needed such as arithmetic operators, character string operators, type conversion operators, etc.

One solution is to introduce functions as in Prolog, but the introduction of functions makes a language too general to describe queries and consistency rules. Therefore, we investigated another solution, and extended the conventional function-free definite program with the concept of a functional predicate. The details of a functional predicate will be described in Section 4.

3 S-Object Model

The S-object model is composed of five components: S-objects, S-relations, S-links, S-constraints, and P-data. S-objects and S-links compose a network structure like hypertexts or semantic networks. An S-relations is a class of S-links and identifies the semantics of S-links. An S-constraint is an integrity constraint on the network structure and expressed by a function-free definite program. A P-data is a physical data, or a binary data, such as number, character string, text, and image. An example of a network structure composed by S-objects and S-links is shown in Figure 1.

3.1 S-Objects

The role of an *S-object* is to uniquely represent semantic entities in a database. Semantic entities can be classified as follows:

- Real World Entities (e.g., individual documents, persons, and pages)
- Conceptual Entities (e.g., sets of documents, persons, etc. and concepts of science, linguistics etc.)
- Physical Data Entities (e.g., number, character string, text, and image)

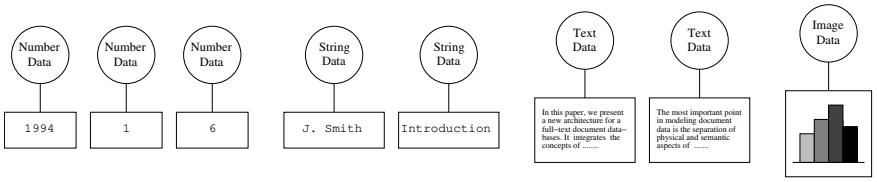


Figure 2: Pairs of P-data and S-object

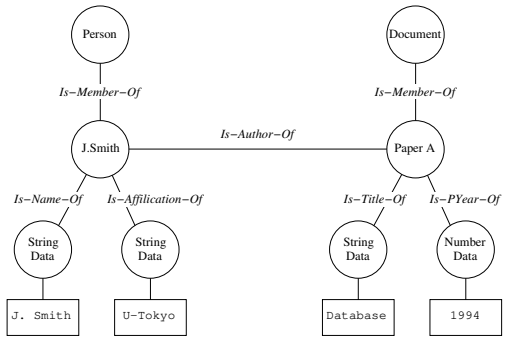


Figure 3: S-link Between an Author and his Document

Since an S-object is just a node, it should not be confused with objects in an object-oriented database.

3.2 P-Data

A *P-data* is a physical data, which is encapsulated with the abstract data types. Each P-data is associated with an S-object which takes the semantic role for the P-data. The mapping between a P-data and its corresponding S-object is one-to-one (Figure 2). With this mechanism, physical data can be treated as an encapsulated component in the network structure composed by S-objects and S-links.

3.3 S-Links and S-Relations

An *S-link* is a hyperlink among S-objects. Each S-link represents a semantic relationship among them. For example, an S-link in the center of Figure 3 represents a relationship which means "J. Smith is the author of Paper A". S-links are expressed in the form of predicates. For example, the S-link above is expressed with a predicate as follows:

```
Is-Author-Of [ Author = ⟨J. Smith⟩,
               Document = ⟨Paper A⟩ ]
```

where `Is-Author-Of` is a predicate name which identifies the semantics of the relationship. Inside the brackets is a list of terms. Each term has a label like `Author` and `Document` to increase the intelligibility. ⟨`J. Smith`⟩ and ⟨`Paper A`⟩ represents S-objects whose names are "J. Smith" and "Paper A," respectively.

An *S-relation* is a class of such S-links that represent the same semantics of the relationship, i.e., a class of such S-links that are expressed with the same predicate.

3.4 S-Constraints

S-constraints are defined for two purposes: one is to preserve the consistency of the network structure, and the other is to define derivation rules for S-links. The S-constraints are defined as function-free definite program with stratified negation (examples are shown in Figure 4):

$$\begin{array}{ll} \text{Contradiction:} & \leftarrow L_1, \ldots, L_n \\ \text{Derivation:} & A \leftarrow L_1, \ldots, L_n \end{array}$$

where $L_1, \ldots, L_n$ are literals and A is an atom. Literals and atoms are function-free, i.e., they include no function in terms. This definition is well studied in the deductive database literature[3]. However, the S-constraint uses an extended version of a function-free definite program. The extension is made by the concept of a functional predicate which provides a general way to introduce arbitrary computation into definite programs.

4 Functional Predicates

S-constraints of the S-object model are defined as a function-free definite program with stratified negation. This class of the predicate logic is well studied in the literature on deductive databases. However, we found that the class is too simple to be applied to practical use, because it has no mechanism to introduce arithmetic operators in a general fashion. In practical applications, many operators are needed such as arithmetic operators, character string operators, type conversion operators, etc. In addition, it is necessary to introduce user-defined operators for the extendibility of a database. Therefore, we extended the conventional function-free definite program with the concept of a functional predicate. It provides a general way to introduce arbitrary computation into definite programs and increases the expressive power without losing the semantics of a function-free definite program.

4.1 Definition of Functional Predicates

Before defining a functional predicate, we need to define an atom evaluation as follows:

Contradiction: The S-object in the `Author` term of `Is-Author-Of` must be a person.

```
← Is-Author-Of [ Author = X, Document = Y ] ∧
  ¬ Is-Member-Of [ Set = ⟨Person⟩, Member = X ].
```

Derivation: `Is-GrandFather-Of` is derived from `Is-Father-Of`.

```
Is-GrandFather-Of [ GrandFather = X, GrandChild = Z ]
        ← Is-Father-Of [ Father = X, Child = Y ] ∧
          Is-Father-Of [ Father = Y, Child = Z ].
```

Figure 4: S-Constraint Examples

Definition 1 An *atom evaluation* is a process to determine values of all the variables for a given atom. The result of an atom evaluation is a set of ground atoms which are instances of the given atom.

An atom evaluation for the following atom is a process to determine the values of `X` and `Y`:

```
Is-Father-Of [ Father = X, Child = Y ]
```

Of course, terms of an atom can be a constant as follows:

```
Is-Father-Of [ Father = ⟨J. Smith⟩, Child = X ]
```

where an atom evaluation for the above atom is to determine the values of the variable `X`. When all the terms of an atom are bound to constants, i.e, when an atom is a ground atom, the result of the atom evaluation is either empty or the atom itself. In conventional deductive databases, atom evaluations are executed in the following ways:

(1) Search ground atoms in the database

(2) Derive ground atoms from the rules

(3) Invoke some computation for the built-in functions (e.g., $<$, $>$, $=$)

The purpose of introducing a functional predicate is to extend this atom evaluation mechanism.

Definition 2 A *functional predicate* is a function-free predicate which invokes any computation to execute an atom evaluation. The computation may be prepared by the system or provided by users.

Obviously, built-in functions of deductive databases are instances of a functional predicate. In addition, predicates which search or derive ground atoms belong to a functional predicate because searching or deriving can be regarded as a kind of computation. Hence, Definition 2 covers all predicates which are used in conventional deductive databases.

The most important point of Definition 2 is that any computation can be invoked by a functional predicate. The following example illustrates this:

```
Is-Substring-Of [ Part = "BC", Whole = "ABCDEF" ]
  Is-Sum-Of [ Sum = X, Value1 = 3, Value2 = 7 ]
```

`Is-Substring-Of` predicate invokes computation which checks if the value of `Part` is a portion of the value of `Whole`, while `Is-Sum-Of` predicate invokes computation which calculates the summation of the values of `Value1` and `Value2`. As you may notice, these predicates cannot be evaluated when the values of all terms are variables. For example, the following atoms cannot be evaluated because the result of the evaluation is infinite.

```
    Is-Substring-Of [ Part = X, Whole = Y ]
Is-Sum-Of [ Sum = X, Value1 = Y, Value2 = Z ]
```

This property of functional predicates is formalized as follows:

Definition 3 A functional predicate is classified into either a *finite predicate* or an *infinite predicate*. A finite predicate is a predicate which can be evaluated even if no terms are bound to constants, while an infinite predicate is a predicate which cannot be evaluated when no terms are bound to constants.

4.2 Semantics of Functional Predicates

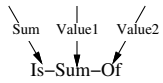
The semantics of functional predicates is quite simple. In the terms of the S-object model, a ground atom corresponds to an S-link, e.g., the following ground atom corresponds to an S-link between ⟨`J. Smith`⟩ and ⟨`Paper A`⟩.

```
Is-Author-Of [ Person = ⟨J. Smith⟩,
               Document = ⟨Paper A⟩ ]
```

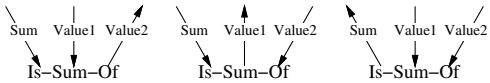
Similarly, the following ground atom corresponds to an S-link among `10`, `3`, and `7`.

```
Is-Sum-Of [ Sum = 10, Value1 = 3, Value2 = 7 ]
```

Therefore, an atom evaluation of functional predicates is a process to determine S-links for a given atom. This means that a functional predicate is a source of S-links, and a finite predicate is a predicate which stores a finite number of S-links, while an infinite predicate is a predicate which stores a infinite number of S-links. According to this standpoint, an atom evaluation for a functional predicate is a process to determine S-links in a network structure composed by S-objects and S-links.



(a) Three Input Terms



(b) Two Input Terms and One Output Term

Figure 5: The Evaluation Mode Diagrams for the Is-Sum-Of Predicate

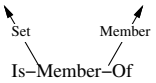


Figure 6: A Finite Predicate

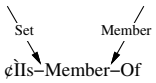


Figure 7: A Negated Predicate

4.3 Functional Predicate Evaluation Problem

Let us consider the `Is-Sum-Of` predicate further.

```
Is-Sum-Of [ Sum = X, Value1 = Y, Value2 = Z ]
```

This predicate can be evaluated by two ways: (1) Binding three terms to constants and checking if they satisfy the relationship, and (2) Binding two of three terms to constants and calculating the value of the remaining one. However, if only one term is bound, then the result of the evaluation will be infinite. Thus, as for a functional predicate, the way of an atom evaluation is limited. To clarify this property, we define an evaluation mode as follows:

Definition 4 An *evaluation mode* of a functional predicate is the way to bind terms on an atom evaluation, i.e., it specifies which terms are bound before the evaluation and which terms are bound by the computation of the predicate. The terms which are bound before an evaluation are called *input terms* and the terms which are bound by the computation of the predicate are called *output terms*.

As for the `Is-Sum-Of` predicate, an instance of an evaluation mode is the way of binding `Value1` and `Value2` before an atom evaluation and binding `Sum` after the evaluation by calculating the summation of `Value1` and `Value2`. In this case, `Value1` and `Value2` are input terms and `Sum` is an output term.

Because the evaluation modes of a functional predicate are limited, the way to evaluate a conjunction of literals is also restricted. Some conjunctions may not be evaluated. Even if possible, the evaluation order of literals must be determined. For example, the following conjunction cannot be evaluated:

```
Is-Sum-Of [ Sum = Z, Value1 = X, Value2 = Y ],
Is-Sum-Of [ Sum = X, Value1 = 2, Value2 = 3 ],
```

The following example can be evaluated, but the right literal must be evaluated first.

```
Is-Sum-Of [ Sum = 10, Value1 = X, Value2 = Y ],
Is-Sum-Of [ Sum = X, Value1 = 2, Value2 = 3 ],
```

Now we define a functional predicate evaluation problem as follows:

Definition 5 A *functional predicate evaluation problem* is a problem to determine whether a conjunction of functional predicates can be evaluated or not and to determine the evaluation order of literals.

4.4 Evaluation Mode Diagrams of Functional Predicates

To solve the functional predicate evaluation problem, we devised an *evaluation mode diagram* as follows:

Definition 6 An *evaluation mode diagram* is a diagram which represents an evaluation modes of a functional predicate, where the direction of an arrow indicates whether a term is an input or an output.

The evaluation mode diagram for the `Is-Sum-Of` predicate is shown in Figure 5. For a finite predicate, because all terms can be bound by the predicate, all arrows are pointing outward (Figure 6). On the other hand, for a negated predicate, all arrows are pointing inward (Figure 7) because the existence of an S-link cannot be determined unless the values of all terms are provided.

4.5 Selection of Evaluation Modes

To evaluate a conjunction of literals, we must select an appropriate evaluation mode for each literal. The selection algorithm is as follows:

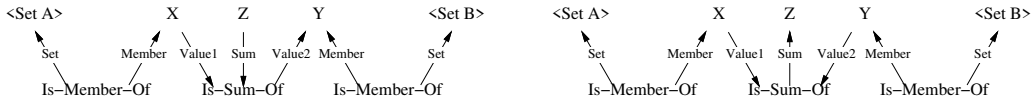
(1) Repeat the steps (2) – (4) for all possible combinations of the evaluation modes of the predicates used in the conjunction.

(2) Make evaluation mode diagrams for each evaluation mode and compose a graph by connecting the diagrams through the variables.

(3) Traverse along arrows starting from the constants and the finite predicates to determine which variables can be bound.

(4) If all variables can be bound, the conjunction can be evaluated by using the combination of the evaluation modes which is selected in the step (1).

For example, the graphs shown in Figure 8 were obtained for the following conjunction:



(a) Not Providing An Evaluation Procedure (b) Providing An Evaluation Procedure

Figure 8: Determination of an Evaluation Procedure

Table 1: Internal State of an Agent

Assignment Table	
Variable	Value
X	7
Y	–
Z	–

Evaluation History Table	
Literal	Evaluated or Not
Is-Member-Of [Set = ⟨Set A⟩, Member = X]	Yes
Is-Member-Of [Set = ⟨Set B⟩, Member = Y]	No
Is-Sum-Of [Sum = Z, Value1 = X, Value2 = Y]	No

Table 2: Literal Cache

S-link Cache			
S-link No.	Label_1	...	Label_n
1	$value_{11}$	...	$value_{1n}$
⋮		⋮	
m	$value_{m1}$	...	$value_{mn}$

Agent List	
Agent	S-link Index
$internal_state_1$	$index_1$
⋮	
$internal_state_m$	$index_m$

```
Is-Member-Of [ Set = ⟨Set A⟩, Member = X ] ∧
Is-Member-Of [ Set = ⟨Set B⟩, Member = Y ] ∧
Is-Sum-Of [ Sum = Z, Value1 = X, Value2 = Y ]
```

In Figure 8(a), the variable Z cannot be bound, while in Figure 8(b), all the variables can be bound; thereby giving a valid combination of the evaluation modes.

4.6 Evaluation Algorithm of Conjunctions of Functional Predicates

Once the evaluation modes for each literal is selected, the evaluation order of literals must be determined before evaluating each literal. It is possible to develop an algorithm which only determines the evaluation order. However, in order to implement a prototype system, we developed an algorithm which determines the evaluation order as well as executes atom evaluations. The basic idea of the algorithm is a graph traverse algorithm. Because the evaluation of functional predicates corresponds to the determination of S-links in a network structure composed by S-objects and S-links, the evaluation can be executed as a graph traverse algorithm. We developed the algorithm based on multiple agent model. Each agent finds the valid S-links by evaluating each literal. The internal state of an agent is depicted in Table 1. An assignment table contains information about binding between variables and values, while evaluation history table records which literal has been evaluated.

To enable agents to receive all applicable S-links, a literal cache is prepared for each literal. A literal cache consists of a link cache and an agent list as shown in Table 2. The S-link cache stores S-links determined by the computation. The agent list stores internal states of all the agents that need to receive the S-links of the literal. Each agent in the agent list is associated with the S-link index which points to the most recently received S-link in the S-link cache.

The execution flow of agents and literal caches are depicted in Figure 9 and 10 respectively.

5 Characteristics of Functional Predicates

The characteristics of functional predicates is as follows:

- **Incorporation of Arbitrary Computation into the Function-Free Definite Program**
  The concept of a functional predicate provides a general mechanism to incorporate arbitrary computation into an atom evaluation; thereby making it possible to incorporate user-defined computation into function-free definite program. This is a great advantage because it increases the extendibility of a database management system.
- **Preservation of the Semantics of the Conventional Function-Free Definite Program**
  Writing a derivation rule or a consistency rule with functional predicates does not destroy the semantics of the conventional function-free definite program. Although more study is needed to define the semantics of a functional predicate formally, rules written with functional predicates can be regarded as a function-free definite program. The only difference is that the rules written with functional predicates have the functional predicate evaluation

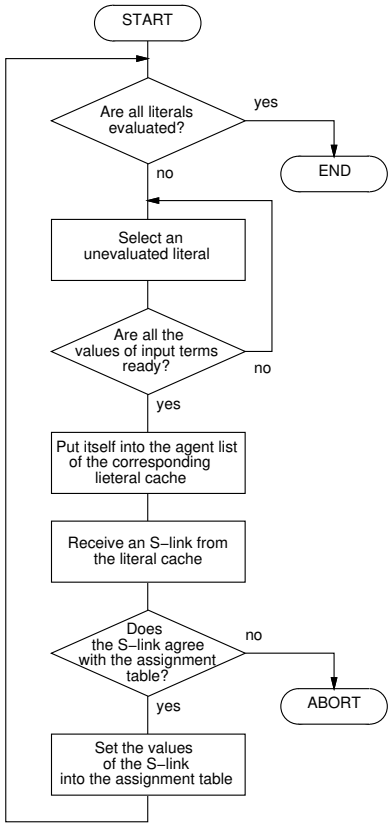


Figure 9: The Execution Flow of an Agent

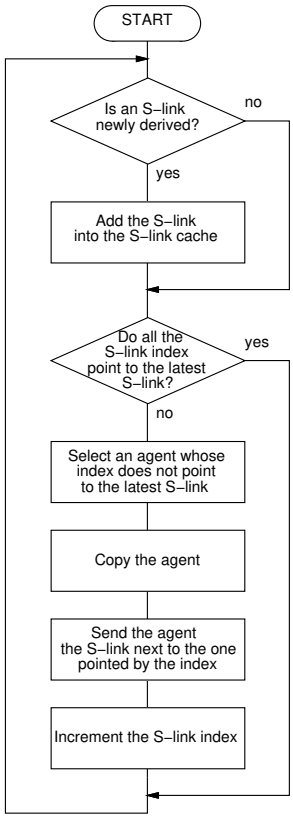


Figure 10: The Execution Flow of a Literal Cache

problem described earlier.

– **Simplicity of the Semantics**
  As stated earlier, the evaluation of functional predicates corresponds to the determination of S-links in a network composed by S-objects and S-links. Therefore, the semantics of functional predicates is simple and easy to be understood. The logic programming such as Prolog is another way to incorporate computation into definite programs, but its unification mechanism is quite complicated. We believe that the simplicity of functional predicates is advantageous in database applications.

6 Logic-Based Approach for Managing Structured Document Data

6.1 Representation of Document Structures

The main issue in managing structured document data is how to represent logical and physical document structures. A logical document structure means the structure of logical components such as title, abstract, sections, and figures, while a physical document structure means the structure of pages, i.e., page layouts. The logical and physical structures of a structured document can be obtained by using TEX or SGML processing programs. In the S-object model, components are modeled as S-objects while relationships among components are modeled as S-relations which are expressed with functional predicates. For example, the following predicates are defined to represent logical document structures:

```
Is-Title-Of [ Document = D, Title = T ]
Is-Abstract-Of [ Document = D, Abstract = A ]
Is-Section-Of [ Document = D, Section = S ]
Is-SectionNo-Of [ Section = S, Number = N ]
Is-Figure-Of [ Document = D, Figure = F ]
Is-FigureNo-Of [ Figure = F, Number = N ]
Is-Page-Of [ Document = D, Page = P ]
Is-PageNo-Of [ Page = P, Number = N ]
```

Components such as titles, abstracts, and sections have corresponding text data, while components such as figures and pages have corresponding image data. The association between components and text or image data are also represented with predicates as follows:

```
Is-Text-Of [ Component = C, Text = T ]
Is-Image-Of [ Component = C, Image = I ]
```

If all the text or image data that corresponds to components are stored within a database, the associations between components and data can be stored one by one in the form of a ground atom. However, this is impractical because an enormous amount of stor-

age would be needed. The practical way is to produce text or image data on demand by using TEX or SGML processing programs. The concept of functional predicates support this method because it provides a mechanism to implement such predicates that invoke programs internally. For example, we can define `Is-Text-Of` as a predicate which invokes a program taking an input from the `Component` term and providing an output into the `Text` term. Thus, the mechanism of functional predicates provides a way to replace an enormous amount of storage into the invocation of programs.

6.2 Derivation Rules

Derivation rules are advantageous to express the intentional relationship between predicates. Document structures are so complex that many kinds of predicates should be defined. They are usually interrelated and the interrelation can be easily expressed with predicate logic. For example, `Is-Component-Of` predicate which represents relationship between a document and its components can be defined as follows:

```
Is-Component-Of [ Document = D, Component = T ]
 ← Is-Title-Of [ Document = D, Title = T ].
Is-Component-Of [ Document = D, Component = A ]
 ← Is-Abstract-Of [ Document = D,
                    Abstract = A ].
Is-Component-Of [ Document = D, Component = S ]
 ← Is-Section-Of [ Document = D, Section = S ].
```

By defining the above derivation rule, a query can be written concisely which retrieves all components of ⟨`aDocument`⟩ that include a string "`database`" in the text as follows:

```
Result [ Component = X ]
 ← Is-Component-Of [ Document = ⟨aDocument⟩,
                     Component = X ],
   Is-Text-Of [ Component = X, Text = T ],
   Is-Substring-Of [ Whole = T,
                     Part = "database" ]
```

The following is an example of a recursive definition. It defines `Is-Followed-By` predicate which represents relationship between a component and the ones which follow it. The predicate can be derived by applying `Previous-Next` predicate recursively which represents relationship between adjacent components.

```
Is-Followed-By [ Component = X, Following = Z ]
 ← Is-Followed-By [ Component = X,
                    Following = Y ],
   Previous-Next [ Previous = Y, Next = Z ]
```

6.3 Advantages of Logic-Based Approach

- Simplicity of the representation
  Document structures are represented with simple network structure composed of S-objects and S-links.
- Expressive power of predicate logic
  S-links are classified into S-relations and expressed with predicates; thereby interrelationships between predicates being easily expressed in the form of derivation rules.
- Simplicity of the semantics
  The concept of a functional predicate preserves the semantics of a function-free definite program even if arithmetic or character string operators are introduced. The semantics of a function-free definite program is simple and easy to be understood.
- Incorporation of arbitrary computation
  The functional predicate mechanism provides a way to implement such predicates that invoke programs internally; thereby making it possible to process TEX or SGML data on demand.

7 Conclusions

A functional predicate and its evaluation algorithm is presented. It enhances the expressive power of the conventional function-free definite program. We are developing a prototype database system basing on our logic-based approach. The evaluation mechanism presented in this paper has been already implemented. Now we are working to implement DBMS facilities to the prototype system and to make it a full-fledged database management system. Final goal is to apply the logic-based approach to practical structured document data and evaluate the effectiveness of the approach.

References

[1] Elmasri, E. and Navathe, S., B., *"Fundamentals of Database Systems (Second Edition),"* The Benjamin/Cummings (1994).

[2] Katayama, N., Takasu, A., and Adachi, J., "A Database with an Explicit Semantic Representation," *Proc. of the 7th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, (1994), 323–332.

[3] Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers (1988).

[4] Tompa, F., W., "A Data Model for Flexible Hypertext Database Systems," *ACM Trans. Inf. Syst.*, **7**, 1 (Jan. 1989), 85–100.

[5] Tanaka, K., Nishikawa, N., Hirayama, S., and Nanda, K., "Query Pairs As Hypertext Links," *Proc. of the 7th Int. Conf. on Data Engineering*, IEEE (Apr. 1991), 456–463.

[6] Ullman, J., *Principles of Database and Knowledge-base Systems*, Computer Science Press, Vol. I/II (1988).