Chapter 4
A Formal Model of ODS Schema

4.1 Preamble.

An ODS schema is defined with the help of accumulated expression, formulae and rules. The quires are also expressed with the help of accumulate expressions and formulae. Therefore, semantics of an ODS schema greatly depends on the model theory of accumulated expression calculus. Moreover, an ODS schema involves recursive specification. Recursion arises with recursive rules as well as with recursive message definition. In order to uniquely define the semantics of such a schema, the structure of models plays an important role. It had been shown in this chapter that for a restricted class of schema, the semantics can be uniquely specified. The approach used here is the least fixed-point approach of specification of semantics. In the following discussion the formulae is excluded, which involve negation. Unless explicitly mentioned a formula is assumed to be free of negation.

In subsequent sections, the developed tools are used to describe the semantics for query processing and to describe the utility of different features of the algebra.

The Accumulated Arrows introduced by Ceri, Ghandeharizadeh, Interbase DDL, were used for dealing with functions that produce more than one output and the functions in programming language modeled closely. In programming languages (except for FP Functional programming languages), it often required routines with more than one output parameters. The classical definition of functions does not allow such usage of functions; therefore, the effect is normally achieved by referential parameters. Accumulated arrows permit usage of functions, which are about to return multiple values with multiple arguments. Thus, accumulated arrows are modeled with set-value or multi-functions to design a formal model of ODS schema.
4.2 Formal Model of ODS.

The use of multifunction helps to generalize the algebra to suit the demand of the database community, where an associative query is often expected to return a set of answerer that satisfies the query predicate. The algebra is based on sorts or types. A sort is a name given to a class of elements, which may possess certain properties. In the present model sorts are used to model classes in the context of object oriented databases. Each sort is associated with a domain of values or entities e.g. to have a sort of user of the library associated with the domain of users, a sort vendor may be associated with the domain of all vendors etc. In the following situation in a library yearly 5000 books are being supplied by 30 vendors. To keep records of the vendor who has received what amount of order and how much order vendor has executed by representing all the in a tabular form using a set of tables with row and columns. The tables are called as relations in this context.

<table>
<thead>
<tr>
<th>Order_no</th>
<th>Date</th>
<th>Code</th>
<th>Vendor</th>
<th>Price</th>
<th>Qty-ordered</th>
<th>Qty-Supplied</th>
</tr>
</thead>
<tbody>
<tr>
<td>50507</td>
<td>05/08/05</td>
<td>DIR12</td>
<td>M/S Academia</td>
<td>700.00</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>50508</td>
<td>05/08/05</td>
<td>DIR13</td>
<td>Progressive Book</td>
<td>14492.00</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>50509</td>
<td>05/08/05</td>
<td>DIR14</td>
<td>Joyti, the book...</td>
<td>9013.00</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>50510</td>
<td>05/08/05</td>
<td>DIR15</td>
<td>Shankar Book</td>
<td>8478.00</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.1 Order in format of Table

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR12</td>
<td>M/S Academia</td>
</tr>
<tr>
<td>DIR13</td>
<td>Progressive Book</td>
</tr>
<tr>
<td>DIR14</td>
<td>Joyti, the book...</td>
</tr>
<tr>
<td>DIR15</td>
<td>Shankar Book</td>
</tr>
</tbody>
</table>

Table 4.2 Vendor details.

These rows establish a relationship between Vendor code and the order. Thus data in the form of a set of tables can be organized. Relations represent facts describing a set of real world entities. In a relation, one entity per row and one attribute per column are represented. The table name and the column names are used to help in interpreting the meaning of the values in each row of the relation.
4.2.1 Different Features of a Relation.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Attribute is the name of a column in a relation.</th>
<th>No two attributes in a relation can have same name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Domain defines the set of all possible values that an attribute takes on</td>
<td>It is possible that several attributes in a relation can have same domain/Domain is a set of atomic values</td>
</tr>
<tr>
<td>Null Value</td>
<td>Null value indicate that the value for corresponding attribute is either not available or not applicable</td>
<td>Zero or blank as value can not be put. For such case special value in the domain called null value is identify.</td>
</tr>
</tbody>
</table>

Table 4.3- Relation.

In relation based languages, what the result is specified by the relationship that is supposed to whole. This is contrary to the concept of relational algebra where how to obtain the result is specified. It is known that in relational algebra (i.e., in the algebra, which is relationally complete), a given language L considered to be completed if it includes equivalent expressions of each of the six algebraic operations (i.e. parenthesis (), exponent^, addition +, Multiplication *, Subtract – and division /). Any calculus expression can be reduced to an algebraic equivalent to infer that the algebra is at least as powerful as the calculus. The relational database can be viewed as a collection of tuples or a collection of domains. Therefore, relational calculus languages can be categories into two groups: tuple relational calculus and domain relational calculus. Tuple relational calculus is a declarative language, which specifies the tuple variables.

<table>
<thead>
<tr>
<th>UName</th>
<th>UCode</th>
<th>Uaddress</th>
<th>Regdate</th>
<th>expdate</th>
<th>Ucat</th>
</tr>
</thead>
<tbody>
<tr>
<td>D Das</td>
<td>05EE1002</td>
<td>E&amp;ECE</td>
<td>10/12/2005</td>
<td>10/12/2008</td>
<td>BTech</td>
</tr>
</tbody>
</table>

For the query (find the names of all students of Department library from the relation STUDENT), to express the same query in tuples calculus is (find the set of tuples of
attribute \( t \) having attribute as 'Name' such that a given predicate formula \( p \) holds true for this tuples
\[
\{ t[Name] \in \text{STUDENT} \land t[\text{DEPARTMENT}]=\text{LIBRARY} \}
\]
\( t \) is a variable that stands for an arbitrary tuple
\( t[Name] \) is the value of attribute Name in tuple \( t \).

The relation from which \( t \) comes is defined by \( t \in \text{STUDENT} \) this is a part of qualifying statement \( t[(Uaddress)]=E&ECE \) is true if the value of the attribute \( E&ECE \) in \( t \) should be equal to \( E&ECE \).

Many effort has be explored to develop declarative language for ODS like sql extensions, the full calculus and deductive program. A declarative language is a formalisms for describing the desired result rather than how to compute. These language are very close to FP – a scheme for Functional programming introduce by BACKUS\(^2\). The function of FP include compassion tuple constructor and selectors; conditional: \( p \) then \( f \) else \( g \), filter pump(insert), select is a special in FP obtain by using function of the form if predicate then \( I \) else null. Where \( I \) is the identity function. A typical query has the form
for SQL
\[
\text{select target list from rangolist where equallist}
\]
In FP Style such a query is composition of a select filter and a project filter(each possibly containing nested filter): \( p(\text{target-list}) p(\text{target-list})(\sigma(\text{qual-list})(\text{range-list})) \).

To illustrate the idea, this query is presented
For each user how many book already had been reserved .
Webdb-type=U_Name:user[U-Code,U_name...............Exp_date]
Sql: Select U_code (Select Call_No ........... where Reserve_date ≠ zero)
FP= \( p(u\_code, P(U\_code)book\_reserve) \)
Lacroix and Pirote first proposed the domain relational calculus in 1977 which later implemented by Forgy and McDermott. The fundamental difference between a tuple relational language and a domain variable in the latter. A domain variable ranges over the values in a domain and specifies a component of a tuple. In other words, the range of a domain variable consists of the domains over which the relation is defined. Each query is an expression of the form

\[ \{ \langle x_1, x_2, x_3, \ldots, x_n \rangle \mid P(x_1, x_2, x_3, \ldots, x_n) \} \]

Where \( x_1, x_2, x_3, \ldots, x_n \) represent domain variables and \( P \) represents a formula similar to that tuple calculus. The definition of \( P \) is exactly same as the of tuple calculus.

Query- 1 Generate a report and display for all the book having the first author like "Singh"

This is the expressed as

```sql
create or replace procedure info(val1 in varchar2)
as
  ano varchar2(10);
  tt varchar2(151);
  au varchar2(50);
  yr number(4);
  cursor getinfo_cur is
    select acc_no, title, mail_entry, year
    from bk
    where first_authour like '%||upper(val1)||'%' 
    or first_authour like '%||val1||'%' 
    or title like '%||upper(val1)||'%' or title like '%||val1||'%' 
    or acc_no like '%||upper(val1)||'%' or acc_no like '%||val1||'%';
begin
begin
```
**dbms_output.put_line('FINDING THE REQUIRED Author.........');**

```sql
open getinfo_cur;
fetch getinfo_cur into ano,tt,au,yr;
while getinfo_cur%found
  loop
    dbms_output.put_line(ano);
    dbms_output.put_line(tt);
    dbms_output.put_line(au);
    dbms_output.put_line(yr);
    fetch getinfo_cur into ano,tt,au,yr;
  end loop;
  close info_cur;
end;
```

The expression allocate some space for (ano,tt,au,yr ) and then searching the similarity or likelyness for the database and store the information into the getinfo_cur memories location and ready to display the information on the client browser.

### 4.2.3 The Calculus of Accumulated Expressions (AEC).

The calculus of accumulated expressions represents the language of terms and expressions developed with the help of type variables and accumulated operation symbols. This language is used in describing the ODS model of database. In the following sections the syntactic notions and the associated interpretations are described. The interpretations are expressed with the help of notation evaluating under an interpretation. An interpretations, as defined here, provides meaning of the fundamental items in an AEC. Meaning of other items, viz. expression, formula and rule are inferred from this fundamental items and the constructs of the calculus. The meaning of expressions and predicates in an interpretation is given by their evaluations. There are two parts of these evaluations. The first part is the type compatibility checking, under the interpretations. The second part is concerned with the semantics of
well-typed expressions and predicates. The type inference rules for type checking is now introduced for the proposed model. The type checking mechanism takes care of sub typing introduced by the ISA relationship in an interpretation. Let, an interpretation $\langle D, r, ISA, AA, P \rangle$, with respect to which considered as the type checking and the semantics of expressions and predicates. In the calculus, every variable is typed with the typing function $\tau$. The typing function gives the most general type of the variable in the partial order, i.e. when there is a variable $x$ then by the domain assignment, it is immediate that $x$ may be bound to values from any domain $s$ of sort $s$, such that $s$ ISA $\tau(x)$.

Notations:

$\cup$ : disjoint union of sets
$\psi$ : a formula free of negation
$\psi(x)$ : a formula with free occurrences of $x$

$x$ may be a vector

$I_i$ : interpretation
$D_i$ : domain assignment of an interpretation $I_i$
$t_i$ : typing assignment of $I_i$
$AA_i$ : assignment of operation symbols, as given by $I_i$
$ISA_i$ : the ISA relationship of $I_i$
$P_i$ : truth value assignment to derived predicates, defined by rules
$f, f, f'$ : AAs and operation symbols
$i, j, k$ : bijections among sets of A As
$C, D$ : sort names
$\Sigma$ : signatures
$\vdash$ : is a model of relationship
$\circ \vdash$ : is a partial model of relationship
$\Rightarrow$ : (binary) logical implication
$\Leftrightarrow$ : (binary) logical equivalence

$i \circ j$ : prefix-composition of functions $i$ and $j$ (i.e. $i \circ j(x) = j(i(x))$).
4.2.4 The structure of interpretations.

**Definition 4.1** Signature, $\Sigma(I)$ (summation) of an interpretation $I = \langle D, r, ISA, AA, P \rangle$ is defined as the tuple $\langle r, ISA, sig \rangle$, where $sig$ is the disjoint union of the signatures.

**Definition 4.2** Two interpretations $I_1 = \langle D_1, r_1, AA_1, ISA_1, P_1 \rangle$, and $I_2 = \langle D_2, r_2, AA_2, ISA_2, P_2 \rangle$ are said to bear the same signature, written as $I_1 \text{cosig} I_2$, if $r_1 = r_2$, $ISA_1 = ISA_2$, and there exists a bijection $i : \cup_t AA_1(f) \rightarrow \cup_t AA_2(f)$ such that the following condition holds for all operation symbols $f$: if $f' \in AA_1(f)$ then $i(f') \in AA_2(f)$ and $\text{sig}(f') = \text{sig}(i(f'))$.

When such a bijection $i$ exists which may define as $i_1 \text{cosig} i_2$, and $i$ is called the corresponding cosig bijection.

This definition of co-signature of interpretations identifies disjoint semantic classes of interpretations. Two interpretations belong to the same semantic category if they can be identified by the same type structure, in the sense of typing of accumulated expression calculus. The following theorem ensures the closure of such semantic classes.

**Proposition 4.1** The relation cosig is an equivalence relation.

Interpretations and models belonging to the same semantic category compared here, i.e. those which are $\text{cosigs}$.

**Definition 4.3** An interpretation $I_1 = \langle D_1, r_1, ISA_1, AA_1, P_1 \rangle$ is said to be a sub-interpretation of $I_2 = \langle D_2, r_2, ISA_2, AA_2, P_2 \rangle$, written as $I_1 \subseteq I_2$, if the following conditions hold:

1. $I_1 \text{cosig} I_2$
2. for every sort $C$, $D_1(C) \subseteq D_2(C)$
3. for every operation symbol $f, \forall g \in AA_1(f), g \leq i(g)$
4. for every derived predicate $\psi, \forall \theta : f \text{ v } (\psi) \rightarrow I_1$, $I_1 \vDash \psi \theta \Rightarrow I_2 \vDash \psi \theta$
5. for every derived predicate $\psi, \forall \theta : f \text{ v } (\psi) \rightarrow I_1$, $I_1 \circ \psi \theta \Rightarrow I_2 \circ \psi \theta$

**Definition 4.4** Two co-signature interpretations $I_1$ and $I_2$ are said to be isomorphic (Similar form between individuals belonging to different species) written as $I_1 \sim I_2$, if the following conditions hold:

1. $I_1 \vDash \psi \iff I_2 \vDash \psi$
2. \( l_1 \models \varphi \Leftrightarrow l_2 \models \varphi \)

From the preceding definition of containment of interpretations and the model theory of the calculus, the following result is immediate.

**Lemma 4.2** \( l_1 \subseteq l_2 \) and \( l_2 \subseteq l_1 \Rightarrow l_1 \sim l_2 \). This lemma provides the antisymmetry required for the following lemma.

**Lemma 4.3** The sub-interpretation (\( \subseteq \)) relation over the domain of cosig interpretations factorized by the equivalence relation \( \sim \) induces a partial order.

**Theorem 4.4** The partial order (\( \subseteq \)) induces a lattice over the domain of cosig interpretations factorized by the equivalence \( \sim \).

**Proof.** I show existence of the \( sup \) and \( inf \), for any pair of cosig nonisomorphic interpretations. Let consider two interpretations \( l_1 \) and \( l_2 \), such that \( l_1 \sim l_2 \), and \( l_1 \cosig l_2 \).

Let \( l_i = \langle D_i, \tau_i, A A_i, \text{I S A}_i, \text{P}_i \rangle, i \in \{1, 2\} \).

\( (sup : ) \) Define \( sup(l_1, l_2) \) as \( l = \langle D, \tau, \text{I S A}, AA, \text{P} \rangle \), such that

s-i) \( I \text{ S A} = I \text{ S A}_1 = I \text{ S A}_2 \), \( \tau = \tau_1 = \tau_2 \)

s-ii) \( D(C) = D_1(C) \cup D_2(C) \)

s-iii) for every operation symbol \( f \), \( AA(f) = \{ f'' : \xi_1 \rightarrow \xi_2, \exists f' \in AA_1(f), \text{such that} \text{sig}(f) = \langle \xi_1, \xi_2 \rangle \text{ and } f'' = \text{lub}(f', f(f')) \} \). Here, it may be noted that, since \( i \) is the cosig bijection between \( l_1 \) and \( l_2 \), \( i(f') \in AA_2(f) \).

s-iv) \( \forall \emptyset : f(v(\emptyset)) \rightarrow I, P(v(\emptyset)) = P_1(v(\emptyset)) \vee P_2(v(\emptyset)) \)

Now it is to be shown that \( l = sup(l_1, l_2) \) is the supremum of \( l_1 \) and \( l_2 \).

The fact that \( l \) is an upper bound of both \( l_1 \) and \( l_2 \), follows from the definition. Let there be any other upper bound \( l_3 \) of \( l_1 \) and \( l_2 \).

Since \( D_1(C) \subseteq D_3(C) \) and \( D_2(C) \subseteq D_3(C) \), \( D(C) \subseteq D_3(C) \).

Since all the interpretations being considered are cosig interpretations, there are cosig bijections between \( l_2 \) and \( l_3 \) (say \( j \)) and also between \( l_3 \) and \( l \), say \( k \).

Therefore, I may assume, \( l_2 \cosig j l_3 \), and \( l_2 \cosig k \).
\( v \circ j \) becomes the cosig bijection between \( I_1 \) and \( I_3 \) and \( i \circ k \) becomes the cosig bijection between \( I_1 \) and \( I \).

Let consider any operation symbol \( f \) and \( f'' \in A.A(f) \). There exists \( f' \in A.A_1(f) \), such that 
\( f' \leq i \circ j (f') \) and \( \iota(f') \leq i \circ j (f') \) and by definition of \( \text{lub} \) of AA's, 
\( f'' = \text{lub}(f', \iota(f')) \leq i \circ j (f') \).

Let \( \psi \) be a derived predicate and \( \theta : \mathbf{fv}(\psi) \rightarrow I \) be a binding to \( I \), such that \( I \models \psi\theta \).

Therefore, \( I \) have two possibilities.
1. \( I_1 \models \psi\theta \).
2. \( I_2 \models \psi\theta \).

Hence, \( I_3 \models \psi\theta \).

Let \( I \models \psi\theta \). By construction of \( I \), it is implied that either \( I_1 \circ \theta \models \psi\theta \), or \( I_2 \circ \theta \models \psi\theta \). In either case it is implied that \( I_3 \circ \theta \models \psi\theta \).

Hence, \( I \subset I_3 \) showing that \( I \) is the desired supremum.

(*in f :*) Define \( \text{in f} (I_1, I_2) \) as \( I =< D, \tau, I S A, AA, P >, \) such that

i-i) \( I S A = I S A_1 = I S A_2, \tau = \tau_1 = \tau_2 \)

i-ii) \( D(C) = D_1(C) \cap D_2(C) \)

i-iii) for every operation symbol \( f \), \( AA(f) = \{f'' : \xi_1 \rightarrow \xi_2, \exists f' \in AA_1(f), \text{such that} \ \text{sig}(f') =< \xi_1, \xi_2 > \text{ and } f'' = \text{glb}(f', \iota(f'))\}

i-iv) \( \forall \theta : \mathbf{fv}(\psi) = P_1(\psi\theta) = P_1(\psi\theta) \wedge P_2(\psi\theta) \)

It is proved that \( I \) is the infimum. Let \( I_3 \) be a lower bound of \( I_1 \) and \( I_2 \).

\[ I_3 \models \psi\theta \Rightarrow \]
\[ I_1 \models \psi\theta \wedge I_2 \models \psi\theta \]
\[ \Rightarrow I \models \psi\theta. \]

\[ I_3 \sim \models \psi\theta \Rightarrow \]
\[ I_1 \circ \models \psi\theta \wedge I_2 \circ \models \psi\theta \]
\[ \Rightarrow I \circ \models \psi\theta. \]

Hence \( I \) is the required infimum.
I have defined the cosig relation for interpretations having a common signature. Let denote the common signatures by \( \Sigma \), so that \( \text{cosig}^\Sigma \) denotes the class of cosig interpretations with the common signature \( \Sigma \). This explicitly allows to talk of a semantic class at a time. In the followinge further structural properties of such semantic classes are explored.

**Theorem 4.5** The quotient \( \text{cosig}^\Sigma / \sim \) is a complete lattice.

**Proof:** Here the definitions of the supremum and infimum of a family of interpretations is provided. Let \( F \) be a family of interpretations in \( \text{cosig}^\Sigma / \sim \). Since \( F \) is a family of cosignature interpretations, between any two interpretations \( I \) and \( J \in F \) there exists a cosig bijection, say \( \iota \). Therefore, for every operation symbol \( f \) and every AA \( g \in AA(f) \), for any arbitrary \( I \in F \), one can define the following set of AAs

\[
g'_F = \{ h \mid h = \iota(g), I \text{ cosig } J \}.
\]

These sets are well defined, and define an equivalence partition over \( \bigcup_{l \in F} AA^I(l) \), where \( AA^I \) denotes the AA assignment AA as given by the interpretation \( I \).

The supremum \( \text{sup}(F) \) is the interpretation \( I = \langle D, \tau, I S A, AA, P \rangle \), where,

- \( D = \cup \{ D'(C) \mid l'(eF) = \langle D', \tau, I S A, AA', P' \rangle \} \)
- \( AA(f) = \{ \text{supg}_F \mid g \text{ AA}'(f), \text{ for some } l'(eF) = \langle D', \tau, I S A, AA', P' \rangle \} \)
- \( (\forall \theta : fv(\psi) \rightarrow l', l' \in F), P(\psi\theta) = \forall\{P'(\psi\theta) \mid \theta \text{ is a valid binding } l' \in F \}. \)

If \( \theta \) is not a valid binding for any \( l' \in F' \), then \( P(\psi\theta) = uu \).

The infimum \( \text{inf}_F (F) = \text{sup}\{l' \mid l' \subseteq I'', \forall I'' \in F \} \).

### 4.3 Informal Model of ODS Schema.

Given this structure of the interpretations, the ODS schema and its semantics are discussed now. An ODS schema consists of a hierarchy of classes. A class is either primitive or is constructed with the help of semantic constructs like specialization, aggregation and grouping. A class is associated with a number of messages. Each message, in turn, is associated with a number of methods. A method is specified by a piece of code, written using accumulated expressions. A class is also associated with constraints which restrict the semantics of objects belonging to the class.\(^{11, 17, 20}\) There
are also certain rules in an ODS schema, which are used to define different predicates e.g. the intentional predicates, integrity constraints etc. pertaining to the schema. Formally stated, an ODS schema, H, is a quadruple H =< CL,CSig,RULE >. The algorithm is as follows:

cursor bk is select
a.acc_no,a.title,b.cntrl_no,b.order_no,c.order_date,c.ven_code,
,d.inv_no,e.inv_date from books a book_accounts b
ord_accounts c
ord_inv d
inv_accounts e
where
a.cntrl_no=b.cntrl_no and
b.order_no=c.order_no and
c.order_no=d.order_no and
d.inv_no=e.inv_no;

- CL is a set of class names of all relation (i.e. Acc_no, Ctrl_no)
- ISA is a specialization hierarchy relation over CL, such that NIL ISA X, for all X \in CL, X ISA b.order_no= c.order_no for all X \in CL. (i.e. Bk.title ISA req.bktitle)
- C Sig assigns a specification to every class name C \in CL. Details of C Sig specification is given below.
- RULE is a set of rules in accumulated expression calculus.

Signature specification C Sig

The signature specification C Sig assigns to every class C \in CL a structure

\[ \text{CSig}(C) = \langle A,F,F,M,C \rangle \]

where \( A \subseteq CL \), (i.e. Fund_booked_date is subset of Bk_accounts)(D is subset...of A)

F is a set of simple arrows, called part-arrows \( f: C \rightarrow D, D \in A \),
F is a set of AAs, \( f: C \rightarrow D, D \in A \), called SET-arrows,
$M$ is a set of messages. Each message $m$ is associated with a collection $\mu(m)$ of AAs, called methods. A method is specified with a signature and a body. A method $AA$, $g$ is of the form $g : C_{\xi_1} \rightarrow \xi_2, \xi_2 \in CL^*$. A method is specified as follows.

```
if $a > a_n$ then
    return ($a + 1$);
else
    return ($a_i + 1$)
```

$g(x_i : s_1, \ldots, x_k : s_k \rightarrow x_{k+1} : s_{k+1}, \ldots, x_n : s_n)
(z_1 : t_1, \ldots, z_r : t_r, \psi)$

Here $\psi$ is an accumulated expression involving the variables $x_i, z_j$, and a special variable $self_C$. The type of the variable $x_i$ is $s_i$, and that of $z_i$ is $t_i$. In this method the variables $x_i, 1 \leq i \leq k$ are the input parameters, $x_i, k + 1 \leq i \leq n$ are output parameters and $z_i$ are the local variables. The expression $\psi$ is the body of the method.

$C$ is a constraint on the class $C$, which is a quantified formula of the form $\forall x \psi(x)$, where the type of the free variable $x$ occurring in $\psi$ is $C$.

Specification assignment is used for Dept.

```
CSig(Dept_acount) =< A, F, F, M, C >
```

```
A = \{string, Employee, Office
   
   Name: Dept \rightarrow Vharchar2

   F = \{Head: Dept \rightarrow Employee
   
   location: Dept \rightarrow Address

   F = \{employees : Dept \rightarrow Employee\}. The class Dept may contain a message faculty with methods all : Dept \rightarrow Employee Employee Employee and graded : Dept int \rightarrow Employee int. The specifications may be given as follows. In the specification I include explanatory comments enclosed by "/*" and "*/".

```
all(-> lect:Employee, rdr:Employee, prof:Employee)
/* This method is processed by a department in response to a message "faculty" received by the department object instance with respective parameters. This method finds the lecturers(lect), readers(rdr) and professors(prof) in the department, which receives the message.*/

```
(` (<self>employees(->lect); select lect where status(lect,l)),
```
Lecturers (lect) are those employees whose status is 1, Readers (rdr) are those employees whose status is 2, Professors (prof) are those employees whose status is 3 */

Another equivalent realization of the method body may be as follows.

All ( -> lect:Employee, rdr:Employee, prof:Employee)
/* This method is processed by a department in response to a message "faculty" received by the department object instance with respective parameters. This method finds the lecturers (lect), readers (rdr) and professors (prof) in the department, which receives the message. */

( emp: Employee

<self>employees(->emp);

((select emp where status(emp,l); (emp -> lect)),
(select emp where status(emp,2); (emp -> rdr)),
(select emp where status(emp,3); (emp -> prof)) )

This method graded is specified as follows.

graded(thres:int -> emp:Employee, age:int)
/* Find the employees and their ages who earn a salary more than a threshold salary thres */

( <self>employees(->emp) ;

select emp where (sal >= thres) and salary(emp,sal);
((emp -) emp),(<emp>age(-> age))
)
To introduce a constraint in this sample schema that, the head of a department works in the same department, expressed as

\[ \forall x: \text{Dept}(x, y) \Rightarrow \text{works_in}(y, z) \]

This is a constraint on the class Dept.

Again if to find all Title of book ordered where order_no=5057

In this case the method will be

Cursor bcu is select title, author, edition, quantity, pub_code, price, cur_code from book_temp

Where

Cntrl_no from book_account

Where

Order_no =5057);

create or replace procedure ex1(t in varchar2,v in varchar2)
as
cursor brqcurr is select * from books_request where title=t;
beginn

dbms_output.put_line('you wanna input'||t||'into book_temp');
for brec in brqcurr
    loop
        insert into book_temp(title,ven_code)
        values(brec.title,v);
        end loop;
end
/

4.4 Semantics of ODS Schema.

Semantics of a schema defines the domain of valid instances of databases and the semantics of queries which may be made on the databases. An ODS database instance consists of objects, which are instances of different classes. A database instance is
valid if it satisfies all the constraints and the semantic relationships expressed in the schema. A query on a valid database instance involves messages and predicates defined in the database schema. Message processing involves execution of methods. Result of a query depends on the semantics of these messages, methods and predicates.

The operations definable on database objects may be classified into two categories according to the nature of specification. The first category of operations includes the part-arrows and set-arrows, which are defined on different classes of objects using the constructs aggregation, grouping and inheritance in the specialization hierarchy. The other category consists of the messages. A database schema defines the signatures of all these operations and certain constraints. The aggregation defines the signatures of the part-arrows, which are simple arrows. The grouping construct provides the signatures of the set-arrows, which are accumulated arrows. A method specification provides the signature of an accumulated arrow, corresponding to the method. A message corresponds to an operation symbol which is associated with the accumulated arrows representing the methods which can be invoked by the message.

An ODS schema defines a signature of an accumulated expression calculus. The signature is \( \Sigma = \langle \tau, ISA^*, \Sigma_A \rangle \), where ISA is the reflexive and transitive closure of the ISA relation specified by the schema, \( \Sigma_A \) consists of the signatures of the methods corresponding to the different messages in different classes, \( \tau \) is the types of variables used in different contexts. At this point certain points must be clarified regarding the abstraction and scoping rules of the object oriented specification. In an object oriented language, classes provide encapsulation to all its elements, e.g. messages, parts etc. Therefore, it is possible that two different classes have messages or parts with the same name. Similarly, scope of a variable appearing in a method is localized within the method. Therefore, a variable appearing in different contexts may have different types. The actual typing and type compatibility is thus a function of the context, i.e. the method, or the class or the rule, where the variables appear. Therefore, to be strict about these important aspects of object encapsulation \( \tau \) as a function cannot be taken, nor the message names can treated as operation symbols of an accumulated expression calculus.
Since one can always consistently rename every message, variable in every expression and specification, one can avoid naming conflicts. This would, however, burden the compiler and its type checking procedure only. Therefore, in the following discussion it is assumed that such naming conflicts do not arise. Without any naming conflict, typing functions messages and operations using calculus of accumulated expressions can be treated. This makes the analysis simpler \(^1,11,28,29,30\).

An ODS schema thus defines an accumulated expression calculus along with the signature of its intended interpretations. The sorts of the calculus are given by the classes \(C \ L\). The operation symbols are given by the part-arrows, set-arrows and the messages, the derived predicate symbols are given by the rule heads. The constraints are formulae of the calculus. A database instance is an interpretation of the calculus which has the same signature (\(cosig\)) as the signature of the schema. A database instance is consistent if it is a model of all the constraints \(^3,32,35\). A database instance thus provides the accumulated operations corresponding to the different operation symbols. The operations for the part-arrows and the set-arrows are explicit and are given by physical inter-relationships among objects. The operation corresponding to a method is given as an expression or as a computation procedure.
4.5 Reference.


