Example 2. Ordering of members

The member records in OCSs of type DE are in descending order by SALARY. This is expressed as:

```
(C2):: (∀de ∈ DE) (∀e1,e2 ∈ EMP): Q(e1,e2)
                ⇒ (POS(D#DE[de],e1,Q) < POS(D#DE[de],e2,Q))
```

where the ordering predicate Q is

```
Q(e1,e2) ⇔ (D#EMP[e1].SALARY > D#EMP[e2].SALARY).
```

Note that D#EMP[e1].SALARY is shorthand for READ(D#EMP[e1],SALARY).

Example 3. Structural Constraint

Subordinate employees must be in the same department as their manager.

```
(C3):: (∀m ∈ MGR) (∀e ∈ EMP):
          (OWNS(D#MGR[m],e) ⇒
             (D#EMP[OWNER(D#MGR[m])].DNAME=D#EMP[e].DNAME))
```

It may be noted that both "recursive sets" (e.g., OCS type MGR) and structural constraints are proposed in CODASYL 78 [Manola 78].

Example 4. Duplicates Not Allowed constraint

The SSNO field in EMP records is a primary key.

```
(C4):: (∀e ∈ EMP):
       (COUNT(FIND(D#EMP, (VAR x: D#EMP[x].SSNO =
                                  D#EMP[e].SSNO)))
         = 1)
```

Note that in Example 4 we have used the cardinality function COUNT defined over the data type LIST.

Example 5. General Integrity Constraint

No employee can earn more than his manager.

```
(C5):: (∀m ∈ MGR) (∀e ∈ EMP):
          (OWNS(D#MGR[m],e) ⇒
             (D#EMP[e].SALARY
                        ≤ D#EMP[OWNER(D#MGR[m])].SALARY))
```

Note that in CODASYL DDL, one has to use a trigger procedure to check this constraint.

From a pragmatic viewpoint, this assertion language has the advantage that we can express a large class of integrity assertions non-procedurally. In contrast, CODASYL DDL provides special constructs for some of these constraints; all other constraints must be coded as trigger procedures.

A more important point is that this assertion language is backed by a proof theory of first order calculus augmented by the axioms of the data types TYPESTATE, RECORD and OCS. The proof theory allows us

- to prove that some properties of a database are valid over all database states
- to detect inconsistencies in integrity specifications during the database design phase
- to detect redundant specifications of integrity assertions, thereby minimizing the number of integrity assertions to be maintained.

We shall take up these uses of the proof-theory in detail in chapter 4.

3.3 Data Manipulation Language and Proof-rules

In this section, we design a simple data manipulation language over the primitive operations of the three abstract data types by integrating the data types with the control structures of PASCAL.

The integration is achieved by including the database objects in the execution environment of the programming language, so that a DML program can directly access the database objects. One consequence of this integration is that the programmer can use the control structures for manipulating the database in the same way as he uses them to manipulate the usual program variables. This advantage has been recognized in a number of recent database language proposals [Date 80], [Wasserman 79], [Schmidt 77]. The other consequence is that the user work area and currency pointers are eliminated from the description of the semantics of the DML statements. This makes the semantics of the DML statements "cleaner", thereby improving the

understanding of programs, and more importantly, simplifying the proof-rules necessary for program verification. (The reader may contrast our formulation with that in [Biller 76], which included the specification of the database I/O area in the language semantics).

3.3.1 Data Manipulation Language

We extend the type declaration facilities of PASCAL by including the schema definitions of RECORD and OCS. The variables introduced are **object variables** and **list variables**. A list variable can have as its value a list of object identifiers. An object variable (analogous to a tuple variable in relational languages) can have as its value an object identifier.

The DML statements are shown below. Out of them, statements M1 and M2 create new objects but assign some pre-defined values to the objects as well. Thus their semantics are given by interpretation in terms of object-creation and object-update statements of our object-manipulation language introduced in chapter 2. The statements M3-M6 update the database and their interpretation can be done in terms of the object-update statement only. The DML statements with their interpretive semantics are as follows:

M1. CREATE R(r) ⇔ [CREATE(R,r); r⇐EMPTY(R)]
This statement creates a record object of type R. The variable r denotes this stored record. The empty record-value is assigned to this record.

M2. CREATE L(l) WITH R(r) ⇔ [CREATE(L,l); l⇐MAKE(L,r)]
This creates an OCS object of type L, with the record denoted by r (of type R) as its owner. The variable l denotes this OCS. The empty OCS-value MAKE(L,r) is assigned to this OCS.

M3. CONNECT R(r) TO L(l) ⇔ [l⇐ADD(D#L[l],r)]. This connects the record denoted by r to the OCS denoted by s.

M4. STORE v IN R(r).F ⇔ [r⇐WRITE(D#R[r],F,v)]. This stores the value v in the field F of the record denoted by r.

M5. DISCONNECT R(r) FROM L(l) ⇔ [l⇐REMOVE(D#L[l],r)].
This disconnects the record denoted by r from the OCS denoted by l.

M6. DELETE S(s) ⇔ [s⇐undefined]. This deletes the object denoted by s. The object type is denoted by S.

The retrieval statement in our DML has the form:

M7. ASSIGN x WITH e ⇔ [x:=e]. The target variable x is either an object variable, a list variable or a usual PASCAL variable. The source expression e is a retrieval expression as defined below.

A **retrieval expression** is a term formed by function composition of only the observer functions defined on the data types TYPESTATE, RECORD and OCS. A retrieval expression can be used to form boolean expressions used for controlling WHILE statements or IF-THEN-ELSE statements.

The DML statements M1-M7 can be used in conjunction with all PASCAL statements. However, we introduce a specific form of FOR-statement [Hoare 72], which is useful for sequential processing of database objects. This statement is shown below:

M8. FOREACH x IN X DO M
Here X is a list variable (or a type name) and x is an element of the list (or object variable of given type). The statement M is repeatedly executed for each element in the list X. The statement M is not allowed to change either x or X.

An example program, based on the schema in figure 3-4, is shown in figure 3-5. This program is intended to give a 10% raise to all employees of the department that is the owner of an OCS object with identifier de.

3.3.2 Proof Rules

The interpretive semantics of the statements M1-M7 are given in terms of the statements from the object-manipulation language. Thus, we can use the axioms of object-creation and object-update, given in chapter 2, to derive the weakest preconditions for the DML statements. These are shown below:

```
VAR e: RECORD EMP;
    M: LIST OF MGR; de: OCS DE;
    s: REAL;
BEGIN
(* assume the object variable de has the identifier of
   the selected OCS object of type DE *)

   ASSIGN E WITH (MEMBERS(D#DE[de]));
   FOREACH e IN E DO
   BEGIN (* process this employee e *)
     ASSIGN s WITH (D#EMP[e].SALARY);
     IF   s <= 80K
          THEN
              STORE (s * 1.1) IN EMP(e).SALARY
   END
END.
```

Figure 3-5: Example of DML Program

A1. wp(M1,Q)⇔
(FRESH(D#R,r') ⇒ Q[D#R'/D#R][r'/r])
where D#R' ⇔ extend(D#R,r)[r <-- EMPTY(R)], and r' does not appear in Q or M.

A2. wp(M2,Q) ⇔
(FRESH(D#L,l') ⇒ Q[D#L'/D#L][l'/l])
where D#L' ⇔ extend(D#L,l)[l<--MAKE(L,r)]l, and l' does not appear in Q or M.

A3. wp(M3,Q) ⇔ Q[D#L'/D#L]
where D#L' ⇔ D#L[l<--add(D#L[l],r)]

A4. wp(M4,Q) ⇔ Q[D#R'/D#R]
where D#R' ⇔ D#R[r<--write(D#R[r],F,v)].

A5. wp(M5,Q) ⇔ Q[D#L'/D#L]
where D#L' ⇔ D#L[s <-- remove(D#L[s],r,R)]

A6. wp(M6,Q) ⇔ Q[D#S'/D#S]
where D#S' ⇔ D#S[s<-- undefined]

A7. wp(M7,Q) ⇔ Q[e/x]

The proof-rule for the statement M8 is taken from [Hoare 72] and is given below:

A8. (X = X1||[x]||X2),I(X1) {M} I(X1||[x])
⇒ I([]) {M8} I(X)
where [] denotes the empty list, and || denotes the list concatenation operator.

The proof-rules for the usual PASCAL statements are those given by Hoare and are shown in figure 3-6.

A9. Q[e/x]{x:=e}Q
A10. P{S}R, (R⇒Q) ⇒ P{S}Q
A11. (P⇒R), R{S}Q ⇒ P{S}Q
A12. P{S1}R, R{S2}Q ⇒ P{S1;S2}Q
A13. (P∧B{M}Q), (P∧ ¬B {M'}Q)
⇒ P{**IF** B **THEN** M **ELSE** M'}Q
A14. (P∧B{M}P) ⇒ P{**WHILE** B **DO** M}(P∧ ¬B)

Figure 3-6: Proof-rules of Pascal Statements

3.3.3 Proof Technique

We now discuss the use of these axioms in proving the correctness of DML programs. A program S is proved correct by showing that if an initial assertion P holds before the program, then a final assertion Q holds on completion of the program. That is, we have to prove a formula P{S}Q, written in Hoare's pseudological notation. Each of the axioms A1-A14 is of the form H1 **and** H2 **and** ... **and** Hn ⇒ P{S}Q. Hence, to show P{S}Q one has to establish the truth of each antecedent Hi. For example, consider axiom A13: in order to show P{if B then M else M'}Q, we have to show the antecedents (P **and** B){M}Q and (P **and not** B){M'}Q. Also, note that axioms A10 and A11 require us to prove **sufficiency conditions**, e.g., P⇒R in axiom A11.

Since these sufficiency conditions are formulas in our assertion language L, they can be proved using the proof theory of our assertion language (i.e., the axioms of the data types together with the proof rules of first order calculus).

To summarize, we obtained the DML by adding the statements M1-M8 to the programming language Pascal. The DML logic is obtained simply by adding the proof-rules A1-A8 to the original set of proof-rules of Hoare's logic and providing the proof-theory of the assertion language L. In the next chapter, we shall examine the application of both the DML logic and DDL logic in the context of Integrity Management.

Chapter 4

Application of DDL and DML Logic

In the previous chapter, we have set the stage for defining the problem of correctness of transactions, which we undertake here. Though the correctness of a transaction system in a multi-user environment is defined in terms of the serializability criterion [Eswaran 76, Bernstein 79], the notion of serializability assumes the correctness of individual transactions acting alone on the database. Here we shall discuss the application of our formalism in ensuring the correctness of individual transactions.

The correctness criteria of our interest, is that each individual transaction preserves the integrity assertions as invariants. Preservation of integrity assertions is important because the integrity assertions supply logical interconnections among the real-world entities and the database objects represent the real-world accurately only if they satisfy the integrity assertions at all times. We formulate this correctness criterion precisely in our formalism as follows. Viewing the integrity assertions as wffs in our assertion language L, a database state I is **consistent** with respect to a set W of wffs iff each wff in W is satisfied in I. A transaction preserves consistency iff its execution results in a consistent database state, if it is started in a consistent database state. A transaction M is **correct** iff for $w_i \in W$, W being the set of integrity assertions defined in a database schema, the following formula in our DML logic is valid:

$$(\wedge w_i)\ \{M\}\ (\wedge w_i).$$

We can apply the proof-techniques of the DML logic to prove such a formula. Thus, the problem of enforcing the correctness criterion on a transaction reduces to verification of DML programs.

Our viewpoint of verifying transaction as a means of integrity management is different from that taken previously in the literature. We are envisaging that the transactions are known apriori and are written by application programmers who have knowledge of the integrity assertions specified in the database schema. Most of the work (except that in [Gardarin 79]), on the other hand, assumes that naive end-users formulate the transactions without any knowledge of the integrity assertions and thus, the problem of ensuring the correctness of such transactions is dealt with by augmenting the user-transactions by run-time checks. There two methods of implementing the run-time checks have been used: the trigger procedures [Eswaran 75] and query modification [Stonebraker 75, Bernstein 82, Bernstein 80]. A trigger procedure is invoked in the event of updating a database object and the procedure checks the truth of the relevant integrity assertions in the updated database state. In the query-modification method, a pre-test is synthesized such that if the test succeeds, the following update operation is guaranteed to result in a consistent database state. Notice that irrespective of whether the tests are coded by the application programmer as in our case or augmented by system-supplied trigger-procedures or by query-modification, the ability to verify the correctness of the transactions with tests underlies the success of all the methods. Our DML logic provides us with precisely this ability. Moreover, if the verification process fails for a transaction, we may be able to synthesize a test to be added to the transaction which will rectify the failure.

There is another issue related to integrity management, but which has not been dealt with explicitly in previous work; namely, the issue of consistency of the integrity assertions among themselves. Before we verify transactions, we have to make sure at the very start that the given set of integrity assertions itself does not have any contradiction. If the set itself has contradictions, then there cannot be any database state which is consistent with respect to this set. As the integrity assertions are wffs in our case, we can apply logic techniques

to detect inconsistency of a given set W of integrity assertions by simply showing W⊢—false in our DDL logic.

For the rest of this chapter, we shall first describe a scenario of designing an application database to provide the context for discussing the issues of checking inconsistencies in a set of integrity assertions and of verifying the correctness of transactions. In section 4.2, we shall illustrate the use of DDL logic in checking inconsistency of schema definitions. In section 4.3, we shall illustrate the use of DML logic for enforcing the correctness of transactions. Finally, in section 4.4, we shall discuss the effectiveness of the program verification as a practical means for integrity management.

4.1 Database Design Scenario

The problems of correctness of transactions as well that of the consistency of schema definitions arise during the development of any database application. In the literature of database design methodology, the following stages of design activity are identified:

1. Requirement Analysis: the designer group gathers the expectations of the various user-groups in terms of their perception of real-world entities, their relationships and the expected usage.
2. View Representation: some means of representing the entities and relationships are used to capture the view of each user-group.
3. View Integration: individual user-view is merged to form the community view or the conceptual schema.
4. Schema Mapping: The conceptual schema is translated to the particular data model used specifically by the DBMS.
5. Transaction Design: Each usage of the data as expected by the users are coded in the DML of the specific DBMS.
6. Physical Database Design: Storage structure and access paths are defined to facilitate the execution of the transactions.

It is the view integration stage when the consistency of integrity assertions needs to be checked. As the integrity assertions from each user view are collated, there is a possibility that different user groups perceive the real-world differently and perhaps in a contradictory fashion. Detection of inconsistency of the integrity assertions is necessary to guard against such possibility.

Consider the following example of a university database. A user group representing the department adminstrations, may require that each teaching assistant, hired by a department, is registered for no more than 9 hours, because the departments expect the teaching assistants to perform their duties whole-heartedly. The user-group representing the Graduate School may, on the other hand, be interested in devotion of the graduate students to studies and therefore, require that each graduate student is registered for at least 12 hours. Finally, the user-group representing the Personnel Division of the university stipulates that each department must have some graduate student employed as teaching assistants. It is obvious that when these three requirements are put together, one finds that there can be no graduate student who meets both the departmental and the graduate school requirements. In fact, the practical database application will have a large number of such integrity constraints and some automated aid in detecting inconsistency is a must.

Once the conceptual schema is coded in the DDL, the application programmers can start designing the transactions, respecting the relevant integrity assertions for each transaction. In order to ensure the correctness of a transaction, the programmer has to insert appropriate tests in the code of the transaction. For example, before establishing the relationship of a graduate student as a teaching assistant in a department, there should be a test whether the student is registered for at most 9 hours (assuming the graduate school has relaxed its requirement). As such the task facing the programmer is not very

different than designing robust programs where the programs contain many data validation tests over and above the main computational task. For a database programmer, some automated aid to ascertain the correctness of his code with respect to the preservation of integrity assertion become very necessary, because of again the number of assertions involved. Better still, if a transaction is found incorrect, some aid to point out the causes of the failure (as given by the intelligent theorem-provers) are welcome.

In contrast with the work on relational databases, where some basis for design aids have developed, there is little available to the practitioners in dealing with such problems for Network databases. The main reason is the lack of adequate formal foundations for Network data model. Our work here is aimed to alleviate precisely this problem.

4.2 Detecting Inconsistencies in Schema Definition

Here we illustrate the use of DDL logic to detect inconsistency among a set of integrity assertions. As stated before, we can express the integrity assertions as wffs in the assertion language L and then use the axioms and the inference rules of its proof-theory to show that false can be derived from the set of wffs.

Returning to the example of the university database, we first assume the entities for departments (DEPT), graduate students (GRAD) and their relationship "employs as a teaching assistant" (TA) have been defined as shown in the figure 4-1. We can now express the three integrity assertions as follows:

Department:

If g is employed as teaching assistant in an instance t of the OCS type TA, then the hours for g is no more than 9.

$$w1::(\forall g \in GRAD)(\forall t \in TA):\ OWNS(D\#TA[t], g) \Rightarrow D\#GRAD[g].HOURS \leq 9$$

Graduate School:

For each graduate student g, the hours of g is at least 12.

w2::(∀g∈GRAD):D#GRAD[g].HOURS ≥ 12

Personnel:

Every department hires some graduate student as teaching assistant.

w3::(∀t∈TA)(∃g∈GRAD):OWNS(D#TA[t],g)

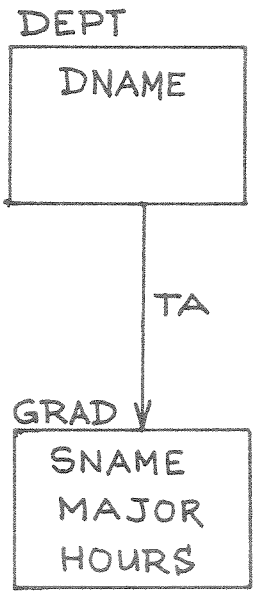

Figure 4-1: Diagramatic Schema for University Database

The set of these wffs W is w1∧w2∧w3 and we want to show W⊢—false. The deduction can be accomplished in the DDL logic T=(L,D), by applying the inference rules of T (namely modus ponens and generalization) to

the conjunction of W and D, the non-logical axioms of T. Instead of blindly applying inference rules to W∧D so as to derive all possible theorems, it is possible to use a refutation proof-procedure based on the resolution principle [Robinson 65]. Briefly, instead of deducing the theorem to be proved from the axioms, a refutation procedure tries to show that the negation of the theorem in conjunction with the axioms is unsatisfiable. Resolution based procedures starts with the negation of the theorem and the axioms, all after being converted to prenex normal form. In this form, the set of wffs are converted to clauses in which all variables are implicitly universally quantified and each clause is a disjunction of literals. Then pairs of clauses (or even a single clause) is checked for resolution i.e., if there are complementary literals which are unifiable under some substitution of terms for variables. The result of the resolution is the obtained by cancelling the literals resolved upon and forming a new clause which is the disjunction of the rest of the literals from the clauses involved, but after applying the unifying substitution to them. The process of resolution stops if ever a empty-clause (which is unsatisfiable) is obtained by resolution. The detail of the resolution based theorem-proving can be found in [Chang 73].

In order to derive false from W∧D, we apply the resolution based refutation procedure as described above. First of all we convert all the wffs into prenex normal form [Chang 73] as follows:

w1:: ¬OWNS(D#TA[t],g) ∨ (D#GRAD[g].HOURS≤9)
w2:: D#GRAD[g].HOURS≥12
w3:: OWNS(D#TA[t],f(t)), where f is the skolem function for the existential quantifier.

Consider also the following axioms of INTEGERs in D, which are written in the clausal form:

d1:: ¬(x≤y) ∨ ¬(y<z) ∨ (x<z)
d2:: ¬(x≥y) ∨ ¬(x<y)
d3:: 9<12

Now, our starting set of Clauses is {w1,w2,w3,d1,d2,d3,true} where true is the negation of the theorem to be proved. Resolving w1 and w3, we obtain

w4:: D#GRAD[f(t)].HOUR$\leq$9

Then, resolving w4 and d1 we obtain:

w5:: $\neg$(9<z) $\vee$ D#GRAD[f(t)].HOUR<Z

From w5 and d3, we obtain:

w6:: D#GRAD[f(t)].HOUR < 12

Now, from w2 and d2, we get:

W7:: $\neg$(D#GRAD[g].HOUR < 12)

Finally, resolving w6 and w7, we get the empty-clause, signifying that the set of clauses {w1,w2,w3,d1,d2,d3} is unsatisfiable.

4.3 Enforcing the Correctness of Transactions

Here we illustrate the use of DML logic in proving that a transaction is correct with respect to a given set of integrity assertions. Briefly, given a set $W=\{w_1,...,w_k\}$ of integrity assertions and a transaction M, we have to prove the following formula in our DML logic:

$$(w_1 \wedge ... \wedge w_k)\ \{M\}\ (w_1 \wedge ... \wedge w_k).$$

The transaction verification is a two-stage process: (1) generation of the verification conditions by applying the proof-rules of the language statements and (2) proving the verification conditions as theorems in the DDL logic. The verification conditions can be automatically generated in case of transactions which are free from loop constructs (FOREACH and WHILE statements). In case of loops, however, the proof-rules require introduction of extra assertions, called the loop invariants. Though loop invariants can be generated automatically for certain special cases of loops [Misra 77], the problem is unsolvable in general. Thus, we would assume that the transactions are annotated with the loop-invariants by the programmers. Proving the verification conditions can be done through the resolution-based procedure as discussed in the previous section.

To illustrate the process of transaction verification, we consider the dml program shown in figure 3-5 and integrity assertion C6 shown below:

```
(C6):: (∀e∈EMP): (e∈MEMBERS(D#DE[de])
                        ⇒  D#EMP[e].SALARY ≤ 88K)
```

This assertion states that the employees within the selected department can earn at most 88K as their salaries. In order to show that the program preserves integrity assertion C6, we have to prove the formula C6{Program}C6.

```
We postulate an invariant I(S)
I(S):: (E ⊆ MEMBERS(D#DE[de]))
   and ((∀i) (i ∈ S ⇒ D#EMP[i].SALARY≤88K))
   and ((∀j) (j ∈ (E - S)
                        ⇒ D#EMP[i].SALARY≤88K))
Let M denote the body of the FOREACH statement.
Define
 P ⇔ if D#EMP[e].SALARY ≤ 80K
     then (∀i) ((i ∈ (E1 || [e]))
                        ⇒ D#EMP[i].SALARY≤88K)
       and (∀j) (j ∈ (E - (E1 || [e]))
                        ⇒D#EMP[j].SALARY≤88K)
          else I(E1 || [e])
It can be mechanically verified that
           P{M}I(E1 || [e]).
```

This can be done by using axioms A13 and A4 to produce a sufficiency condition. By using axiom r2 of data type RECORD this sufficiency condition can be reduced to

$(y \leq 80K) \Rightarrow ((y * 1.1) \leq 88K)$

which is true.

We must next prove the tedious but trivial lemma:

```
(E = E1 || [e] || E2) and I(E1) ⇒ P
```

and this gives us by axiom A11 :

```
((E = E1 || [e] || E2) and I(E1)){M}I(E1 || [e])
```

The proof rule A8 for the FOREACH statement enables us to conclude

```
I([]){FOREACH e IN E DO M}I(E)
```

Thus we have shown that

```
I([]) :: (E ⊆ MEMBERS(D#DE[de]))
```

```
and ((∀ j) (j ∈ E
    ⇒ D#EMP[j].SALARY ≤ 88K))
```

Now by axioms A7 and A11, we generate another sufficiency condition

```
C6 ⇒ (∀j) (j ∈ MEMBERS(D#DE[de])
                ⇒ D#EMP[j].SALARY ≤ 88K),
which is also true.
```

This concludes the proof of the formula C6{program}C6.

4.4 Discussion

In the last two sections, we saw the examples of using the DDL logic and the DML logic to resolve the two issues related with integrity management, namely detecting inconsistency of schema definitions and enforcing the integrity assertions on the transactions. Questions remain as to how good are these methods. Here we try to answer some of these questions.

First question is if the schema is detected to be inconsistent, what action can the designer take. Can he detect which assertions are conflicting ? The answer is positive in the sense that the identification of the original wffs which caused the inconsistency can be obtained from the refutation tree if each resolution step is tagged with the identifications of the wffs to which the resolvent clauses belong. From this information, the designer can consult the respective user-groups to settle their differences of requirements.

As far as the transaction verification is concerned, it is admittedly expensive to execute. The main cost is incurred in proving the verification conditions. As a first cut to limit the number of integrity assertions taken into consideration, one can eliminate the integrity assertions to which no substitution has been made due to the application of DML proof-rules, because those assertions are unaffected by the transaction.

A better scheme would be to minimize the number of assertions in the

schema itself by removing the redundant ones. Again, we can use the DDL logic to formulate the redundancy criterion for a wff w in a given set W of wffs; w is redundant iff (W-{w})⊢w. If we could scan the wffs in a given set and remove the redundant ones successively, we would get eventually a minimal set of wffs which are free from redundancy. Unfortunately, this process cannot be implemented effectively, because if a wff w is not redundant, then the theorem-proving procedure may not terminate at all due to the unsolvability of the decision problem for an assertion language as general as first order predicate calculus. The same problem occurs when a transaction is incorrect because the proving the verification condition may not terminate; and in the case when a given set of wffs is indeed consistent. In order to remove these objections and lend practicality to the methods suggested here, we shall identify a rich subset of our assertion language for which theorem-proving is solvable. The next chapter is devoted to this issue of solvable subclass of our assertion language.

Chapter 5

A Decidable DDL Logic

In the previous chapter, we saw that solutions to the problems of schema design related to integrity assertions and of integrity management, all involve proving theorems in the DDL logic. As we want to build automated tools for solving these problems, we require a decision procedure for proving theorems in the DDL logic. Unfortunately, given that the language of our DDL logic as general as first-order predicate calculus, there is no algorithm which can decide whether or not an arbitrary formula is a theorem. In this chapter, we identify a decidable subset of the DDL logic and present a polynomial time decision procedure which can be used to decide whether a formula in this subset is a theorem or not. Here we first establish the necessary methods of proof and then discuss their applications to the problems of schema design and integrity management.

The decidable subset of the DDL logic consists of universally quantified Horn formulas (class UHF). By restricting ourselves to universally quantified formulas, we guarantee its decidability because such formulas can be transformed into equivalent function-free formulas in the Bernays-Schoenfinkel class, which is known to be decidable [Dreben 79]. Freedom from function symbols is the key to obtaining decidability. The choice to limit the formulas to Horn form is motivated by the fact that such formulas are amenable to efficient algorithms.

Though from the point of view of mathematical logic, the class UHF is one of simplest decidable classes, it turns out to be quite attractive for our

purposes. First, it is rich enough to include most of the integrity assertions that occur commonly in Network databases. Second, all the axioms of the DDL logic, which are used to prove theorems, are included in this class. These two facts together imply that the validity of a UHF formula can be proved or disproved algorithmically in the DDL logic. The schema design problems with UHF formulas used as integrity assertions are therefore solvable. In fact, we shall be able to use the decision procedure as basis for the solutions. Finally, under certain reasonable restrictions, if the transactions are annotated by the UHF formulas the verification conditions generated during transaction verification are of the form $A \Rightarrow B$, where both A and B are UHF formulas. The validity of such implication of UHF formulas in the DDL logic can also be proved by the decision procedure. Therefore, we are able to efficiently solve all of the three problems related to integrity assertions for a large number of cases, thereby meeting our goal.

Apart from the application of the decidability results to the database problems, our work here has two important aspects which we wish to emphasize. First, we show here that a class of network database transactions when annotated by the UHF formulas always produce verification conditions of the same form. This means that the theorem proving process follows the same pattern for every annotated program in the same class. Classifying annotated programs by the pattern of the VCs is important because specialized theorem provers can be designed for each class, thereby reducing the cost of theorem proving. Second, we present here a reduction technique which can be useful in identifying decidable subsets of theories involving data types such as our DDL logic. Our reduction technique transforms a formula in the theory of interest to an equivalent function-free quantificational formula. Function-free quantificational formulas serve as uniform basis for identifying decidable subcases of the theory of interest. For, they have been widely studied by the logicians and many solvable cases of them have already been identified.

The rest of the chapter is organized as follows. In section 5.1, we prove the basic decidability result, namely that the problem of logical consequence of universally quantified formulas is decidable. In section 5.2, we restrict the universally quantified formulas to Horn form and present a decision procedure for the logical consequence problem for such formulas. In section 5.3, we combine the results of the previous two sections to establish that the class of universally quantified Horn formulas are efficiently decidable and consider the problem of implementing algorithmic solutions to the schema design problem. Here we demonstrate that the decidable class includes most of the important integrity assertions and the axioms of the DDL logic. Finally, in section 5.4, we consider the problem of verifying transactions with respect to integrity assertions from the decidable class. We first identify some conditions under which transactions annotated by UHF formulas are verification decidable. Then we present some common transaction schemes which satisfy these conditions and are therefore verification decidable. We argue that these schemes encompass a large number of update transactions in network databases.

5.1 Decidability of Universally Quantified Formulas

In this section, we consider the class UF, consisting of universally quantified formulas with function symbols and show that the problem of logical consequence for this class is decidable.

In order to prove this decidability, we start with the knowledge that the satisfiability problem for the Bernays-Schoenfinkel class (the class BS) of function-free formulas is decidable. We then show that every formula with function symbols can be effectively reduced to an equivalent function-free formula, by systematically replacing the function symbols in the given formula by new predicate symbols. This reduction signifies that the satisfiability problem for the Bernays-Schoenfinkel class of formulas with function symbols (the class BSF) is also decidable. Finally, we show that the problem of logical

consequence for the class UF can be recast in terms of the satisfiability problem for the class BSF and hence is decidable.

The rest of this section is organized as follows. In subsection 5.1.1, we review some standard terminologies from mathematical logic. In particular, we take this opportunity to explain why the function-free formulas in the Bernays- Schoenfinkel class have a solvable satisfiability problem. In section 5.1.2, we present a reduction algorithm to remove the function symbols from any formula in the class BSF, thereby reducing it to a formula in the class BS. In subsection 5.1.3, we prove the decidability of the logical consequence problem for the class UF.

5.1.1 Background from Mathematical Logic

In most of our discussion about the classification of formulas, we shall be considering formulas in the prenex normal form. A formula in the prenex normal form (in short prenex formula) is of the form

$$(Q_1x_1)...(Q_nx_n)\ \ M$$

where $1 \leq i \leq n$ and each Q_i is either a universal or an existential quantifier. The part M is called the matrix and the sequence of quantifiers preceding it is called the prefix.

Most of the solvable cases of quantificational formulas are classified by either the pattern of the prefix or some restriction on the matrix or both. For example, the Bernays-Schoenfinkel class (the class BS) of formulas consists of those prenex formulas whose prefix is $\exists^*\forall^*$ (i.e., zero or more occurrences of existential quantifiers followed by zero or more occurrences of universal quantifiers) and the matrix has no function symbols. As we shall see, the absence of function symbols is the key to the solvability of this class.

Even if a prenex formula is free from function symbols, the Skolem functions may be introduced during the skolemization of a prenex formula. The process of skolemization removes the existential quantifiers from a prenex

formula as follows. If the existential quantifier Q_r in question has no universal quantifier appearing before it in the prefix, then we introduce a new constant symbol (Skolem constant), replace all x_r (the existentially quantified variable) from the matrix M by this constant and delete $(Q_r x_r)$ from the prefix. Otherwise, let $Q_{i_1},\ldots,Q_{i_k}$ be the universal quantifiers preceding Q_r. In this case, we introduce a new k-ary function f (Skolem function), replace all x_r in M by $f(x_{i_1},\ldots,x_{i_k})$ and delete $(Q_r x_r)$ from the prefix. Thus, every existential quantifier which is preceded by some universal quantifier, does in effect introduce a function symbol. It is easy to see that for a formula in the Bernays-Schoenfinkel class, this is not the case. Also, if we skolemize a given formula until all the existential variables are eliminated, we can delete the universal quantifiers as well resulting in a formula in Skolem form (in short a Skolem formula). In a Skolem formula, all the variables are regarded as universally quantified.

We now turn our attention to the satisfiability of formulas and explain why the testing of satisfiability of a formula in the Bernays-Schoenfinkel class is solvable. A formula is satisfiable if there is an interpretation in which the formula evaluates to true. A formula is unsatisfiable if in every interpretation the formula is false. By these definitions, in testing whether a formula is satisfiable or not, one has to try all possible interpretations over all possible domains and see if the formula is true is any of these interpretations - Obviously this is an infeasible task.

Fortunately, given a formula F, there exists a domain, called the Herbrand universe of F, such that it suffices to consider only the interpretations over this fixed domain. A Herbrand universe H_u for a formula is the set of all constants including the Skolem constants and all variable-free terms which are obtained by instantiating the terms of F by these constants and other terms already in H_u. A Herbrand base H_b for a formula F is the set of all variable-free atomic formulas which are obtained by instantiating the

atomic formulas in F by the terms of the Herbrand universe of F. A Herbrand expansion of F is the set of all variable-free instances of F obtained by replacing the variables in F by the terms of the Herbrand universe of F.

Example 5-1: Let F be the formula $(\forall x)(P(x)\Rightarrow P(f(x)))\wedge P(a)$. The Herbrand universe H_u for F and the Herbrand base H_b are $H_u=\{a, f(a), f(f(a)),...\}$ and $H_b=\{P(a), P(f(a)), P(f(f(a))),...\}$. The Herbrand expansion of F is $\{(P(a)\Rightarrow P(f(a))\wedge P(a)), (P(f(a))\Rightarrow P(f(f(a))))\wedge P(f(a))),...\}$. Notice that the infiniteness of H_u is caused by the presence of the function symbol f.

Returning to the question of testing satisfiability of a formula F, we now have to consider all possible truth assignments to the atoms in the Herbrand base of F and check for each truth assignment if all the members of the Herbrand expansion evaluate to true or not. This is still a formidable task. Because the possibility of the Herbrand expansion being infinite (as shown in the preceding example), we potentially have an infinite amount of checking to do. Thus, the simplest possible case where the satisfiability can be checked in a finite amount of time is when the Herbrand universe, and thereby the Herbrand expansion, is finite.

For a formula in the Bernays-Schoenfinkel class, the Herbrand universe is finite because there are no Skolem functions due to existential quantifiers nor are there any function symbols in the matrix. Thus, to test the satisfiability of a formula in this class, we have only a finite set of truth assignments over the atoms of the Herbrand base and a finite number of formulas in the Herbrand expansion to consider. Therefore, an algorithm can be constructed to test the satisfiability of these formulas.

5.1.2 Satisfiability of the Class BSF

In the previous subsection, we saw that the satisfiability problem for the class BS is solvable, because a formula in this class has no function symbol in its Skolem form. Here we show that the same is possible for the formulas in the class BSF because we can eliminate the function symbols from a BSF formula. In the next paragraph, we present an algorithm which eliminates the function symbols from a BSF formula by replacing them with equivalent predicate symbols.

The algorithm R is as follows: let w be a formula in which all variables are implicitly universally quantified.

step 1. If w has no function symbol then stop. Otherwise, if w has a term $f(\underline{x})$, $\underline{x}$ denoting a sequence of arguments to f, we create a new variable z and replace the formula w by the formula

$$f(\underline{x})=z \Rightarrow w(z).$$

Note that w(z) denotes the formula obtained by replacing all occurrences of $f(\underline{x})$ in w by z.

step 2. We introduce a new predicate symbol P_f and replace the formula obtained in the previous step by

$$P_f(\underline{x},z) \Rightarrow w(z).$$

Note that the intended interpretation of $P_f(\underline{x},z)$ is the statement that the result of applying f to the arguments $\underline{x}$ is z.

step 3. Go to step 1.

We claim that the algorithm R is correct. The algorithm terminates because the input formula w has only finitely many terms involving a function symbol and each iteration over the steps 1 and 2 removes at least one such term. The transformations to the formula accomplished by the steps 1 and 2 preserves the satisfiability because of the following proposition:

Proposition 5-1: w(f(x)) is satisfiable iff (f(x)=z⇒w(z)) is satisfiable.

Proof: (⇒) Let I be an interpretation in which w(f(x)) is true. If (f(x)=z) is true in I, then w(z)≡w(f(x)) is also true in I. If (f(x)=z) is

false in I, then $(f(x)=z)\Rightarrow w(z)$ is true in I. Hence, $(f(x)=z\Rightarrow w(z))$ is true whenever $w(f(x))$ is true.

($\Rightarrow$) Let $(f(x)=z\Rightarrow w(z))$ be true in an interpretation I. If $f(x)=z$ is true in I then, $w(f(x))\equiv w(z)$ is also true in I. Otherwise, let $f(x)=z'$. Then, $w(f(x))\equiv w(z')$ is also true in I. Hence, $w(f(x))$ is true whenever $f(x)=z\Rightarrow w(z)$ is true. ///

The formula obtained in step 2 is equivalent to $(f(x)=z\Rightarrow w(z))$ by definition of P_f and hence, by the proposition above, is satisfiable iff $w(f(x))$ is satisfiable.

The result of this algorithm R is a function-free formula:

$$P_{f_n}(\underline{x}_n,z_n)\Rightarrow(\ldots\Rightarrow(P_{f_1}(\underline{x}_1,z_1)\Rightarrow w(z_1,\ldots,z_n))$$

which is equivalent to

$$P_{f_n}(\underline{x}_n,z_n)\wedge\ldots\wedge P_{f_1}(\underline{x}_1,z_1) \Rightarrow w(z_1,\ldots,z_n).$$

Notice that the new variables $z_1,\ldots,z_n$ are all universally quantified. The algorithm R therefore transforms a formula w in the class BSF to a formula in the class BS. Now, we can state our intented result in the form of the following lemma:

Lemma 5-1: The satisfiability problem for the class BSF is decidable.

Proof: The satisfiability problem for the class BS is solvable and the algorithm R reduces a BSF formula to an equivalent formula in the class BS. ///

In the next subsection, we shall use this lemma in proving the decidability of the logical consequence problem for the class UF.

5.1.3 Logical Consequence Problem for the Class UF

We are now ready to prove that the problem of logical consequence for the class UF is decidable.

A formula w is a logical consequence of a set W of formulas iff for every interpretation in which each of the formulas in W is true, w is also true in the interpretation. We shall use the notation $W \models w$ in that case. The logical consequence problem is to test whether given W and w, $W \models w$?.

Here we are interested in the logical consequence problem where w and all formulas in W are in the class UF.

In order to prove that testing W |= w is decidable, we recast the problem in terms of the unsatisfiability of the formula W∧¬w as stated in the following lemma:

Lemma 5-2: Given a set W of formulas and a formula w, W |= w iff the formula W∧¬w is unsatisfiable.

Proof: (⇒) Let I be an arbitrary interpretation. If all formulas in W are true in I, then by the definition of logical consequence, w is true in I, i.e., ¬w is false in I. In that case, the formula W∧¬w is false in I. On the other hand, if there is some formula in W which is false in I, then W∧¬w is false in I. Thus, we have shown W∧¬w is false under all interpretations, i.e., the formula is unsatisfiable.

(⇐) Suppose W∧¬w is unsatisfiable, i.e., the negation of this formula is true under all interpretations. The negation of the formula is W⇒w. For any interpretation I, if W is true in I, then w is also true in I. Therefore, w is the logical consequence of W. ///

This lemma implies that in order to check whether w is a logical consequence of a set W of formulas, we have to check whether W∧¬w is unsatisfiable or not. If both w and the formulas in W are in UF, then W∧¬w is a formula with a prefix $\exists^*\forall^*$ (i.e., in BSF) because w and W have no bound variable in common. By lemma 5-1, we can check algorithmically whether the resulting BSF formula is satisfiable or not. In other words, we have the following theorem:

Theorem 5-2: Given a set W of formulas in the class UF and a formula w in UF, the problem of logical consequence is decidable.

Proof: Follows from lemmas 5-1 and 5-2. ///

Now that we know that there are algorithms for checking W |= w for the class UF, we would like to find an efficient algorithm. In the next section, we present one such efficient algorithm.

5.2 A Decision Procedure for Universally Quantified Function-free Horn Formulas

In the previous section, we saw that testing the logical consequence for the class UF of universally quantified formulas with function symbols, is decidable. Moreover, we saw that any formula in the class UF can be transformed by the algorithm R into an equivalent universally quantified function-free formula. Here we shall focus on this function-free form and try to find an efficient decision procedure for the logical consequence problem for function-free formulas.

We can achieve the goal of efficiency if we further restrict the matrices of the function-free formulas to the Horn-form. Thus, we shall consider only the class of function-free Horn-formulas in this section, and shall present an efficient decision procedure to test the logical consequence problem for this class of formulas.

The choice of dealing with only Horn formulas is not unnatural. The algorithm R produces a function-free Horn formula if the input to R is an implicational formula in the class UF. As we shall see in the section 5.3 on proving facts in our DDL logic, such implicational formulas include the axioms of the data types (the non-logical axioms of DDL logic) as well as many integrity constraints which arise naturally in the context of network databases. The decision procedure that we present in this section will therefore serve as a basis for many applications of our DDL logic.

The rest of this section is devoted primarily to the decision procedure for the logical consequence problem for the class of function-free Horn-formulas. After a brief subsection where we review the definition of Horn-formulas and see that the algorithm R actually produces such formulas, we shall present the decision procedure in subsection 5.2.2. In subsection 5.2.3, we prove the correctness of the the procedure and analyze its complexity.

5.2.1 Background

Any Skolem formula can be converted to a clausal form, called the conjunctive normal form (CNF). A CNF formula has as its matrix a conjunction of clauses, where each clause is a disjunction of signed atomic formulas. A Horn clause is a disjunction of signed atoms in which there is at most one positive atom. The following clause is an example of a Horn clause:

$$\neg P_1 \vee \ldots \vee \neg P_k \vee Q$$

in which Q is the only positive atom. Notice that this formula is equivalent to the following implication:

$$P_1 \wedge \ldots \wedge P_k \Rightarrow Q.$$

We shall call those CNF formulas with each conjunct being a Horn clause, a Horn formula.

To obtain some familiarity with Horn-formulas, let us consider the form of the formulas produced by the algorithm R. If the input formula w is an atomic formula $w(f_1(\underline{x}_1),\ldots,f_n(\underline{x}_n))$, then the result is a function-free Horn formula:

$$P_{f_n}(\underline{x}_n,z_n) \wedge \ldots \wedge P_{f_1}(\underline{x}_1,z_1) \Rightarrow w(z_1,\ldots,z_n)$$

Second, if the original formula w was of the form $A \Rightarrow B$, the result of R is

$$P_{f_n}(\underline{x}_n,z_n) \wedge \ldots \wedge P_{f_1}(\underline{x}_1,z_1) \Rightarrow (A(z_1,\ldots,z_n) \Rightarrow B(z_1,\ldots,z_n))$$

which is equivalent to

$$P_{f_n}(\underline{x}_n,z_n) \wedge \ldots \wedge P_{f_1}(\underline{x}_1,z_1) \wedge A(z_1,\ldots,z_n) \Rightarrow B(z_1,\ldots,z_n).$$

In other words, if the original UF formula was a Horn-formula, the result of R is a function-free Horn-formula.

Example 5-2: Consider the axiom o8 of the data type OCS, as shown in figure 3-2, which can be rewritten as a conjunction of the following three Horn-clauses:

```
(1) n=POS(ADD(s,r),r,Q)  ⇒ GET(ADD(s,r),n,Q)=r
(2) n>POS(ADD(s,r),r,Q)  ⇒ GET(ADD(s,r),n,Q)
                             =GET(s,n-1,Q)
(3) n<POS(ADD(s,r),r,Q)  ⇒ GET(ADD(s,r),n,Q)
                             =GET(s,n,Q)
```

Let POSP, ADDP and GETP be the three predicates corresponding to the function symbols POS, ADD and GET respectively. The result of the algorithm R are the three clauses:

(1') ADDP(s,r,z1) ∧ POSP(z1,r,Q,z2) ∧ GETP(z1,n,Q,z3)
∧ n=z2 ⇒ z3=r
(2') ADDP(s,r,z1) ∧ POSP(z1,r,Q,z2) ∧ GETP(z1,n,Q,z3)
∧ GETP(s,n-1,Q,z4) ∧ n>z2 ⇒ z3=z4
(3') ADDP(s,r,z1) ∧ POSP(z1,r,Q,z2) ∧ GETP(z1,n,Q,z3)
∧ GETP(s,n,Q,z4) ∧ n<z2 ⇒ z3=z4

The clauses (1')-(3') are Horn-clauses.

5.2.2 The Decision Procedure P

Here we present an algorithm which can decide whether a function-free Horn-clause is a logical consequence of a set of function-free Horn clauses or not. We choose to deal with Horn clauses rather than Horn formulas without any loss of generality, because a Horn formula is simply a conjunction of Horn clauses.

The idea behind the algorithm is as follows. Suppose we have a Horn clause c of the form $P_1 \wedge ... \wedge P_k \Rightarrow Q$, and we want to test whether c is a logical consequence of a set W of Horn clauses. Assuming the hypothesis part of c to be true, i.e., each of the P_i-s is true, we compute the set of all atoms which are implied by the clauses in W and then we check to see if the consequent Q of c is in the set of implied atoms.

We compute the set of implied atoms one by one by applying the clauses in W as production rules until no more new atoms can be generated. Whenever each of the atoms in the left hand side of a clause w in W "matches" under some substitution σ with some atom that has been implied so far, we can conclude that the atom on the right hand side w under the substitution σ is also implied and therefore add it as a new implied atom.

The process of matching an atom with another is similar to the notion

of unification as is used in resolution based theorem proving, but is relatively much simpler because the atoms in our case are function-free. For example, given two atoms P(x1,...,xn) and Q(y1,...,yn), they match if P and Q are same predicate symbol. The substitution in that case is σ={y1/x1,...,yn/xn}, meaning that if we substitute yi for xi, $1 \leq i \leq n$, in P and Q, then the resultant atoms are identical. Actually, in the course of applying a clause w in W as a production rule, a sequence of substitutions is computed, one substitution for each atom in the left hand side of w matching with some already implied atom, and the overall substitution σ is the composition of the individual substitution in the sequence. Let us take an example to clarify the process of applying a production rule.

Example 5-3: Let the production rule to be applied be

$$w \equiv P1(x1,x2,x3) \wedge P2(x4,x5) \Rightarrow Q(x3,x5)$$

and let the atoms P1(x1,x4,x5) and P2(x2,x3) be already in the set of implied atoms. Then, we get two substitutions corresponding to the two pairs of matching predicates, as follows:

$\sigma 1$={x4/x2,x5/x3} and $\sigma 2$={x2/x4,x3/x5}.

The overall substitution is

$\sigma = \sigma 1.\sigma 2$={x4/x4,x3/x3,x2/x4,x3/x5}={x2/x4,x3/x5}. In this case, the new implied atom is

Q(x2,x3,x5).σ=Q(x3,x3). The reader may refer to Chang and Lee [Chang 73, p.76] for further details on the composition of substitutions.

The algorithm P is formally presented in figure 5-1. We shall prove the correctness of the algorithm P and analyze its complexity in the next subsection.

5.2.3 Analysis of the Algorithm P

The correctness of the algorithm P is established by showing its termination and its soundness and completeness with respect to the derivation of c from the set W of clauses. We take up each of these in turn in the following paragraphs.

```
INPUT:: W is a set of n function-free Horn clauses
        c is a function-free Horn clause
OUTPUT:: "YES" if W |= c
         "NO" otherwise
DATA STRUCTURES:
  (1) The Horn clauses are represented by integers 0 to n.
      c is represented by 0 and the clauses in W by 1 to n.
  (2) LS[0:n] is an array of sets, containing the atoms
      in the left side of each clause.
  (3) RS[0:n] is an array of atoms, each representing the
      the right side a clause.
  (4) IMPLIED is a set of atoms found to be implied so
      far.
  (5) DONE is a boolean variable.

begin
   IMPLIED := LS[0];  DONE := false;
   repeat (* find new implied atoms *)
      DONE := true;
      for i:= 1 to n do (* check all of W *)
         if each atom in LS[i] matches with some
            atom in IMPLIED with the substitution σ
            and Rs[i].σ ∉ IMPLIED
         then begin
               IMPLIED := IMPLIED ∪ {RS[i].σ};
               DONE := false
              end
       end; (* end checking W *)
   until DONE; (* end generating new atoms *)
   if RS[0] ∈ IMPLIED
   then print "YES"
   else print "NO"
end (* end algorithm *)
```

Figure 5-1: Algorithm for Deciding Logical Consequence

Termination. We observe that the if-statement in the algorithm prevents introducing an atom if it is already in the set IMPLIED. Therefore all the atoms in IMPLIED are distinct. Given a predicate P with arity k and a set of v distinct variables, the maximum number of atoms possible is $a = v^k$, because the total number of ways of arranging k items out of v items with repetition of items being allowed, is v^k. Thus, if p is the total number of distinct predicate symbols in the input of the algorithm P, k is the maximum arity of a predicate symbol and v is the total number of distinct variable in the

input to the algorithm R, the size of the set IMPLIED is bounded above by $N=pv^k$.

Now, we observe that in each iteration of the repeat-loop, at least one new atom is added. Therefore, it takes only a finite number of iterations to generate all the possible distinct atoms in IMPLIED.

Complexity. The time complexity of algorithm P can be determined as follows. We observe that in each iteration of the repeat-loop, all the atoms in W are checked for matching against the current atoms in IMPLIED. Let |W| denote the total number of atoms in W, i.e., the length of the input. For each iteration, the number of atoms in IMPLIED grows at worst by one. Therefore, the total time spent in completing the repeat-loop is $O(N^2|W|)$, N being the maximum number of possible implied atoms. Finally, at the end of the repeat-loop, we have to scan the set IMPLIED to see if RS[0] is in it or not. Therefore, the total time spent by algorithm P is $O((N^2|W|)+N) = O(N^3)$, because $|W|<N$. We state this result as a theorem:

> **Theorem 5-3:** Given a set W of function-free Horn clauses and another function-free Horn clause c as input, the algorithm P decides whether $W|=c$ in time $O(N^3) = O((pv^k)^3)$, where p is the total number of distinct predicates in the input, v is number of variables and k is the maximum arity of a predicate symbol.

Notice that the maximum arity k is a constant ≤ 4 when we deal with DDL formulas. This decision procedure P is more efficient than the general purpose resolution based theorem provers which deal with general clauses (Horn and non-Horn) with function symbols and are of exponential complexity.

Soundness: If RS[c] has been added to the set IMPLIED, then RS[c] is logically implied by W and LS[c].

Proof: (basis)
Suppose RS[c] have been added to the set IMPLIED in the first iteration. Then, there is a formula $w \in W$ such that for some substitution σ, $LS[w].\sigma = LS[c]$ and $RS[w].\sigma = RS[c]$. By modus ponens, LS[c] and w then imply RS[c].

(induction)
Suppose RS[c] have been added to the set IMPLIED in n iterations. Let $w \in W$ be the formula that has been used in the nth iteration. Then, w must be such that under some substitution σ, RS[c] = RS[w].σ and for each atom $a \in$ LS[w], $a.\sigma \in$ IMPLIED. By induction hypothesis, each such $a.\sigma$ is implied by LS[c] and W. ///

Completeness: If LS[c] and W logically imply RS[c], then RS[c] is added to the set IMPLIED.

Proof: (basis)
Suppose c is implied by W in one derivation step. That is, there is a formula $w \in W$ such that LS[c] and w imply RS[c]. Then, under some substitution σ, LS[w].σ=LS[c] and RS[w].σ=RS[c]. As LS[c] is in the set IMPLIED at initialization, RS[c] will be added to the set IMPLIED by the if- statement.

(induction)
Suppose c is implied by W in n derivation steps. Let w be the last formula used to derive c. Then, under some substitution σ, RS[w].σ=RS[c] and LS[w].σ is implied by W. By induction hypothesis, atoms in LS[w].σ are all in the set IMPLIED. Therefore, RS[w].σ i.e., RS[c] will be added to the set IMPLIED by the if-statement. ///

To summarize, we have obtained an efficient decision procedure P by restricting our attention to function-free Horn clauses. The algorithm is similar to resolution-based theorem proving procedure with unit-support strategy [Loveland 78], but is more efficient. First, the absence of function symbols makes the unification algorithm to match atoms extremely simple. Second, the restriction of using only Horn clauses makes it possible to match clauses from the given set W against only the atoms in the set IMPLIED, instead of having to match clauses within the set W itself; the latter would have been the case if general clauses were used. In the next two sections, we shall see that the class of function-free Horn formulas include many naturally arising integrity assertions and the algorithm P provides a basis for solving several problems with integrity assertions efficiently.

5.3 The Decidable Subset of DDL Logic

Having developed the necessary decidability results in the previous two sections, we now turn our attention to the realm of databases and apply these results to the problems related to integrity assertions. We are interested in two problems here: that of checking consistency of a set of integrity assertions and of minimizing a set of integrity assertions in a given schema. In this section, we demonstrate that both the problems are efficiently decidable for a large class of integrity assertions that are important for Network databases.

In order to achieve efficient decidability, we shall focus on a subset of DDL logic which consists of universally quantified Horn formulas with function symbols (in short the class UHF). The problem of logical consequence for this class is efficiently decidable because one can transform formulas of this class into equivalent function-free Horn formulas by the algorithm R and then apply the decision procedure P on the function-free formulas. In subsection 5.3.1, we shall define this class UHF and show that it includes most of the integrity assertions of interest.

In subsection 5.3.2, we show that the two schema design problems can be algorithmically solved by the decision procedure P, provided we consider the integrity assertions only from the class UHF. In general, we explain here how a theorem in the fragment UHF of our DDL logic can be proved by the decision procedure P.

The capabilities that we develop here for the Network databases are similar to those developed by dependency theory for relational databases. In subsection 5.3.3, we discuss the similarities and differences of our formulation with those in the dependency theory.

5.3.1 Expressing Integrity Assertions in the class UHF

Before we consider the formulas in the class UHF, let us recall the syntax of the assertion language L of the DDL logic which we defined in Chapter 3. The set of predicate symbols in language L include the membership predicate $\in$, the equality $=$, the inequality NE, the predicates of the data types TYPESTATE, OCS, RECORD, LISTID and the comparison predicates LT, GT, LE, and GE from the implicit data type INTEGER. An atomic formula in L is obtained by applying a predicate symbol on the terms of L, the terms being in turn compositions of function symbols.

A formula in the class UHF is a closed formula in L with the individual variables being universally quantified and the matrix having either of the following forms:

1. an implicational formula in L of the form:
$$P_1 \wedge \ldots \wedge P_n \Rightarrow Q,$$
where $1 \leq i \leq n$, P_i and Q are atomic formulas and $n > 0$;
2. conjunction of implicational formulas.

Notice that disjunction of formulas such as "x is either a or b" are not allowed in the class UHF. The syntax for the formulas in the class UHF is chosen so that algorithm R transforms them into universally quantified function-free Horn formulas.

The integrity assertions that occur commonly in Network databases can be classified as constraints over record types, constraints over OCS types and constraints over multiple object types. In the following, we illustrate via examples that important assertions in each class can be expressed as formulas in the class UHF. The examples are based on the database schema shown diagramatically in figure 5-2.

Record Constraints

Example 5-4: ENO is a primary key for the record type EMP.

$(\forall e1)(\forall e2):(\in(e1,EMP) \wedge \in(e2,EMP)$

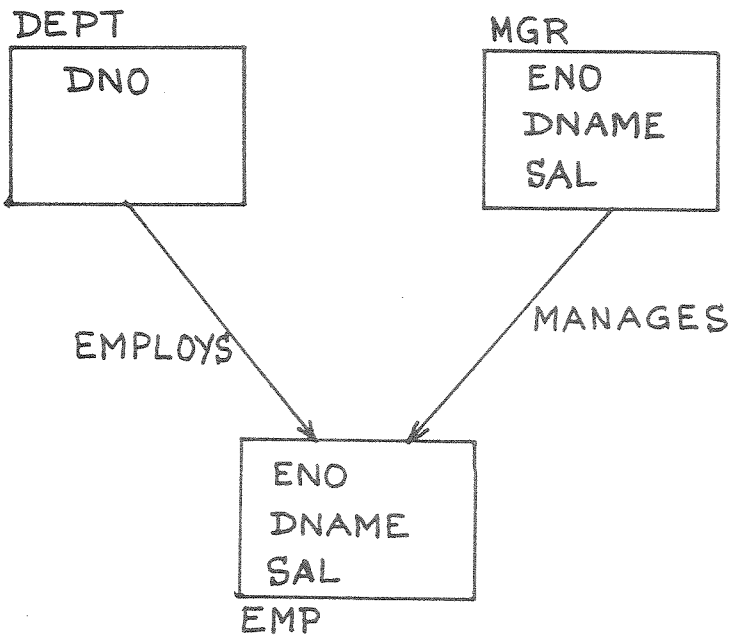

Figure 5-2: An Example Schema for a Company DB

$$\wedge\ \text{D\#EMP[e1].ENO} = \text{D\#EMP[e2].ENO})$$
$$\Rightarrow\ \text{e1=e2}$$

Note that this is an implicational formula.

Example 5-5: The values of the SAL field in record type EMP are in the subrange of integers 10 and 20.

$$(\forall e):\ (\in(e,\text{EMP}) \Rightarrow \text{GE(D\#EMP[e].SAL,10)})$$
$$\wedge\ (\in(e,\text{EMP}) \Rightarrow \text{LE(D\#EMP[e].SAL,20)})$$

This is a conjunction of two implicational formulas.

OCS constraints

Example 5-6: For the OCS type MANAGES, there is a structural constraint over the field DNAME.

```
(∀me) (∀e) (∀m) : (∈(me,MANAGES)
                   ∧(OWNER(D#MANAGES[me])=m)
                   ∧OWNS(D#MANAGES[me],e))
                   ⇒ D#MGR[m].DNAME=D#EMP[e].DNAME
```

Example 5-7: For the OCS type EMPLOYS, there is an ordering constraint that the members are positioned in the descending order of the field SAL.

```
(∀de) (∀e1) (∀e2): (∈(de,EMPLOYS)
    ∧ OWNS(D#EMPLOYS[de],e1)
    ∧ OWNS(D#EMPLOYS[de],e2) ∧ Q(e1,e2))
 ⇒ LT(POS(D#EMPLOYS[de],e1,Q),POS(D#EMPLOYS[de],e2,Q))
where the ordering predicate Q is

Q(e1,e2)≡ LT(D#EMP[e1].SAL,D#EMP[e2].SAL).
```

Example 5-8: Every employee has exactly one manager. This means an EMP record cannot be a member of more than one OCS. This is called the functionality of link constraint embedded in CODASYL data structures.

```
(∀e) (∀me1) (∀me2): (OWNS(D#MANAGES[me1],e)
                     ∧ OWNS(D#MANAGES[me2],e)) ⇒ me1=me2
```

Constraint over multiple object type

Example 5-9: The field DNAME in the record types MGR and EMP are synonyms, i.e., if there are two instances of these record types which agree in the key field ENO, then they must agree in the field DNAME. This is called an extended functional dependency (XFD) in [Casanova 83].

```
(∀m) (∀e): D#MGR[m].ENO=D#EMP[e].ENO
                ⇒ D#MGR[m].DNAME=D#EMP[e].DNAME
```

Example 5-10: The instances of the type MGR form a subset of the instances of type EMP. In other words, the OCS MANAGES could have been implemented as a recursive set.

```
(∀m): ∈(m,MGR) ⇒ ∈(m,EMP)
```

Notice that each of the above constraints are formulas in the class UHF. As shown in the examples, the class UHF includes, apart from general value-based constraints, several important constraints of CODASYL DDL [Codasyl 71].

One class of integrity assertions which cannot be expressed in the class UHF is that of existential assertions such as the Automatic Membership clause

of Codasyl (c.f. section 3.2) or the inclusion dependency constraint [Casanova 82] (a generalization of the subset constraint). An example of automatic membership constraint is that every instance of the record type EMP must be a member of some instance of OCS type EMPLOYS. In the general case, when the number of fields of the record types MGR and EMP are different, then the subset constraint signifying that every manager is an employee (ISA relationship), is an example of the inclusion dependency. For both these cases we need existential quantifiers, which are beyond the class UHF.

5.3.2 Solutions to the Schema Design Problems

This subsection discusses how the decision procedure P can be used to efficiently decide the schema design problem for integrity assertions of the class UHF.

Recall from Chapter 4 that both checking the consistency of a schema and minimizing the integrity assertions therein require deriving facts in the DDL logic. A schema with a set W of UHF formulas as integrity assertions is inconsistent if we can derive "false" from W. Similarly, in order to minimize the number of integrity assertions, we have to repeatedly detect and remove the redundant integrity assertions from the set W; an integrity assertion w is redundant in W if w is derivable from the set W-{w}.

We can recast the problem of deriving a formula w from a set W of formulas in DDL logic as a problem of logical consequence. A formula w is derivable in the DDL logic if it is true in every interpretation in which all the axioms of the DDL logic are true. That is, w is derivable in the DDL logic if $D \models w$ where D is the set of axioms of the DDL logic. Similarly, w is derivable from a set W of formulas in the DDL logic if $D \wedge W \models w$. Thus, the problem of detecting inconsistency reduces to testing $D \wedge W \models$ false; and the problem of detecting redundancy of w in a set W becomes testing $D \wedge (W-\{w\}) \models w$. We know that both these logical consequence problems are efficiently decidable

provided that the integrity assertions in the set W and w as well as the set D of axioms of the DDL are UHF formulas. As we have already decided to restrict ourselves to the class UHF for integrity assertions, the remaining question is whether the axioms of the DDL logic are also in the class UHF.

The set D of axioms of the DDL logic consist of the axioms defined with the specification of the data types TYPESTATE, OCS, RECORD, LISTID and the assumed data type INTEGER. The axioms of the data types are universally quantified equations or conditional equations. The equations are clearly in UHF because they are atomic formulas. In the following, we illustrate via examples that the conditional equations with some meaningful restrictions are also expressible as UHF formulas.

Example 5-11: Consider the axiom r2 of the READ function of the data type RECORD, as shown in figure 3-1. It can be rewritten as a conjunction of the following two formulas:

(1) READ(WRITE(s,F,d),F) = d
(2) NE(F_1,F_2)⇒ READ(WRITE(s,F_1,d),F_2)
= READ(s,F2)

Formula (1) is clearly a UHF formula. Formula (2) is a UHF formula provided the inequality predicate NE is available in the language, which is the case.

Example 5-12: Consider the axiom d6 for the FIND function of the data type TYPESTATE, shown in figure 2-4. The axiom can be rewritten as the following set of formulas:

(1) Q(i) ⇒ (FIND(CHANGE(D#T,i,o),Q)
= INSERT(FIND(D#T,Q),i))

(2) ¬Q(i) ⇒ (FIND(CHANGE(D#T,i,o),Q)
= FIND(D#T,Q))

The qualification expression Q is an open formula with one free variable. We claim that if Q(i) is a conjunction of atomic formulas, then both (1) and (2) are Horn formulas. Let Q(i)≡ $Q_1 \wedge ... \wedge Q_n$. Then (1) is clearly a Horn formula. To see why (2) would also be a Horn formula, we do the following transformations:

$$(2) \equiv \neg(Q_1 \wedge \ldots \wedge Q_n) \Rightarrow R$$
$$\equiv (Q_1 \wedge \ldots \wedge Q_n) \vee R$$
$$\equiv (Q_1 \vee R) \wedge \ldots \wedge (Q_n \vee R)$$
$$\equiv \wedge_{i=1}^{n} (\neg Q'_i \vee R),$$
where Q'_i is a negation of Q_i.

Because each Q_i is of the form $P(t_1,t_2)$, P being taken from the set {=, NE, LT, GT, LE, GE}, both Q_i and its negation Q'_i are expressible as positive atoms. Thus, the formula (2) is a conjunction of implications $Q'_i \Rightarrow R$, $1 \leq i \leq n$ and is therefore a Horn formula. Similar restriction of being positive atomic formula applies to the ordering predicate as used with the navigational functions POS and GET.

From these examples, we see that all the axioms of DDL logic are UHF formulas, provided the ordering predicates and qualification expressions are conjunctions of atoms. We now return to the question of implementing algorithmic solutions to the schema design problems, i.e., implementing the tests $D \wedge (W-\{w\}) \models w$ and $D \wedge W \models$ false. For the integrity assertions in the class UHF, the solution to the first problem is a straight forward application of the decision procedure P. Implementing the solution to the second problem needs some explanation. We observe that $D \wedge W \models$ false is equivalent to $D \models (W \Rightarrow$ false). Thus, the element LS[0] in the algorithm P will contain all the atoms in the set W as the left side of the formula to be proved. The element RS[0] will contain "false" as the atom of the right side of the formula to be proved. The algorithm will proceed as usual generating all the implied atoms by applying the axioms in D as production rules. The atom "false" will be generated as a new implied atom if ever some atom and its negation are both in the set of already implied atoms, because the tautology $P \wedge \neg P \Rightarrow$ false is in the set D of axioms. Thus, if the set of integrity assertions is inconsistent the atom "false" will be found in the set IMPLIED.

To summarize, we saw that the class UHF includes the axioms of the DDL logic. Deciding whether a UHF formula in the DDL logic is a theorem can

be accomplished by the decision procedure P. As most of the integrity assertions of interest in Network databases are expressible in the class UHF, we can again apply the algorithm P to solve the problems of schema consistency and minimization of integrity constraints for Network database schemas.

5.3.3 Relating the Results to Dependency Theory

The formulation of integrity assertions as logical formulas, allows us to compare the theoretical results of relational dependency theory to our work for Network databases. We observe that the class UHF is similar to the classes of dependencies considered in the relational dependency theory, namely the class of implicational dependencies [Fagin 82] and the class of generalized dependencies [Grant 82]. Both these classes consist of universally quantified function-free Horn formulas, identical to those obtained by transforming the UHF formulas by the algorithm R. In view of our solvability results, we know that the classes of implicational dependencies and generalized dependencies have solvable logical consequence problem. But the logical consequence problem for generalized dependencies is noted to have an exponential time complexity [Grant 82, Maier 79], while we have polynomial time complexity for the class UHF. The reason for this apparent discrepancy is due to the fact that the class of generalized dependencies are formulated as domain-relational calculus as opposed to the tuple-relational (using object-variables as individual variables) formulation of the class UHF. For domain-relational formulation, the relation symbols are treated as predicate symbols and hence, their maximum arity is dependent on the input to the decision procedure. For a tuple relational formulation, on the other hand, the predicates have constant arity, independent of the input. From theorem 5-3, we see that the complexity for domain relational formulas would be exponential on input while it is polynomial for a tuple relational formulation.

As stated in subsection 5.3.1, inclusion dependencies cannot be expressed in UHF. But the logical consequence problem for inclusion

dependencies is known to have a PSPACE-complete decision procedure [Casanova 82] in the general case and a polynomial decision procedure in a restricted case. The solvability results that we established here for the universally quantified Horn formulas can be used to explain why the restricted case (such as ENOs of MGR records is a subset of the ENOs of the EMP records) would have a decision procedure, because the domain relational formulation of this restricted inclusion dependency consists of universally quantified Horn formulas.

5.4 Verification Decidability of DML Transactions

We now turn our attention to the third problem of integrity maintenance, namely the problem of verifying DML transactions against integrity assertions. We are interested in those cases where the verification conditions (VCs) fall within a decidable class, because then we can declare a transaction to be correct or incorrect by proving or disproving the VCs. Following [Suzuki 80], we shall call a transaction verification decidable if all the VCs are within a decidable subset of the DDL logic. Of course, we require the VCs to be efficiently decidable by the algorithm P.

As the VCs depend upon both the transaction and the inductive assertions against which the transaction is to be verified, we shall actually consider annotated transactions, i.e., the transaction codes are interspersed with inductive assertions. Our main result in this section is that if a DML transaction is annotated with assertions in the class UHF and the conditional expressions within the transaction are conjunctions of atomic formulas, then the transaction is verification decidable. This is so because under these restrictions on annotated transactions, the VCs are all of the form $A \Rightarrow B$, where A and B are UHF formulas. Such formulas are decidable by the algorithm P, because $D \models (A \Rightarrow B) \Leftrightarrow D \wedge A \models B$, and the latter is a case of the logical consequence problem for UHF formulas.

This result by itself would not be so interesting if the annotated transactions encountered in practice during integrity enforcement did not obey these restrictions. In the case of enforcing integrity assertions, we know that the input/output assertions are the in the class UHF. If a transaction is a simple sequence of the DML statements M1-M7, i.e., without an alternative statement or a loop statement, there is no need for intermediate assertions and hence such straight line programs, being annotated by only the integrity assertions at their input and output, will always satisfy the restrictions of the theorem. For transactions with alternative statements, one may certainly choose to write a conditional expression which is not a conjunction of atomic formula. But such an alternative statement can always be rewritten as an equivalent one with only atomic formulas as conditions. Similar arguments apply to the conditional expressions controlling WHILE loops. That is, the limitation imposed on the conditional expressions is in principle not restrictive. The question as to whether annotated transactions with loop statements also satisfy the restriction is, however, open at this point. For, the loop invariants may not be UHF formulas even if the transaction has UHF formulas as input and output assertions. But, in many common cases the alternative and loop statements meet the restrictions of the theorem directly and thus lead to verification decidable transactions. In fact, we have identified three transaction schemes involving alternative and loop statements which are naturally verification decidable. These schemes include cases when a IF-statement is used to guard the update of several objects or a loop-statement is used to update a set of objects or loop-statements are used to select a set of objects and then update them.

The rest of the section is organized as follows. In subsection 5.4.1, we prove our main result by considering the proof-rules of the DML logic. In subsection 5.4.2, we consider a transaction scheme in which an IF-statement guards a sequence of updates and show that transactions instantiating this scheme are verification decidable. In subsection 5.4.3, we present a transaction

scheme in which a set of objects are updated inside a FOREACH statement. We show that the loop invariant for this scheme is a UHF formula leading to the verification decidability. We provide also an instance of this scheme as an example of a verification decidable transaction.

5.4.1 On Generating Decidable Verification Conditions

In this section, we prove the following theorem which states the necessary restrictions on the annotated transactions under which the VCs are formulas decidable by the algorithm P:

> **Theorem 5-4:** Let M be an annotated DML transaction. If (i) the inductive assertions used to annotate M are UHF formulas and (ii) the conditional expressions are conjunctions of atomic formulas, then the verification conditions are of the form of an implication A⇒B, A and B being UHF formulas.

Proof: We prove the theorem constructively by case analysis. Let us consider that we are interested in proving the Hoare formula $\alpha\{M\}\beta$, and α and β are formulas in UHF. As none of the proof-rules of the DML logic introduce an existentially quantified variable, we shall show in case of M that the VCs are of the form of an implication of Horn formulas A⇒B.

Case (i): Let M be either of the two CREATE-statements M1 or M2. From section 3.3, we see that the weakest pre-condition wp(M1,β) is

$$\text{FRESH(D\#R,i)} \Rightarrow \beta[\text{D\#R'/D\#R}][\text{i/r}].$$

Now, the consequent of this formula is a UHF formula β' because the substitutions do not alter the sequence of logical connectives in β. Thus, wp(M1,β) ≡ FRESH(D#R,i)⇒β', which is a Horn formula. Hence, the VC α⇒wp(M1,β) is an implication of two Horn formulas. Similarly for the case when M=M2.

Case (ii): Let M be either of the DML statements M3-M7. We observe that the weakest preconditions for these statements are all obtained by substitution to the post-condition β. Therefore, they are all Horn formulas. Hence the VC has the desired form of implication of two Horn formulas.

Case (iii): Let M be **IF** B **THEN** M_t **ELSE** M_f, where M_t and M_f are in the set {M1,...,M7}. The VCs in this case, as seen from the proof-rule A13, are the formulas:

(1) $(\alpha \wedge B) \Rightarrow wp(M_t, \beta)$
$\equiv \alpha \Rightarrow (B \Rightarrow wp(M_t, \beta))$

(2) $(\alpha \wedge \neg B) \Rightarrow wp(M_f, \beta)$
$\equiv \alpha \Rightarrow (\neg B \Rightarrow wp(M_f, \beta))$

If B is a conjunction of atomic formulas $B_1 \wedge ... \wedge B_n$, then the consequent part of formula (1) is clearly a Horn formula. To see why the consequent part of the formula (2) is also a Horn formula, we do the following transformations:

$\neg B \Rightarrow wp(M_f, \beta)$
$\equiv \neg(B_1 \wedge ... \wedge B_n) \Rightarrow wp(M_f, \beta)$
$\equiv (B_1 \wedge ... \wedge B_n) \vee wp(M_f, \beta)$
$\equiv \wedge_{i=1}^{n} (B_i \vee wp(M_f, \beta))$
$\equiv \wedge_{i=1}^{n} (\neg B'_i \vee wp(M_f, \beta))$
where each B'_i is a negation of B_i.

Because each B_i is of the form $P(t_1,t_2)$, P being one of {=, NE, LT, GT, LE, GE}, both B_i and its negation B'_i are expressible as positive atomic formulas. Thus, the preceding formula is a conjunction of implications $B'_i \Rightarrow wp(M_f,\beta)$, and is therefore a Horn formula. Thus, each of the VCs is an implication of two Horn formulas.

Case (iv): Let M be the loop-statement **WHILE** B **DO** M_r. Let the loop-invariant I be a UHF formula. From the proof-rule A14, we get the following VCs:

(1) $\alpha \Rightarrow I$
(2) $I \wedge B \Rightarrow wp(M_r, \beta)$
(3) $(I \wedge \neg B) \Rightarrow \beta$

VC (1) is clearly in the desired form. By the arguments similar to those given in case (iii), if the conditional expression B is a conjunction of atomic formulas, then both $(B \Rightarrow wp(M_r,\beta))$ and $(\neg B \Rightarrow \beta)$ are Horn formulas. Thus, VC (2) being equivalent to $I \Rightarrow (B \Rightarrow wp(M_r,\beta))$ is an implication of two Horn formulas. Similarly for VC (3).

Case (v): Let M be the loop statement **FOREACH** x **IN** X **DO** M_r. Let the loop invariant I be a UHF formula. From the proof-rule of the FOREACH statement, the following VCs are generated:

(1) $\alpha \Rightarrow I([])$
(2) $I(X1) \Rightarrow wp(M_r, I(X1||[x]))$
(3) $I(X) \Rightarrow \beta$.

As I is a Horn formula, I([]), I(X1) and I(X1||[x]) are all Horn formulas. Hence the VCs (1), (2) and (3) are all in the desired forms. ///

This concludes the proof of the theorem. Given a VC of the form A⇒B, we can decide whether it is a theorem in the DDL logic or not by posing the question D∧A |= B to the algorithm P, and all the formulas being Horn formulas, the algorithm P produces the correct answer. Theorem 5-4 provides an easy way of determining whether a given annotated transaction is verification decidable or not, because the necessary conditions stated in the theorem can be readily checked by simple inspection.

As to the application of this theorem in practice, recall that the program verification is applied to only the update transactions to ensure the preservation of integrity assertions. Thus, the update transactions annotated at their input and output by the integrity assertions (i.e., by UHF formulas) constitute our domain of interest. The question is whether these transactions are always verification decidable. By theorem 5-4, the question boils down to whether the conditional expression in these transactions are always conjunction of atomic formula and the loop invariants are universally quantified Horn formulas. In the next two subsections, we take up answering these questions respectively.

5.4.2 The IF Statements Guarding Updates

We consider here a transaction scheme which updates a sequence of objects and may involve a conditional statement to preserve the integrity assertions. We argue that conditional expression for this scheme is always a conjunction of atomic formulas and hence the transactions following this scheme are verification decidable.

The transaction scheme M is as follows:

```
if B
   then begin
          x1⇐v1;
          ...
          xn⇐vn
        end
```

Let X represent the sequence of objects $<x_1,...,x_n>$ being updated in M and V represent the sequence of values being assigned to them. Now consider that we want to verify that the transaction M preserves the following integrity assertion w:

$$(\forall X)\ (P(X) \Rightarrow Q(X))$$

We are interested in finding what the conditional expression B should be such that w{M}w is true. We claim that either B is empty (no test required) or B is an atomic formula Q(V). To prove this fact, we resort to the proof-rule for IF-statement and see that w{M}w is true if the following VC is true:

$$w \wedge B \Rightarrow (P(V) \Rightarrow Q(V))$$
$$\equiv w \Rightarrow (B \wedge P(V) \Rightarrow Q(V))$$

Assuming the hypothesis w to be true, we need to show $B \wedge P(V) \Rightarrow Q(V)$. There are two cases regarding the truth of this implication. If P(V) is false, then the implication is vacuously true and hence, no test B is needed at all. On the other hand, if P(V) is true, then the implication is true if $B \Rightarrow Q(V)$ is true, which can be easily established if $B \equiv Q(V)$. Thus, the conditional expression B is an atomic formula Q(V). Notice that the condition B in this case may be synthesized in a manner of Query-modification [Stonebraker 75].

Actually, the situation can be generalized if w is a conjunction of several implicational formulas $\wedge_{i=1}^{n} P_i \Rightarrow Q_i$. In this case, the conditional expression B will be either empty or a conjunction of atomic formulas $\wedge_{i=1}^{n} Q_i(V)$. The following example illustrates the construction of the conditional expression.

Example 5-13: Consider the Company DB as shown in figure 5-2.

Suppose we want to write two transactions, HIRE(e,me) to add an employee e under a specific manager who owns the OCS me of type MANAGES, and RAISE(e,n) to give a raise of n thousand dollars to an employee e. The affected integrity assertions are the assertions shown in the examples 5-6 and 5-5 respectively and repeated here:

```
(1) ∈(me,MANAGES)∧OWNS(D#MANAGES[me]=m)
      ⇒ (D#EMP[e].DNAME
            = D#MGR[OWNER(D#MANAGES[me])].DNAME)

(2) ∈(e,EMP) ⇒ LE(D#EMP[e].SAL,20)
```

The transactions are shown below:

```
HIRE(e,me):: IF D#EMP[e].DNAME
                  = D#MGR[OWNER(D#MANAGES[me])].DNAME
             THEN CONNECT EMP(e) TO MANAGES(me);

RAISE(e,n):: IF LE(D#EMP[e].SAL+n, 20)
             THEN STORE (D#EMP[E].SAL+n) IN D#EMP[E].SAL;
```

In the case of the RAISE transaction, the conditional expression is obtained by substituting the updated value for the SAL field in the consequent part of the integrity assertion (2). The conditional expression in the HIRE transaction is also similarly obtained from the consequent of assertion (1) but is further simplified to the form shown above by taking into consideration the fact that the OWNER of the OCS named me and the record e are unaltered by the update. In both cases, we have the conditional expressions as atomic formulas.

The transaction scheme M is quite commonly used, because the IF statements are embedded in the transactions to guard against the integrity violation by the updates. Such transactions being verification decidable, we can detect algorithmically if any required test is not coded into the transaction.

In the general case when an alternative statement is used for purposes other than guarding updates, one may want to use conditional expressions which are not conjunctions of atomic formulas. But with any such IF statement

can be rewritten as another IF statement (possibly nested for disjunctive expressions) which uses only atomic formulas as conditional expressions. Similarly, one can rewrite any WHILE loop as one using only conjunction of atomic formulas as loop condition. Thus, the restriction on conditional expression is not one of principle but one of convenience. The purpose of the preceding discussion in this subsection was to show that in the case of alternative statements guarding update, conjunction of atomic formulas is the natural conditional expression if the integrity assertion being affected by the update is a UHF formula.

5.4.3 Transactions with Loops

In this subsection, we consider the question whether the loop invariants for update transactions which are annotated at their input and output by UHF formulas, are themselves UHF formulas. The question for an arbitrary transaction with loops, remains open at this point. Instead, we shall consider here two common cases of using a loop in an update transaction: when the loop scans a set of objects either to update the objects or to select a subset of them and then update the objects in the subset. In both cases, we show that a loop invariant which is adequate to prove integrity preservation by the loop, is a universally quantified Horn formula.

Consider the following update transaction scheme M_1 where a set X of objects are scanned:

$M_1(X,S)$:: foreach x IN X do S(x,y)

The loop body S is a group of statements and may either update a variable y by some function of the object x in X or update each object x in X by some function of the variable y. As we are interested only in proving that M_1 will preserve some integrity assertions, we can formulate an adequate loop-invariant independent of the loop body S.

Let a UHF formula $w \equiv \forall^* P \Rightarrow Q$ be an integrity assertion to be

preserved by the transaction scheme M_1, i.e., $w\{M_1\}w$ should be true. Actually, the integrity assertion w should be true for all values of the variables in the transaction scheme and as such the integrity assertion defines a closed domain. Misra [Misra 77] argued that a closed domain is a loop invariant. We utilize this idea to postulate the following loop invariant I:

$$\begin{aligned} I(Z) \equiv\ & (\forall i)(i \in Z \Rightarrow w(i,y)) \\ & \wedge\ (\forall i)(i \in (X-Z) \Rightarrow w(i,y)) \\ & \wedge\ Z \subseteq X \end{aligned}$$

The intuition behind this loop-invariant I(Z) is that Z being the already scanned portion of the input set X, the integrity assertion must hold for the objects in Z (because on exit w holds for all the scanned objects though possibly modified) as well as for the unscanned objects in (X-Z) (because initially all the unscanned objects do satisfy the integrity assertion). Clearly, I(Z) is a universally quantified Horn formula if w is one.

Now we claim that I(Z) is an adequate loop-invariant. To prove that we must show the following:

(1) $(\forall j)(j \in X \wedge w(j,y) \Rightarrow I(\Phi))$
Φ being the empty list,
(2) $I(X) \Rightarrow (\forall j)(j \in X \Rightarrow w(i,y))$,
(3) $(X = X1 \| [x] \| X2) \wedge I(X1)$ {S} $I(X1 \| [x])$.

The conditions (1), (2) and (3) are entry, exit and iteration conditions. The condition (1) and (2) are obvious because I(Z) includes by design both the input and output conditions. To show the condition (3), we must prove that the loop body S updates x (or y) in such a way that w(x,y) is true after the update. But in the previous subsection, we have already discussed the case of updating a single object and showed that guaranteeing satisfaction of the integrity assertion by the updated object can be accomplished by a test involving a suitable conditional expression. Assuming that w(x,y) is true after the single object update, the condition (3) easily follows.

Thus, an adequate loop invariant required to prove $w\{M_1\}w$ is a UHF formula. In many cases, however, the transaction scheme M_1 will be preceded

by an initialization to the variable X. In general, X will contain a set of objects which are selected according to some qualification expression. Achieving this selection in a navigational fashion, involves loops which are schematized in the following search scheme M_2:

```
M2(T,P,X)::
  X := Φ;
  foreach t in T do
        if P(t) then X := X||[t];
  end
```

Again, the adequate loop invariant I2 can be obtained by generalizing the input and output conditions i.e., the property P holds for every $x \in X$ and $X = \Phi$ initially,

$$I2(Z) \equiv Z \subseteq T \wedge (Z = \Phi \Rightarrow X = \Phi) \wedge ((\forall i)(i \in Z \wedge P(i) \Rightarrow i \in X))$$

Notice that I2(Z) is also a UHF formula, as long as P is a conjunction of atomic formulas. We have already discussed this issue about the form of qualification expressions and decided to use conjunction of atomic formulas for them (c.f. Example 5-12).

The two common cases of using loops in update transactions are prototyped by the transaction schemes M_1 and M_2 and the loop invariants for them are UHF formulas. By theorem 5-4, these transaction schemes when annotated at their input and output by UHF formulas, are verification decidable. We now take an example which instantiates the transaction scheme M_1 and observe that the loop invariant is indeed a UHF formula.

Example 5-14: Consider again the company DB shown in figure 5-2. Suppose we want to write a transaction TRANSFER(m1,m2) to effect transferring all employees managed by a manager m1 to another manager m2. Let the parameters actually denote the respective OCS of type MANAGES whose owners are the managers in question.

```
TRANSFER(m1,m2)::
   E := MEMBERS(D#MANAGES[m1]);
```

```
d := D#MGR[OWNER(D#MANAGES[m2])].DNAME;
(* d is the DNAME of the second manager, and
   E is the set of employees under the first manager
*)

foreach e IN E do
begin
   disconnect EMP(e) from MANAGES(m1);
   store d into EMP(e).DNAME;
   connect EMP(e) to MANAGES(m2)
end;
```

The transaction TRANSFER as coded above, scans each member of m1 (kept in E), removes it from m1, updates it so that the updated DNAME field of the employee record equals that of the owner of m2, and finally connects it to m2. The objects updated by this transaction are m1, m2 and the objects in E.

The following UHF formula w represents the structural constraint on the OCS type MANAGES:

$$w(e,m) \equiv OWNS(D\#MANAGES[m],e) \Rightarrow (D\#EMP[e].DNAME = D\#MGR[OWNER(D\#MANAGES[m])].DNAME)$$

We postulate the loop invariant I(Z):

$$I(S) \equiv S \subseteq E \wedge (\forall i)(i \in S \Rightarrow w(i,m2)) \wedge (\forall i)(i \in (E-S) \Rightarrow w(i,m1))$$

To show that I(S) is a loop invariant, we have to show

$E=S||[e]||S' \wedge I(S)\{$loop body$\}I(S||[e])$

The loop body makes sure that after e is connected to m2, the DNAME field of e equals the DNAME field of m2's owner, thereby making w(e,m2) true. With this fact, the preceding iteration condition easily follows. The exit condition I(E) clearly implies the integrity assertion w. The integrity assertion also implies the entry condition $I(\Phi)$ after E

has been substituted by the initialization MEMBERS(D#MANAGES[m1]). Thus, I(S) is an adequate loop invariant. Moreover, all the assertions involved in proving w{TRANSFER(m1,m2)}w are all UHF formulas, thereby making the annotated transaction verification decidable.

In the preceding paragraphs, we saw that to verify transactions instantiating the scheme M_1 preserves UHF formulas as integrity assertions, the loop invariants are also UHF formulas. In fact, the loop invariant states that the integrity assertion is true for the elements scanned (i.e., possibly modified) as well as for the unscanned elements. Therefore, values input to the loop at beginning of each iteration come from a class of values where every member satisfies the integrity assertion. Thus, a generalization of the integrity assertion is adequate as a loop invariant for the transaction schemes considered.

To summarize this chapter, we have identified a decidable subset of the DDL logic as the class of universally quantified Horn formulas. By restricting the matrices of the formulas to Horn form, we obtained on one hand, a relatively more efficient decision procedure for this class of formulas, as compared to the general purpose resolution-based theorem provers. For our specific domain of application, namely that of verifying DML transactions against integrity assertions, we can abandon the general purpose theorem proving in favor of the decision procedure P. In fact, development of such domain-specific theories are being advocated by the researchers in the program verification area [Good 82] and our work here may serve as a step in this general direction. On the other hand, from the point of view of Network databases, we obtained in the decision procedure P a basis for algorithmic solutions to three problems: detecting consistency of a database schema, minimizing the integrity assertions therein and verifying preservation of integrity assertions by a DML transaction. The scope of applicability of the solutions is fairly large because many naturally arising integrity assertions and update transactions obey the conditions applicability.

Chapter 6

Conclusion and Future Research

We have presented a logic for Network databases and its application to the problem of formulating and maintaining integrity assertions. We developed the logic by integrating techniques from program specification methodology, and showed how existing program verification methods can implement the applications of this logic to the problems of Network databases. Finally, we presented a decidable subset of this logic and a decision procedure for the logical consequence problem for this subset. We showed that application of logic techniques to the problems of Network databases can be implemented in many naturally arising cases by this decision procedure, thereby avoiding the use of general purpose program verifiers.

The work presented here has made contributions not only to Network database technology but also gives an important example of the state of the art in the specification and verification of programs. In the following, we list the contributions from the points of view of both database as well as programming methodology. Then, we give some future research directions.

6.1 Contributions

The contributions to the database technology are as follows:

(1) We provide a set of axiomatized primitive operations on Network database structures. Formal semantics of other DMLs can be given by interpreting them in terms of these primitive operations.

(2) We provide a logic for Network databases. This logic permits us to verify correctness of the two components of a database definition:

the schema and the transactions. Developing formal specification with an eye towards verification separates our work from the earlier work on formal specification of Network databases [Biller 76].

(3) We proposed transaction verification as a method of integrity management at compile time for Network databases. Network DBMSs at present use for integrity monitoring ad-hoc trigger procedures, which are executed at run-time and may even require transaction rollback. Compared to the earlier work in using compile time techniques for integrity management in relational databases [Hammer 78, Gardarin 79], we use neither general purpose program verifiers nor heuristics, but an efficient decision procedure to implement our method for a rich class of integrity assertions. Moreover, the formal specifications developed here for the network databases can easily be incorporated into an existing program verifier, which may use a general purpose theorem prover.

(4) We showed that the class of universally quantified Horn formulas are adequate also for a large class of integrity assertions for Network databases. For this class, we presented a decision procedure to solve the logical consequence problem. This procedure equips us with some capabilities towards automated schema design tools which are comparable with the capabilities provided by the relational dependency theory. In particular, we can algorithmically solve the problems of detecting inconsistency of a given schema and minimize the number of integrity assertions therein.

(5) We presented a methodology based on abstract data types to specify the database structures formally. The methodology should be useful for specifying other conceptual data models such as the entity-relationship model. Compared to the earlier attempts of applying abstract data type techniques to databases, our work is thorough and can handle the sharing of objects.

From the point of view of programming methodology, our work contributes the following:

(1) a practical solution to the problem of algebraically specifying the behavior of shared mutable objects. The solution, compared to the earlier work [Berzins 79], permits recycling of existing verifiers and specification checkers to verify programs using such objects.

(2) a class of verification decidable programs for a specific application domain. Programs in this class produces verification conditions which are all of the same form, namely an implication of two Horn formulas. The advantage of classifying annotated programs by the type of the verification conditions is that proving any member of this class requires the same structure for the proofs and the verifier can be tailor made for this class. Moreover, the applicability of the specialized proof procedure is readily determined in our case by simple inspection of the given transaction and its annotations.

6.2 Future Research Directions

As to the directions for future research, there are topics which emerged from the discussions in the various chapters. We list some of them in the following:

(1) We do not know whether the class of UHF formulas is a maximal subset of the DDL logic which is decidable. We have reasons to believe that larger classes which include some existential dependencies may still be decidable. For example, we know that the class of inclusion dependencies which is beyond the class UHF, has a decidable logical consequence problem. We need further investigations to find larger decidable subsets of our DDL logic.

(2) In case we detect a transaction being incorrect, it is desirable to synthesize code which will make it correct. The ability to infer the reasons for failure in proving a verification condition seems to be crucial as a first step towards synthesizing code. The work in the area of program synthesis may be applicable to this problem.

(3) There is definitely a need for gaining experience in the verification of DML transactions. The experience can be helpful in further identifying higher order properties which are often used in proving the transactions. These properties should be used as lemmas for proving transactions, rather than proving them from the scratch every time.

(4) Our goal is to build design aids for Network database designer. The ideas and algorithms presented here need to be implemented and tried out in practice.

bibliography

[Beeri 79a] Beeri, C., Mendelzon, A.O., Sagiv, Y. and Ullman, J.
Equivalence of Relational Schemes.
In *Proc. ACM Symposium on Theory of Computation*. 1979.

[Beeri 79b] Beeri, C. and Bernstein, P.A.
Computational Problems related to the Design of Normal Form Relational Databases.
ACM Trans. on Database Systems 4(1):30-59, 1979.

[Bernstein 79] Bernstein, P.A. et. al.
A Formal Model of Concurrency Control Mechanisms for Database Systems.
IEEE Trans. Software Engineering , May, 1979.

[Bernstein 80] Bernstein, P.A., Blaustein, B.T., and Clarke, E.M.
Fast Maintainence of Semantic Integrity Assertions using Redundant Aggregate Data.
In *Proc. Intl. Conf. on Very Large Databases*, pages 126-136. 1980.

[Bernstein 82] Bernstein, P.A. and Blaustein, B.T.
Fast Methods for Testing Qualified Relational Calculus Assertions.
In *Proc. ACM SIGMOD Conf.*. ACM, June, 1982.

[Berzins 79] Berzins, V.
Abstract Model Specification for Data Abstractions.
Technical Report LCS-TR-221, MIT, 1979.

[Biller 76] Biller, H., Glatthar, W. and Neuhold, E. J.
On the Semantics of Databases: The Semantics of Data Manipulation Languages.
In G. M. Nijssen (editor), *Modelling in Database Management Systems*, . North Holland, 1976.

[Casanova 80] Casanova, M. and Bernstein, P. A.
A Formal System for Reasoning about Programs Accessing a Relational Database.
ACM Trans. on Progamming Languages and Systems 2(3), 1980.

[Casanova 82] Casanova, M.A., Fagin, R. and Papadimitriou, C.H.
Inclusion Dependencies and their Interaction with Functional Dependencies.
In *Proc. ACM SIGMOD/SIGACT Symposium on Principles of Database Systems.* 1982.

[Casanova 83] Casanova, M.A. and Vidal, V.M.P.
Towards a Sound View Integration.
In *Proc. ACM SIGMOD/SIGACT Symposium on Principles of Database Systems.* 1983.

[Chang 73] Chang, C.-L and Lee, R.C.-T.
Symbolic Logic and Mechanical Theorem-Proving.
Academic Press, 1973.

[Codasyl 71] *Database Task Group of the CODASYL Programming Language Committee*
ACM, New York, 1971.

[Date 80] Date, C.J.
An Introduction to the Unified Data Language (UDL).
In *Proc. Intl. Conf. on Very Large Databases.* ACM, 1980.

[Dreben 79] Dreben, B. and Goldfarb, W.D.
The Decision Problem - Solvable Cases of Quantificational Formulas.
Addison-Wesley, 1979.

[Eswaran 75] Eswaran, K.P. and Chamberlin, D.D.
Functional Specification of a Subsystem for Database Integrity.
In *Proc. Intl. Conf. on Very Large Databases.* ACM, 1975.

[Eswaran 76] Eswaran, K.P. et al.
The Notions of Consistency and Predicate Locks in a Database System.
Comm. of the ACM 19(11), 1976.

[Fagin 79] Fagin, R.
Normal Forms and Relational Database Operators.
In *Proc. ACM SIGMOD Conf.*. 1979.

[Fagin 82] Fagin, R.
Horn Clauses and Database Dependencies.
Journal of the ACM 29(4):952-985, 1982.

[Flon 79] Flon, L. and Misra, J.
A Unified Approach to the Specification and Verification of Abstract Data Types.
In *Proc. of Symposium on Reliable Software*. IEEE, 1979.

[Gardarin 79] Gardarin, G. and Melkanoff, M.
Proving Consistency of Database Transactions.
In *Proc. Intl. Conf. on Very Large Databases*. ACM, 1979.

[Good 82] Good, D.I.
Reusable Problem Domain Theories.
Technical Report 31, Institute for Computing Science, The University of Texas at Austin, 1982.

[Grant 82] Grant, J. and Jacobs, B.E.
On the Family of Generalized Dependency Constraints.
Journal of the ACM 29:986-997, 1982.

[Guttag 78] Guttag, J. V. and Horning, J. J.
The Algebraic Specification of Abstract Data Types.
Acta Informatica 10(1), 1978.

[Hammer 78] Hammer, M.M. and Sarin, S.K.
Efficient Monitoring of Database Assertions.
In *Proc. ACM SIGMOD Conf.*. 1978.

[Harrel 79] Harrel, David.
First Order Dynamic Logic.
In *Lecture Notes in Computer Science*, . Springer Verlag, 1979.

[Hoare 69] Hoare, C.A.R.
Axiomatic Basis of Programming.
Comm. of the ACM 12, Oct., 1969.

[Hoare 72] Hoare, C.A.R.
A Note on the For Statement.
BIT 12:334-341, 1972.

[Hoare 74] Hoare, C. A. R. and Lauer, P.
Consistent and Complementary Formal Theories of the Semantics of Progamming Languages.
Acta Informatica 2:135-155, 1974.

[Liskov 77] Liskov, B. et. al.
Abstraction Mechanisms in CLU.
Comm. of the ACM 20(8), 1977.

[Loveland 78] Loveland, D.
Automated Theorem Proving: A Logical Basis.
North Holland, 1978.

[Luckham 79] Luckham, D. C. and Suzuki, N.
Verification of Array, Record, and Pointer operations in Pascal.
ACM Trans. on Progamming Languages and Systems 1(2), 1979.

[Maier 79] Maier, D., Mendelzon, A.O. and Sagiv, Y.
Testing Implications of Data Dependencies.
ACM Trans. on Database Systems 4(4):455-469, 1979.

[Manola 78] Manola, F.
A Review of the 1978 CODASYL Database Specifications.
In *Proc. Intl. Conf. on Very Large Databases*, pages 232-242. ACM, 1978.

[Misra 77] Misra, J.
Prospects and Limitations of Automatic Assertion Generation for Loop Programs.
SIAM J. of Computing 6(4), 1977.

[Robinson 65] Robinson, J.A.
A Machine Oriented Logic Based on the Resolution Principle.
Journal of the ACM 12:25-41, 1965.

[Schmidt 77] Schmidt, J.W.
Some High Level Language Constructs for Data Type Relation.
ACM Trans. on Database Systems 2:247-267, 1977.

[Stonebraker 75] Stonebraker, M.
Implementation of Integrity Constraints and Views by Query Modification.
In *Proc. ACM SIGMOD Conf.*. 1975.

[Suzuki 80] Suzuki, N. and Jefferson, D.
Verfication Decidability of Presburger Array Programs.
Journal of the ACM 27:191-205, 1980.

[Wasserman 79] Wasserman, A.I.
The Data Management Facilities of PLAIN.
In *Proc. ACM SIGMOD Conf.*, pages 60-70. 1979.

VITA

Dipayan Gangopadhyay was born in Jalpaiguri, West Bengal, India, on December 1, 1951, son of Sudharani and Madhusudan Gangopadhyay. After completing his work at Sagore Dutt High School, Kamarhati, West Bengal, in 1967, he entered Jadavpur University, Calcutta, West Bengal. In 1972, he received the degree of Bachelor of Electronics and Tele-Communication Engineering with Honors and University Gold Medals. In 1973, he received the Post Graduate Diploma in Computer Science from the same university. Since August 1973, he held positions in research and development at DCM Data Products, New Delhi and Tata Institute of Fundamental Research, Bombay. During 1977-1978, he was invited as a Visiting Scientist at Hahn Meitner Institute, West Berlin, Germany. In January 1979, he entered the Graduate School of The University of Texas at Austin. Since September, 1983, he is employed as a Research Staff Member at IBM T.J. Watson research Center at Yorktown Heights, New York.

Permanent Address: 2389 Hawthorne Drive
Yorktown Heights, New York 10598

This dissertation was typed by the author using the Scribe document formatting system created by Brian K. Reid. The current version has been maintained and enhanced by Unilogic, Ltd. The Scribe format definitions for thesis format for The University of Texas at Austin were developed by Richard M. Cohen.