Describing Integrity Constraints in Advanced Database Systems

Yihua Shi † Haiyan Xu ‡ Tetsuya Furukawa ††

† Fukuoka Junior College of Technology
‡ Dept. Electronics, Fukuoka Institute of Technology
†† Dept. Economic Engineering, Kyushu University

Abstract

In advanced database applications, specifying integrity constraints in a declarative way allows to maintain the database consistency so that the database satisfies the integrity constraints. By associating each data with its status, in this paper, we show the consistency information of advanced database applications can be expressed declaratively by a set of clauses, which are classified into four basic categories. We give a semantics of them declaratively and discuss the tradeoff between the representation capability and the verification complexity of integrity constraints on some subclasses of the first order logic.

1 Introduction

In order to guarantee the accuracy or validity of data in databases, database management systems should be aware of the constraints which users must not violate. Consistency of a database is partly achieved by utilizing the concepts offered by the data model, and by proper design of the database schema. Data model consistency and schema consistency are collectively referred as the internal consistency. Consistency which could not be covered by the internal consistency must be specified in some suitable language. We call this kind of consistency the external consistency.

Typical database systems, however, do not have enough capability for the external consistency. Constraints specified declaratively are only basic ones. Integrity checking is mainly based on the assumption that each transaction keeps the consistency of the database if executed alone. It is desirable to be able to specify integrity constraints in a more declarative fashion so that database management systems can check the consistency. A system which supports such specification would relieve users of significant burden.

In these days, advanced database systems are required to support cooperative work such as CAD/CAM, office automation, VLSI design, software development, and so on. It is widely recognized that integrity management is an important function for these new applications. There are many research made in active databases, which try to maintain the consistency of the database automatically [8]. These researches, however, typically focus on integrity maintenance with relatively litter attention to the problem of how to describe the integrity constraints declaratively so that a database is consistent if it satisfies the constraints. This capability is useful in a wide spectrum of applications including query optimization [7], concurrency control [5, 10], and view materialization. On the other hand, the specification and enforcement of intertask dependencies or workflow has recently received much attention [1, 4]. If execution states of tasks are recorded by status, dependencies between tasks can be expressed by logical style easily.

In this paper, we first investigate the semantics of well-define environments for cooperative work. A cooperative work environment is *well-defined* if and only if all of the working data satisfy the constraints. By associating each data with its status and using a set of clauses, we propose a method to describe integrity constraints declaratively. The constraints are the same concepts as well-defined environments of the cooperative work or consistency of advanced database systems. We also discuss the tradeoff between the representation capability and the verification complexity of integrity constraints on some subclasses of the first order logic. Furthermore, we give a truth maintenance algorithm to treat updates of integrity constraints.

This paper is organized as follows. In Section 2, we give a basic data structure of advanced database systems, and show that the data structure is efficient to support the cooperative work environments. Section 3 briefly reviews the constraint specification in coopera-

tive work environments. Using the status of objects as predicate, in Section 4, it is shown that the constraints of cooperative work environments are declaratively described by a set of clauses. We also propose to use this set as the integrity constraints of the advanced database system. Section 5 discusses tradeoff between the representation capability and the verification complexity of integrity constraints on some subclasses of the first order logic. We also discuss the truth maintenance method to treat updates of integrity constraints in Section 6. Section 7 is the conclusions.

2 Advanced Database Systems

Unlike commercial environments, cooperative work environments are very complex and special. Typically, a group of users works in a cooperative work environment. The members of the group work cooperatively and coordinate their work. A lot of working data would be created during various working progress by the members. Furthermore, to make cooperative works be executed smoothly, all of the members should observe many rules and constraints.

In order to support the cooperative works, an advanced database system has to integrate and manage all of the working data and the constraints of the cooperative work environment.

In general, there are the following four kinds of working data in cooperative work environments.

1. Entity description data, such as a program source, an IC diagram, etc.
2. Relationship description data. They represent relationships among entities such as component relationships, equivalence relationships, history relationships, group relationships, etc.
3. Progress and situation description data. They represent progress of works or working situation of entities such as a program has been compiled, an IC has passed the logical simulation, programming for a program has been aborted, etc.
4. Constraints description data. They represent relationships among working progresses or working situations such as some works have to be executed in a fixed order, some works can be executed in parallel, etc.

As to the correctness of entity description data, there are two kinds of "correctness" of entities, in cooperative work environments. One is the "absolute correctness" of entities themselves, for example "*a program is correct.*" The other is the "relative correctness" of entities on the environment, for example "*a compiled program.*" Since the absolute correctness of entities depends on the application domain, it is very difficult that advanced databases support such correctness. We emphasis that the relative correctness is the correctness which should be supported by advanced databases.

In order to support the latter correctness concept, we introduce the basic data structure *object* by associating each entity with its status to record its current working situations.

Definition 1 *An advanced database* $DB\langle O, R\rangle$ *is composed of a set of objects* O *and a set of relationships* R.

- *An object* $o \in O$ *has two parts, the entity part* $e(o)$ *and the status part* $at_k(o)$ $(1 \leq k \leq n)$. *The status part is used to represent the situation of the object.*
- $R = \bigcup\{r \mid r \subseteq O^m\}$

$at_k(o)$ is defined as a predicate: $at_k(o)$ is true if and only if $e(o)$ has successfully finished the k-th work or $e(o)$ satisfies k-th property.

A basic data manipulation unit in advanced database systems can be an entity, a state, or a relationship, but not an object. Thus we use

$$\mathcal{DB} = \left\{ \begin{array}{l} e(o), at_k(o), \\ r(o_{i_1}, \cdots, o_{i_l}) \end{array} \middle| \begin{array}{c} o, o_{i_1}, \cdots, o_{i_l} \in O, \\ 1 \leq k \leq n, \\ r \in R, l > 1 \end{array} \right\}$$

to represent such basic data manipulation units in DB. We call each unit *a data item*. $I(d)$ is the value of data item d, and for a set of data items $D \subseteq \mathcal{DB}$, $I(D) = \{I(d) \mid d \in D\}$.

Based on such a data structure, we are able to consider the correctness of cooperative works.

3 Constraints Specification

In order to provide a well-defined part of the cooperative work for every member, a database has to treat many kinds of constraints. The constraints described below are the constraints required in cooperative work environments.

Sequential Constraints

During the execution of tasks in a cooperative work environment, some tasks have to pass through several work stages step by step according to a certain specified order while some tasks can be operated in parallel. For example, a VLSI design task may have to pass the work stages in order, logic design, circuit design, layout design, etc.

Constraints on sequential working procedure, called SE constraints, are essentially of the form:

> *Task o_j can enter work stage at_k after task o_i has finished work stage at_l.*

This type of constraints may capture both concurrent and sequential execution of tasks.

In general, there can be multiple constraints between tasks.

Alternative Constraints

There may be several methods to do a task. For example, software development task may design a program in different methods.

Constraints on alternative tasks or objects, called AL constraints, are essentially of the form:

> *Task o_j can enter work stage at_k after task o_{i_1} has passed work stage at_{l_1} or o_{i_2} has passed work stage at_{l_2} or $\cdots$ or o_{i_t} has passed work stage at_{l_t}.*

Exclusive Constraints

Although there are many tasks which can be executed in parallel at the same time, some tasks can not be executed at the same time. For example, a task and its contingency task can not be executed at the same time since the contingency task such as an abnormal treatment may be used if the main task has been aborted.

Constraints on exclusive tasks, called EX constraints, are essentially of the form:

> *Work stage at_k of task o_i and work stage at_l of task o_j can not be executed at the same time.*

Entity Constraints

An object is consistent if its status correctly express the current working progress/situations of its entity.

An entity and its status may be updated in various working stages separately. To keep the consistency between entities and their status, it is necessary that the constraints between the update operations of entities and status. For example, if a user updates an entity such as a program source, he/she needs to update its status such as compiled, tested, or debugged at the same time. We call such kind of constraints the entity constraints.

4 The Well-Defined Environments and Integrity Constraints

In cooperative work environments, the correctness of a work is judged generally by whether the work violates relevant constraints or not. A cooperative work environment called *well-defined* means every user's works, manipulations on entities or status, don't violate any given constraints. We use the concept of well-defined to represent the correctness of the whole environment.

In order to support cooperative work, an advanced database system needs to store and manage above constraints. We introduce a set of clauses to represent the constraints as follows.

By associating each object with its status, SE constraints can be abstracted as a partial order on the set of objects' status. Since status takes truth value, the order "$at_k(o_i) < at_l(o_j)$" can be represented by a clause "$at_k(o_i) \leftarrow at_l(o_j)$" naturally.

AL constraints such as task $at_l(o_j)$ should be carried out after task $at_k(o_i)$ or $at_t(o_i)$ is done, can be expressed as a clause "$at_k(o_i), at_t(o_i) \leftarrow at_l(o_j)$."

EX constraints such as task $at_k(o_i)$ and task $at_t(o_i)$ cannot be executed at the same time can be described by a clause "$\leftarrow at_k(o_i), at_t(o_i)$."

Using first order logic variables and the relationships among objects, we can express these constraints formally.

$$
\begin{array}{ll}
at_k(x) \leftarrow at_l(y), r(x,y) & \text{(SE clause)} \\
at_k(x), at_t(y) \leftarrow at_l(z), r(x,y,z) & \text{(AL clause)} \\
\leftarrow at_k(x), at_t(y), r(x,y) & \text{(EX clause)}
\end{array}
$$

where x, y, and z can be an $o \in O$ or a variable quantified on O universally, and the relationship r ($r \subseteq O^m$) can be used to quantify x, y, and z for a subset of O.

Definition 2 *Let $DB\langle O, R\rangle$ be an advanced database and P_S be the set of all of the situation predicate symbols.*

The integrity constraints of DB, denoted by $\boldsymbol{C}$, is a set of clauses:

$$A_1, \cdots, A_n \leftarrow B_1, \cdots, B_m$$

where $n \geq 0$, $m \geq 1$, the predicate symbols of $A_1, \cdots, A_n$ is in P_S, the predicate symbols of $B_1, \cdots, B_m$ is in $P_S \cup R$, and at the least one B_i is in P_S. The parameters of these predicates are some $o \in O$ or some variables quantified on O universally.

We use such $\boldsymbol{C}$, a set of clauses, to represent the constraints of the cooperative works in advanced database systems.

For simplicity, we except entity constraints from $\boldsymbol{C}$ since their verification is different from the others' as the entity constraints can be implemented by using the trigger [9].

In order to discuss the consistency of DB, we define the consistency of an advanced database in a logic style [6], and introduce some notations.

Definition 3 A *substitution* θ is a finite set of the form

$$\{x_1 := t_1, \cdots, x_n := t_n\},$$

where each x_i is a variable, each t_i is a term distinct from x_i, and the variables $x_1, x_2, \cdots, x_n$ are distinct. Each element $x_i := t_i$ is called a binding for x_i.

Definition 4 Let $\theta = \{x_1 := t_1, \cdots, x_n := t_n\}$ be a substitution and F be a logic formula. $F\theta$, the *instance* of e by θ, is the formula obtained from e by simultaneously replacing each occurrence of the variable x_i in F by the term t_i $(i = 1, \cdots, n)$.

Let S be a set of status. $T_{\{C,R\}}(S)$ is defined as follows:

$$T_{\{C,R\}}(S) = S \cup \left\{ A\theta \;\middle|\; \begin{array}{r} A \leftarrow B_1, B_2, \cdots, B_m \in \boldsymbol{C}, \\ \exists\theta \;\; \{B_1\theta, B_2\theta, \cdots, B_m\theta\} \\ \subseteq S \cup R \end{array} \right\}$$

Furthermore, we use an abbreviation:

$$T_{\{C,R\}}(S) \uparrow n = \underbrace{T_{\{C,R\}}(T_{\{C,R\}}(\cdots(T_{\{C,R\}}(}_{n} S))\cdots))$$

Definition 5 *Let $DB\langle O, R\rangle$ be an advanced database and $\boldsymbol{C}$ be integrity constraints of DB. DB is consistent if and only if $I(\mathcal{DB}), \boldsymbol{C} \not\vdash \bot$ *.*

To ensure the users work in a well-defined work environment, the advanced database only needs to keep the consistency of $I(\mathcal{DB}) \cup \boldsymbol{C}$. At the same time, we emphasis that the concept *well-defined* gives a declarative description of the semantics of consistency to advanced database systems.

After the integrity constraints $\boldsymbol{C}$ is defined, database systems have to check the satisfiability of $\boldsymbol{C}$. If there is a contradiction in $\boldsymbol{C}$, $\boldsymbol{C}$ cannot represent any well-defined environment. Generally, it is an NP-complete problem [3] to compute the satisfiability of a set of clauses which take their form as:

$$A_1, A_2, \cdots, A_n \leftarrow B_1, B_2, \cdots, B_m.$$

where $m, n \geq 0$ and $A_1, \cdots, A_n, B_1, \cdots, B_m$ are atoms. However, if clauses of "$A_1, \cdots, A_n \leftarrow$." are expected in the set of clauses, such set of clauses is always satisfiable.

In advanced database, $\boldsymbol{C}$ is used to manage the consistency of DB. If $\boldsymbol{C}$ includes some clauses in the form of "$A_1, \cdots, A_n \leftarrow$." or "$A_1, \cdots, A_n \leftarrow r(x_1, \cdots, x_m)$." such constraints would require that some situations of some objects can not be changed at any time. Therefore, we consider such constraints are very special and not necessary for the most of cooperative working environments. That is why we exclude such constraints from $\boldsymbol{C}$ in Definition 2 by $m \geq 1$ and $\exists B_i \in P_S$.

We can avoid such NP-complete computation. Satisfiability of $\boldsymbol{C}$ may be guaranteed by the initial DB where $\forall o \in O$ and $k = 1, \cdots, n, I(at_k(o)) = false$.

Proposition 1 *The integrity constraints $\boldsymbol{C}$ is satisfiable.*

Since integrity constraints $\boldsymbol{C}$ is defined with the constraints given by users of the cooperative work environment, it may include some "bad" constraints. It is desirable to provide a method to check obvious errors in the constraints.

Users have a final goal of their works, which can be represented by $I(D)$ $(D \subseteq \mathcal{DB})$. It is necessary to be able to check whether such a goal can be reached according to the integrity constraints set $\boldsymbol{C}$. Using the verification $I(D), \boldsymbol{C} \not\vdash \bot$, we can treat such a problem as follows.

Proposition 2 *Let DB, $\boldsymbol{C}$, $D \subseteq DB$, and $I(D)$ be an advanced database, its integrity constraints, a set of data items, and a goal of works, respectively.*

The goal $I(D)$ is unreachable if $I(D), \boldsymbol{C} \vdash \bot$.

However, $I(D), \boldsymbol{C} \not\vdash \bot$ is not enough to prove the goal $I(D)$ is reachable.

Although it is very primary, we consider it is an useful utility to treat such problems.

5 Consistency Verification

In order to support cooperative work, advanced database systems need to manage all of four kinds of working data shown in Section 2 in an unified way. In the paper [10], we have proposed a new correctness criterion of concurrency control for an advanced database DB based on its integrity constraints $\boldsymbol{C}$. Based on this new criterion, we argue to manage the cooperative works of the members by using the concurrency control function of advanced database systems, such as to manage the cooperative work by deciding whether

*$I(\mathcal{DB}), C \not\vdash \bot$ means "no contradiction is derived from $I(\mathcal{DB})$ and C."

a schedule keeps the consistency of DB, and ensures that users' work do not interleave each other.

To support concurrency control based on integrity constraints $\boldsymbol{C}$, we need to do a basic verification that $I(D)$ is satisfiable by $\boldsymbol{C}$,

$$I(D), \boldsymbol{C} \not\vdash \bot$$

where $D \subseteq \mathcal{DB}$ is a set of data items. That is, the users' works D must satisfy $\boldsymbol{C}$. In order to provide an efficient verification method for advanced database systems, we discuss the tradeoff between the verification complexity of consistency and expression power of integrity constraints based on the following subclasses of clauses.

General Clauses

Integrity constraints $\boldsymbol{C}$ defined in Definition 2 is a set of general clauses, which take the form as:

$$A_1, A_2, \cdots, A_n \leftarrow B_1, B_2, \cdots, B_m.$$

where $n \geq 0, m \geq 1$ and at least one situation predicate occurs the body of the clause.

General clauses can express all kinds of constraints (SE, AL, and EX), simple ones or complex ones. In this meaning, we say that their expression power is sufficient.

However, the consistency verification problem, $I(D) \cup \boldsymbol{C} \not\vdash \bot$ ($D \subseteq \mathcal{DB}$) is NP-complete. Thus, we can not provide an efficient verification procedure for such $\boldsymbol{C}$ and DB.

Horn Clauses

In order to avoid the problem of NP-complete, we consider a restricted class, called Horn clauses, which takes the forms as:

$$A \leftarrow B_1, \cdots, B_m \qquad \text{(SE clause)}$$
$$\leftarrow B_1, \cdots, B_m \qquad \text{(EX clause)}$$

Using Horn clauses, we can describe SE constraints and EX constraints.

For Horn clauses $\boldsymbol{C}$, $T_{\{C,R\}}(S) \uparrow n$ is monotonic and $\boldsymbol{C}$ does not include any function symbol. Thus, there is a fixed point for $T_{\{C,R\}}$ [6] and it can be computed as follows, where k is finite:

$$T_{\{C,R\}}(S) \uparrow \omega = T_{\{C,R\}}(S) \uparrow k = T_{\{C,R\}}(S) \uparrow (k+1).$$

$T_{\{C,R\}}(S) \uparrow \omega$ is called "a set of logical consequences of $\boldsymbol{C}$ and S."

Based on the fixed point semantics, we give a necessary and sufficient condition for the consistency verification.

EX clauses in $\boldsymbol{C}$ are rewritten as

$$EX \leftarrow B_1, \cdots, B_m.$$

by introducing a predicate EX.

Proposition 3 *Let $DB\langle O, R\rangle$ be an advanced database and $\boldsymbol{C}$ be its integrity constraints restricted as Horn clauses.*
$I(D) \cup \boldsymbol{C} \not\vdash \bot$
if and only if
$EX \notin T_{\{C,R\}}(D^+) \uparrow \omega$ and
$T_{\{C,R\}}(D^+) \uparrow \omega \cap D^- = \phi$
where $D \subseteq \mathcal{DB}$, $D^+ = \{d \mid I(d) = true, d \in D\}$, and $D^- = \{d \mid I(d) = false, d \in D\}$.

Since the fixed point of Horn clauses is computed in a polynomial time [2], *the satisfiability problem of $I(D) \cup \boldsymbol{C}$ is polynomial time computable.* In this meanings, we can provide an efficient verification procedure for such $\boldsymbol{C}$ and DB.

Specific Clauses

Although the AL constraints are very useful in many cooperative work environments, Horn clauses can not represent them.

In order to deal with the gaps between representation capability and complexity problem, we consider the specific clauses which may only express the commonest constraints and manage the consistency effectively by restricting the form of clauses.

As shown in Section 3, SE, AL, and EX constraints may be simplified to the forms:

$$at_k(x) \leftarrow at_l(y), r(x, y). \qquad \text{(SE clause)}$$

Its typical meaning is: if there is a relationship r between objects x and y, users have to complete i-th working stage of x before doing j-th working stage of y.

$$at_k(x), at_l(y) \leftarrow at_k(z), r(x, y, z). \qquad \text{(AL clause)}$$

Its typical meaning is: if there is a relationship r among objects x, y, and z, users have to complete i-th working stage of x or j-th working stage of y before to do k-th working stage of z.

$$\leftarrow at_k(x), at_l(x), r(x, y). \qquad \text{(EX clause)}$$

Its typical meaning is: if there is a relationship r between objects x and y, users can not complete i-th working stage of x and j-th working stage of y at the same time.

We should distinguish the situations (e.g. $at_k(x)$) from the relationships (e.g. $r(x, y)$) in advanced

database systems. We use integrity constraints to manage working situations, not relationships between entities. Relationship predicates are only used to bound the scope of constraints. In this means, we can assume relationship predicates appear only in the bodies of clauses.

Thus, Integrity constraints can be restricted by the forms:

$$\begin{array}{ll} A_1 \leftarrow B_1, \gamma. & \text{(SE clause)} \\ A_1, A_2 \leftarrow B_1, \gamma. & \text{(AL clause)} \\ \leftarrow B_1, B_2, \gamma. & \text{(EX clause)} \end{array}$$

where A_1, A_2, B_1, and B_2 are situation predicates and γ is a relationship predicate. We call such clauses specific clauses.

In order to verify the consistency of $\boldsymbol{C}$ and DB, we rewrite $\boldsymbol{C}$ to its contraposition, denoted by $\boldsymbol{C}^*$,

$$\begin{array}{ll} \neg B_1 \leftarrow \neg A_1, \gamma. & \text{(SE clause)} \\ \neg B_1 \leftarrow \neg A_1, \neg A_2, \gamma. & \text{(AL clause)} \\ \neg B_1 \leftarrow B_2, \gamma. & \text{(EX clause)} \\ \neg B_2 \leftarrow B_1, \gamma. & \text{(EX clause)} \end{array}$$

$\boldsymbol{C}^*$ is divided into two parts, $\boldsymbol{C}_1^*$ and $\boldsymbol{C}_2^*$, consisting of SE and AL clauses and EX clauses, respectively. $\boldsymbol{C}_1^*$ may be a set of definite clauses.

A necessary and sufficient condition for the consistency verification is given as follows.

Proposition 4 *Let $DB\langle O, R\rangle$ be an advanced database and $\boldsymbol{C}$ be integrity constraints of DB restricted to specific clauses.*

$$\Delta^-_{C_2^*}(D^+) = \left\{ B_1\theta \;\middle|\; \begin{array}{l} For\ \neg B_1 \leftarrow B_2, \gamma \in \boldsymbol{C}_2^*, \\ \exists\theta\ s.t. \\ \gamma\theta \in R\ and\ B_2\theta \in D^+ \end{array} \right\}$$

$$I(D) \cup \boldsymbol{C} \not\vdash \bot \quad \textit{if and only if} \quad T_{\{C_1^*, R\}}(D^- \cup \Delta^-_{C_2^*}(D^+)) \uparrow \omega) \cap D^+ = \phi,$$

where $D \subseteq \mathcal{DB}$, $D^+ = \{d \mid I(d) = true, d \in D\}$, and $D^- = \{d \mid I(d) = false, d \in D\}$.

Since $\Delta^-_{C_2^*}(D^+)$ is computed in polynomial time of $|D^+|$ and $T_{\{C_1^*, R\}}$ is a fixed point computation, *the consistency problem of advanced database systems, $I(D) \cup \boldsymbol{C} \not\vdash \bot$, is polynomial time computable.* Thus we can say that an efficient verification procedure can be provided to such $\boldsymbol{C}$ and DB.

Furthermore, we can extend the specific clause class as following forms easily and show the extended class has the same properties as the specific clause class.

$$\begin{array}{ll} A_1 \leftarrow B_1, \gamma. & \text{(SE clause)} \\ A_1, A_2, \cdots, A_n \leftarrow B_1, \gamma. & \text{(AL clause)} \\ \leftarrow B_1, B_2, \cdots, B_m, \gamma. & \text{(EX clause)} \end{array}$$

where $A_1, A_2, \cdots, A_n, B_1, \cdots, B_n$ are situation predicates and γ is a relationship predicate.

6 Truth Maintenance of Databases

During the use of a database DB, the set of integrity constraints $\boldsymbol{C}$ of DB may be required to change. Obviously eliminating an existing clause does not destroy the consistency of DB and $\boldsymbol{C}$. On the other hand, updating an existing clause or adding a new clause may destroy the consistency. In order to preserve the consistency of advanced database systems, a truth maintenance method (procedure) is necessary.

Let $DB\langle O, R\rangle$ be an advanced database, $\boldsymbol{C}$ and $\boldsymbol{C}'$ be old and new integrity constraints, respectively.

The *truth maintenance procedure* need to compute $I'(\mathcal{DB})$ which holds the following properties.

1. $I'(\mathcal{DB}), \boldsymbol{C}' \not\vdash \bot$.
2. $\forall d \in \mathcal{DB}$, $I(d) = true$ if $I'(d) = true$.
3. $\nexists I'' \neq I'$ s.t. $I''(\mathcal{DB}), \boldsymbol{C}' \not\vdash \bot$ and $\forall d \in \mathcal{DB}$, $I''(d) = true$ if $I'(d) = true$.

It is required to get the maximum subset $I'(\mathcal{DB})$ which satisfies $I'(\mathcal{DB}), \boldsymbol{C}' \not\vdash \bot$. Algorithm 1 is such algorithm to compute the maximum subset $I'(\mathcal{DB})$.

Algorithm 1
input: added/updated constraint set $\boldsymbol{C}(+)$ and current database $I(\mathcal{DB})$
output: truth maintained $I'(\mathcal{DB})$ the list of data resented

```
foreach x ∈ DB+ do
  foreach C : (A1, ···, An ← B1, ···, Bm) ∈ C(+) do
    if (I(x) = true and ∃θ x = Biθ)
        or (I(x) = false and ∃θ x = Ajθ) then
      begin
        Read I({A1θ, ···, Bnθ}) from I(DB);
        if Cθ=false then   /* a contradiction */
          begin
            select Biθ and reset it from I(DB);
            call propagate(Biθ);
          end
      end
  end
end
```

procedure propagate(x)
foreach $C : (A_1, \cdots, A_n \leftarrow B_1, \cdots, B_m) \in \boldsymbol{C}'$ **do**
 if $\exists\theta\ x = A_i\theta$ **then**
  begin
   Read $(A_1\theta, \cdots, B_m\theta)$ from $I(\mathcal{DB})$;
   if $C\theta$=false **then** /* a contradiction */
    begin
     select $B_i\theta$ and reset it from $I(\mathcal{DB})$;
     call propagate($B_i\theta$);
    end
  end
end

Let added/updated constraint set be $\boldsymbol{C}(+)$. In order to maintain the consistency of $I(\mathcal{DB})$ and $\boldsymbol{C}'$, we verify the consistency of $I(\mathcal{DB}) \cup \boldsymbol{C}(+)$. If they are consistency, $I(\mathcal{DB}) \cup \boldsymbol{C}' \not\vdash \perp$.

Otherwise, there are some contradictions between a clause and a tuple of data. Such as:
$A_1, \cdots, A_n \leftarrow B_1, \cdots, B_m \in \boldsymbol{C}$ and
$I(A_1\theta) = \cdots = I(A_n\theta) = false$ and
$I(B_1\theta) = \cdots = I(B_m\theta = true$.
$\{A_1\theta, \cdots, A_n\theta, B_1\theta, \cdots, B_m\theta\}$ is called a suspect list. To solve this contradiction, we only need to select some one from the suspect list to change it truth value. Since changing the truth value of $A_j\theta$ from false to true is unreasonable in the cooperative work environment, we select a culprit $B_i\theta$ and reset it. At this time, a policy how to select the culprit is needed if $m > 1$. Then, we have to treat the culprit's effect recursively.

This verification only ascertain the truth by substituting the values of predicates to a clause. Since a recursive call about *propagate*(x) reset a predicate to $false$, such recursive calls terminate in finite times.

Furthermore, if we restrict the $\boldsymbol{C}'$ as a set of specific clauses, we can avoid the indeterminate in how to select a culprit from the suspect list.

7 Conclusions

In order to use advanced database systems to support the cooperative work environments, it is required that the correctness of data in advanced databases agrees with the correctness of the corresponding cooperative work environments. The main contributions of this paper are follows.

1. An efficient method to represent cooperative work environments was given, which associates each data with its status and records the current working situations.

2. A declarative description for semantics of the integrity constraints of advanced databases was provided, and the verification method about the integrity constraints are proposed.

3. It was shown that in the classes of Horn clauses and specific clauses, their verification procedures are computable in a polynomial time, and that data satisfy the integrity constraints of the database if and only if the operations on these data is well-defined in cooperative work environments. Some methods(procedures) were given to maintain the changes of DB and $\boldsymbol{C}$ dynamically.

References

[1] Attie, P. A., Singh, M. P., Sheth, A. and Rusinkiewicz, M.: Specifying and Enforcing Intertask Dependencies, *Proc. Int. Conf. Very Large Data Bases*, pp. 134-144 (1993).

[2] Chandra, A. and Harel, D.: Structure and Complexity of Relational Queries, *J. Comput. and Syst. Sci. 25*, pp. 99-128 (1982).

[3] Cook, S. A.: The complexity of theorem-proving procedures, *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pp. 151-158 (1971).

[4] Elmagarmid, A. K.; *Database Transaction Models for Advanced Applications*, Morgan Kaufmann Publishers (1992).

[5] Harumoto, K., Yahata, T. and Nishio, S.: A Data Management Model for Computer Supported Cooperative Work (in Japanese), *Trans. IEICE*, Vol. J79-D-I, No. 5, pp. 271-279 (1996).

[6] Lloyd, J. W.: *Foundations of Logic Programming*, Springer-Verlag (1987).

[7] Michael, S., Edward, S. and Sharon, S.: A Method for Automatic Rule Derivation to Support Semantic Query Optimization, *ACM Trans. Database Syst.*, Vol. 17, No. 4, pp. 563-600 (1992).

[8] Stonebraker, M.: The Integration of Rule Systems and Database Systems, *IEEE Trans. Knowl. Data Eng.*, Vol. 4, No. 5, pp. 415-423 (1992).

[9] Ullman, J. D.: *Principles of Database and Knowledge-base Systems*, Vol. I, Computer Science Press (1988).

[10] Xu, H., Furukawa, T. and Shi Y.: Database Concurrency Control Utilizing Consistency Knowledge (in Japanese), *Trans. IPS Japan*, Vol. 35, No. 12, pp. 2752-2761 (1994).