TR-334

Deductive Approach For Nested Relations

by

K. Yokota

January, 1988

DEDUCTIVE APPROACH FOR NESTED RELATIONS

Kazumasa Yokota

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
junet: kyokota@icot.junet
csnet: kyokota%icot.jp@csnet-relay
uucp: {enea, inria, kddlab, mit-eddie, ukc}!icot!kyokota

ABSTRACT

A recent trend of database theory is representation of structured data and a deductive database. One such representation is a nested (or non-first-normal-form) relational model that is an extension of a relational model and inherits clarity of the formalism. A deductive database is also an extension of it in the framework of first order theory. This paper proposes a logic programming language called CRL for their integration and discusses a deductive approach for a nested relational model.

Table of Contents

1. INTRODUCTION

The relational model has played an important role not only as one of the data models but also as a theoretical backbone in the development of database theory. It has simple syntax and clear semantics, but restricts expressive capability and application domains. To extend the model while reserving the advantage, many approaches have been proposed, such as a deductive database, a nested relational model and a semantic model.

The deductive database approach is based on familiarity between a relational model and first order predicate logic [Gallaire 78]. As first order logic provides uniform description of tuples, inference rules and integrity constraints, its approach is intended to extend the model in the framework. The conventional approach to formalize a database is called model-theoretic. In it, relations are considered as interpretations of first order theory. On the other hand, in a deductive database, relations are considered as a part of first order theory, and the approach is proof-theoretic [Gallaire 84, Reiter 84]. In this framework, query processing can be considered as theorem proving in theory. Furthermore, many areas, such as introducing negative and disjunctive information and optimizing queries are studied as a deductive database in the uniform framework.

From an application point of view, it has been pointed out that a relational model has some difficulties in coping with applications for structured data, such as CAD/CAM, office automation and document processing, where the first normal form of a relational model is considered not to be appropriate. As one of the solutions of the problem, a nested (or unnormalized or non-first-normal-form) relational model was proposed [Makinouchi 77] and has been studied. There are many works on the model [Schek 82, Fischer 83, Abiteboul 84, Kambayashi 84, Pistor 86], and the first international workshop was held in April 1987 [Scholl 87].

At ICOT, many knowledge information processing systems have been developed and planned on the personal sequential inference machine (PSI). For such systems, especially for natural language processing and proof checking systems which are considered to play an important role in the fifth generation computer project, a database and knowledge base management system was planned to process various kinds of data and knowledge with complex structures in such applications. The KAPPA project was started for this purpose in September 1985. The system is based on a nested relational model from the viewpoints of efficiency of performance and representation of structures, and is intended to support deductive functions like a deductive database [Yokota 87].

However, it is only recently that research has been conducted in integration between a nested relational model and a deductive database, because it is very difficult to consider the model in the framework of the usual first order predicate logic. Recently, some formalism which corresponds to record structures of a nested relational model has been proposed in other fields [Ait-Kaci 84,86, Mukai 87, Rounds 86]. In the area of databases, a calculus for complex objects [Bancilhon 86, Abiteboul 86] and a logic for objects [Maier 86] have been proposed, with the possibility of representing nested relations although they are not explicitly related to them. As an extension of Prolog, LDL1 [Beer 87] and LPS [Kuper 87] has been proposed, aiming at a logic programming language for nested relations.

This paper presents a logic programming language called CRL, under some restrictions of nesting levels, which is intended to be a language for a deductive database based on nested relations. The next section explains some motivations of our approach, which is based on representation of record structures. Section 3 defines the syntax and semantics of a nested term for nested records, and Section 4 describes unification between nested terms. Section 5 defines a program based on nested terms, and Section 6 discusses relations between CRL and nested relations. Section 7 describes research plans and Section 8 compares this research to related research.

2. MOTIVATION

A 'record' is one of the basic representations of the real world and its structure appears not only in a database domain but also in various domains, and relates to objects, complex objects, feature structures, situations and other knowledge representation. In a deductive database for a relational model, a record (tuple) is flat and corresponds to a predicate. However, it has some problems.

First, it fixes the position of arguments (attributes). In the case of large records it is not appropriate for a user language because its predefined order and position are not essential for what he wants, and it is not data-independent if it is embedded in a program. Instead of fixed position notation, we use attribute-value pair notation:

person(galois, 1811) ⇒ person(last_name/galois, born_in/1811),

where the left side of / is an attribute name and the right side is the corresponding value. In this notation, the position and order are not fixed and they do not appear duplicated, that is, the pairs are commutative, associative and idempotent.

Secondly, a first order predicate also fixes the number of arguments. It is very tedious for a user to write all the arguments even if he uses anonymous variables. Hence, we admit partial description of attribute-value pairs:

person(_, 1811, _, _) ⇒ person(born_in/1811).

Using this notation, in a database a value not expressed explicitly is an unknown (applicable) value and in a user query it is out of his interest.

A nested record in nested relations can be considered as a combination of two kinds of nesting, column-nesting and row-nesting. A relational model does not admit column-nesting, however, column-nesting can be expressed in the usual first order predicate as hierarchy of functors and a predicate symbol (a relation name), and a predicate symbol is treated as a top-level functor. In our attribute-value pair notation, both functor and predicate symbols are also attributes, and column-nesting can be expressed uniformly in the tree of attributes:

person(name(evariste, galois), 1811)

⇒ person/(name/(first/evariste, last/galois), born_in/1811).

However, it is very difficult to express row-nesting in the usual first order term. A simple row-nesting (set value) can be expressed by distinguishing between a set domain and an atom domain, that is, by using two-sorted logic like [Kuper 87]. In CRL a power set of an atom domain is used for simplicity:

person/(name/{john}, hobby/{ski, tennis, baseball}).

Our notation admits deep nesting but current CRL is restricted to this simple row-nesting. For deep row-nesting, there are some problems to be solved, which are described in Section 7.

Even if a row-nested record can be expressed and stored in a database, a user language should be independent of the status of the database. Consider the following database (brackets are omitted in a case of a singleton):

person/(name/john, hobby/{ski, tennis}),

person/(name/{john, jack}, hobby/baseball).

To ask John's hobbies, the query should be written simply as person/(name/john, hobby/X), although we have not say the semantics. The answer is a set {ski, tennis, baseball}. That is, it is unnecessary for a user to know how a database is row-nested.

Our formalism is based on the above nested attribute-value pair. CRL is a logic programming language based on such record structures, and is intended to provide deductive functions to nested relations. An answer to a query is a set of values, or a nested subrecord which is partial information of some records and considered as constraints for them.

3. NESTED TERM

3.1 Syntax

We assume a set of constants, D, a set of variables, V, and a set of attributes, ATTR, where they are disjoint from each other. For a set-value, we use a power set of D (denoted by P(D)) as a domain of set-values, and simply call the element a constant. We add a special constant, ω, whose intended interpretation is an 'unknown value'. ATTR contains an empty attribute, <>.

Our language consists of the following symbols:

constants:	$c, d, \cdots$ ($\in P(D)$); ω,
variables:	$x, y, \cdots$ ($\in V$),
attributes:	$\langle\rangle, a, b, \cdots$ ($\in$ ATTR),
auxiliary symbols:	*, /, (,).

First, a nested attribute-value paired term (NAV) corresponding to a restricted nested record is defined:

Definition: Nested attribute-value paired term

A nested attribute-value paired term (NAV) is defined recursively:

i) A constant or a variable is a NAV,

ii) (a1/t1*⋯*an/tn) is a NAV, where t1,⋯,tn are NAVs, ai,⋯,an$\in$ATTR and for i, j ($i \neq j$), $ai \neq aj$.

A NAV defined by ii) is called proper.

Examples:

1) A record (NAV) with attributes of name, age and address:

    (name/{john}*age/{28}*address/{PA}),
    (name/{smith}*age/{26}*address/{NY}).

A record is defined as a combination by '*' of pairs of attributes and values by '/'. A value is specified by an attribute and can be located anywhere. According to the definition, a value is always a set, but the brackets of a single value (a singleton) are often omitted. Parentheses are also omitted except in cases of possible misinterpretation, such as the following:

    name/john*age/28*address/PA,
    name/smith*age/26*address/NY.

2) A row-nested record with name and hobby:

    name/john*hobby/{ski, tennis, baseball},
    name/smith*hobby/{music, ski}.

In a case of multiple values, the attribute value is expressed explicitly as a set. We consider that a record with set values has the same semantics as its decomposed ones. For example, the second record is equivalent to a set of

    'name/smith*hobby/music' and 'name/smith*hobby*ski'.

It means that row-nesting and row-unnesting operations do not change the meaning (details in next subsection). Note that it is important from the viewpoint of the user language, because a set of nested records does not generally have unique representation.

3) A column-nested record

    student/(name/(last/john*first/ω)*affiliation/icot*hobby/{ski, tennis}
        *school/(school_name/mita*school_address/tokyo)).

A nested record is expressed as a nest of attributes and a set value. 'first/ ω' means that the first name is unknown, and it shows the possibility of values attached.

We define some simple notations:

attrs(t) is a set of attributes included in a NAV t,
vars(t) is a set of variables included in t,
subterm(ai, t)=ti is defined when t=a1/t1*⋯*an/tn.

Further, for the sequence of attributes <b1, b2, ⋯, bn>, subterm(<b1, b2, ⋯, bn>, t) is recursively defined:

subterm(<b>, t)=subterm(b, t),
subterm(<b1, b2, ⋯, bn>, t)=subterm(<b2, ⋯, bn>, subterm(b1, t)).

When subterm (p, t) is a constant or a variable, p is called a terminating path of t. We simply use a notation p//<b1, ⋯, bn>=q when subterm(<b1, ⋯, bn>, p)=q.

Example:
Let t be a NAV of the above example 3).
attrs(t)
  ={student, name, last, first, affiliation, hobby, school, school_name, school_address}
A set of terminating paths of t is
    {<student, name, first>, <student, name, last>, <student, affiliation>,
        <student, school, school_name>, <student, school, school_address>},
t//<student, name, last>=brown.

We give some axioms explicitly for rewriting NAVs:

a/(t1*t2)=a/t1*a/t2,
t1*(t2*t3)=(t1*t2)*t3,
t1*t2=t2*t1,
t*t=t.

Lemma: A proper NAV can be transformed using the above axioms into the following form:

<a1, ⋯, an>/u*⋯*<b1, ⋯, bm>/v,
where <a1, ⋯, an>, ⋯, <b1, ⋯, bm> are terminating paths.

Proof: Obvious, because recursive definition of a NAV reserves a path form.

This transformed form is called a path form of a NAV.

Note that the above definition for a NAV is like a partially specified term (PST) in CIL [Mukai 87] if the domain of constants is restricted to D, since a NAV defined by **ii**) can be rewritten as {a1/t1,⋯ ,an/tn} because '*' is associative, commutative and idempotent, and {} used instead of ω.

3.2 Semantics

We define a partially tagged tree (PTT) and consider a set of PTTs as a domain of interpretation of NAVs. The same symbols are used for a set of attributes ATTR and a set of constants D.

Definition: Attribute string

A concatenation '.' of attributes is defined as follows:

$\langle\rangle.a=a.\langle\rangle=a,$

$a1.(a2.a3)=(a1.a2).a3.$

A concatenation, a1.⋯.an, is called an attribute string, which is denoted by ⟨a1,⋯,an⟩, and ATTR* is a set of attribute strings.

Definition: Tree domain

A subset T of ATTR* is called a tree domain iff it satisfies the following conditions:

If $a.b \in T$ then $a \in T$, and

For any $a \in T$, a set $\{b \mid a.b \in T\}$ is finite.

Clearly, every tree domain contains an empty attribute ⟨⟩.
An element, l, of a tree domain, T, is called a leaf of T
iff $\forall m \in ATTR\ l.m \in T \rightarrow m=\langle\rangle$. leaf(T) denotes a set of all leaves of T.

Example:

$T1=\{\langle\rangle, a, a.b, c\}$ and $T2=\{\langle\rangle, a, a.b, a.b.a, a.b.a.b, \cdots\}$ are tree domains.
$leaf(T1)=\{a.b, c\}$ and $leaf(T2)=\phi$.

Definition: Partially tagged tree (PTT)

Let T be a tree domain and f be a partial function from leaf(T) to D. A pair (T,f) is called a partially tagged tree (PTT). A function, f, is expressed as a set of pairs of a leaf and an element of D, which might be an empty set.

Example:

$(\{\langle\rangle\}, \phi)$ and $(\{\langle\rangle, a, a.b, c\}, \{(a.b, v), (c, v')\})$ are PTTs.

Definition: Merge of PTTs

Let t1=(T1,f1), t2=(T2,f2) be PTTs. A merge operation of t1 and t2 is denoted as t1+t2 and defined as $t1+t2=(T1 \cup T2, f1 \cup f2)$ only if $f1 \cup f2$ is a function.

We use the notation $\sum_{i=1}^{n} ti$ for t1+⋯+tn.

A set of PTTs is an associative, commutative, idempotent semigroup with a unit element $(\{\langle\rangle\}, \phi)$ with respect to the merge operation.
This merge operation is extended naturally on a set of PTTs:
For a set of PTTs s and s', $s+s'=\{ti+tj' \mid ti\in s,\ tj'\in s',\ ti+tj' \text{ is defined}\}$.

Example:
$(\{\langle\rangle, a, a.b, c\}, \{(a.b, v), (c, v')\})+(\{\langle\rangle, c, d\}, \{(d, u)\})$
$=(\{\langle\rangle, a, a.b, c, d\}, \{(a.b, v), (c, v'), (d, u)\})$
$(\{\langle\rangle, a, a.b, c\}, \{(a.b, v), (c, v')\})+(\{\langle\rangle, c, d\}, \{(c, u'), (d, u)\})$ is not defined
if $v'\neq u'$.

Useful notations:
for $a\in ATTR$ and PTT $t=(T, f)$,
$a.t=(a.T, a.f)=(\{x \mid \exists z, y\in T\ x.z=a.y\}, \{(a.l, v) \mid (l, v)\in f\})$.
This is also extended on a set of PTTs:
$a.\{t1, \cdots, tn\}=\{a.t1, \cdots, a.tn\}$.
Partial order is defined:
for PTT $t=(T, f)$ and $t'=(T', f')$, $t\leq t'$ iff $T\subseteq T'$ and $f\subseteq f'$.

Now we have prepared for semantics of NAVs. Let η be variable assignment from V to a set of PTTs. Assignment $I[\eta]$ from a NAV to a set of PTTs is defined recursively:

$$\begin{aligned} I[\eta](t) &= \{(\{\langle\rangle\}, \phi)\} && \text{if } t=\omega \\ &= \{(\{\langle\rangle\}, \{(\langle\rangle, c)\}) \mid c\in s\} && \text{if } t=s\in P(D) \\ &= \eta(x) && \text{if } t=x\in V \\ &= \sum_{i=1}^{n} ai.I[\eta](ti) && \text{if } t=a1/t1*\cdots*an/tn \end{aligned}$$

Example:
Consider a NAV $p1=a/c*b/\{d, e\}$, $p2=a/c*b/d$ and $p3=a/c*b/e$, and denote a simple PTT as $tx=(\{\langle\rangle\}, \{(\langle\rangle, x)\})$. Then
$I[\eta](p1)=\{a.tc\}+\{b.td, b.te\}=\{a.tc+b.td, a.tc+b.te\}$,
$I[\eta](p2)=\{a.tc\}+\{b.td\}=\{a.tc+b.td\}$,
$I[\eta](p3)=\{a.tc\}+\{b.te\}=\{a.tc+b.te\}$.
Clearly $I[\eta](p1)=I[\eta](p2)\cup I[\eta](p3)$.
When we define a program based on a NAV in Section 5, we give semantics of it such that even if p1 is divided into p2 and p3, they have the same meaning.

Example:
Consider $p=a/c*b/\omega$ and $q=a/c$.
$I[\eta](p)=a.tc+b.\{(\{\langle\rangle\}, \phi)\}=a.tc+\{(\{\langle\rangle, b\}, \phi\}$, and $I[\eta](q)=a.tc$.
Each meaning of p and q is different, however, they play a same role for a nested relation with attributes a and b.

4. UNIFICATION

As usual, we start with a definition of substitution:

Definition: Substitution
A substitution, θ, is a function from V to a set of NAVs, which is different from identity, only on a finite subset of V. For a NAV, t, application θ to t is written as $t\theta$. θ is also denoted by $\{x1/t1, \cdots, xn/tn\}$ where ti is substituted for xi.

Composition of substitutions, generality, renaming substitution and variant relation are defined just in the usual way [Lloyd 84]. We use a representative NAV of an equivalent class modulo renaming.

Usually, unification of p and q is to solve an equation p=q. However, a NAV makes its position different due to its partiality and its set value. Note the following points:

i) Whether unification of sets corresponds to their usual equality or their set operation such as intersection or union

ii) Whether 'occur check' should be done or not, that is, infinite trees are allowed as solutions or not

For the first point, we select the intersection of sets as their unification, because the intersection should correspond to database operations if we consider a set of NAVs as a relation or a table. We give a semantics to a NAV as a set of PTTs, each of which does not have a set value on the leaf, and if we consider a set of PTTs as a table, unification corresponds to a join or a selection operation of tables. Unification of NAVs corresponds to a merge operation of sets of PTTs.

For the second point, we restrict a solution to a set of finite restricted PTTs, although CIL allows a more general solution. From the viewpoint of the nested relation, the restrictions are not strict but natural, that is, such a record structure does not appear in the model. However, if more general record structures such as taxonomic information are handled, the restrictions should be weakened and a solution in an infinite tree domain must be considered. The problem is considered in Section 7.

Definition: Partial Order
Ordering of NAVs is recursively defined:

i) if $p, q \in P(D)$ and $p \subseteq q$ then $p \leq q$,

ii) if p and q are $\langle p1\rangle/v1 * \cdots * \langle pn\rangle/vn$ and $\langle p1\rangle/v1' * \cdots * \langle pn\rangle/vn'$, and $vi \leq vi'$ for $1 \leq i \leq n$, then $p \leq q$.

Definition: Compatibility

NAVs p and q are compatible iff

i) if $p, q \in P(D)$ then $p \cap q \neq \phi$,

ii) $p \in V$ or $q \in V$, or

iii) if p and q are proper (without loss of generality, let $p=t*t'$ and $q=u*u'$ where $t=a1/p1*\cdots*an/pn$, $t'=b1/p1'*\cdots*bm/pm'$, $u=a1/q1*\cdots*an/qn$ and $u'=c1/q1'*\cdots*ck/qk'$), then pi and qi for each $1 \leqq i \leqq n$ are compatible.

For compatible p and q, we define frontier(p, q):

i) if $p \in V$ then frontier(p, q) = {p, q}, and if $q \in V$ then frontier(p, q) = {p, q},

ii) if $p=a1/p1*\cdots*an/pn*b1/p1'*\cdots*bm/pm'$, $q=a1/q1*\cdots*an/qn*c1/q1'*\cdots*ck/qk'$),

then $\mathrm{frontier}(p,q)=\bigcup_{i=1}^{n}\mathrm{frontier}(pi,qi)$, and

iii) otherwise frontier(p, q) = ϕ.

Unification is obtaining a substitution θ such that $p\theta$ and $q\theta$ are compatible for given NAVs, p and q.

Definition: Environment

A set E of sets of NAVs is an environment iff

i) for any $w \in E$, there exists at least one variable $x \in w$,

ii) for any variable $x \in w \in E$ there does not exist $w'(\neq w)$ such that $x \in w'$, and

iii) for any $p, q \in w \in E$, there exists $w' \in E$ such that frontier(p, q) $\subseteq w'$.

For environments E and E', a 'transitive closure' of $E \cup E'$ is defined:

i) $E0=E \cup E'$.

ii) If $\exists x \in V\ x \in w \in Ei$, $x \in w' \in Ei$ and $w \neq w'$

then $Ei+1=(Ei-\{w, w'\}) \cup \{w \cup w'\} \cup \{\mathrm{frontier}(p,q) \mid p \in w, q \in w', \text{ where } \neg p, q \in V\}$.

iii) If $p, q \in w \in Ei$, $p, q \in P(D)$ and $p \cap q \neq \phi$, $Ei+1=(Ei-w) \cup \{(w-\{p,q\}) \cup (p \cap q)\}$.

If for some n, En+1 cannot be defined, then En is a transitive closure of E and E'.

Given compatible NAVs p and q (we assume that vars(p) and vars(q) are disjoint), an environment can be obtained by the following procedure:

i) If vars(p) or vars(q) is empty, then $E=\{\{x\} \mid x \in \mathrm{vars}(p) \cup \mathrm{vars}(q)\}$.

ii) If $p \in V$, then $E=(\{\{x\} \mid x \in \mathrm{vars}(p) \cup \mathrm{vars}(q)\}-\{p\}) \cup \{p, q\}$. If $q \in V$, then $E=(\{\{x\} \mid x \in \mathrm{vars}(p) \cup \mathrm{vars}(q)\}-\{q\}) \cup \{q, p\}$.

iii) If p and q are proper ($p=a1/p1*\cdots*an/pn*b1/p1'*\cdots*bm/pm'$, $q=a1/q1*\cdots*an/qn*c1/q1'*\cdots*ck/qk'$), then let Ei be an environment of pi and qi for $1 \leqq i \leqq n$, and

$E0=(\bigcup_{i=1}^{m}\mathrm{vars}(pi')) \cup (\bigcup_{i=1}^{k}\mathrm{vars}(qi'))$. Then E is a transitive closure of $\bigcup_{i=0}^{n} Ei$.

We denote a set of variables included in E by vars(E) and define ordering 'occur check':

Definition: Acyclic environment

Given an environment E, we define ordering on vars(E):

    if $x, t \in w \in E$, $x \in V$ and $y \in vars(t)$ then $x<y$.

This ordering on vars(E) is acyclic iff it does not contain the sequence $x<\cdots<x$ for any variable x. And E is called acyclic if the ordering on E is acyclic.

Definition: Consistent environment

An environment E is consistent iff for any $p, q \in w \in E$ p and q are compatible.

Definition: Unification

Compatible NAVs p and q are unifiable iff there exists a substitution θ such that

i) $p=\omega$ or $q=\omega$,

ii) if $p\theta, q\theta \in P(D)$ then $p\theta \cap q\theta \neq \phi$,

iii) if $p\theta = q\theta \in V$, or

iv) if p and q are respectively $a1/p1*\cdots*an/pn*b1/p1'*\cdots*bm/pm'$ and $a1/q1*\cdots*an/qn*c1/q1'*\cdots*ck/qk'$, then for each $1 \leqq i \leqq n$, $pi\theta$ and $qi\theta$ are unifiable.

Such a substitution is a unifier.

We define a unified NAV (denoted by $uni(p, q, \theta)$ as the result of unification of p and q by a unifier θ:

i) if $p=\omega$ then $uni(p, q, \theta)=q$, and if $q=\omega$ then $uni(p, q, \theta)=p$,

ii) if $p\theta, q\theta \in P(D)$ then $uni(p, q, \theta)=p\cap q$,

iii) if $p\theta, q\theta \in V$ then $uni(p, q, \theta)=p\theta$,

iv) if p and q are respectively $a1/p1*\cdots*an/pn*b1/p1'*\cdots*bm/pm'$ and $a1/q1*\cdots*an/qn*c1/q1'*\cdots*ck/qk'$,
then $uni(p, q, \theta)=$
$a1/uni(p1, q1, \theta)*\cdots*an/(pn, qn, \theta)*b1/p1'\theta*\cdots*bm/pm'\theta*c1/q1'\theta*\cdots*ck/qk'\theta$.

Theorem: Compatible NAVs p and q are unifiable iff there exists an acyclic and consistent environment E of p and q.

Proof: only-if-part:

Let θ be a unifier and $E'=\{\{x, t\} \mid x/t \in \theta\}$. Then a transitive closure of $E' \cup vars(p) \cup vars(q)$ is an acyclic and consistent environment.

if-part:

Let $E0=\{w \mid \exists x, y \in w \in E, x \neq y\}$ and $S0=\phi$. If $x, y \in w \in Ei$ and $x, y \in V$ then $Ei+1$ is an environment such that all y in $(Ei-\{w\}) \cup \{w-\{y\}\}$ is replaced by x, and $Si+1=Si \cup \{x/y\}$. Let En be an environment such that for any $w \in En$ there is only one variable in w. Let $E'0=En$, and $S'0=Sn$. If x is one of the minimal elements in ordering in $\{x \mid \{x, t1, \cdots, tn\} \in E'i, x \in V\}$ and $w=\{x, t1, \cdots, tk\} \in E'i$, then as the above procedure reserves consistency of E. t1,…,tn are compatible and a unified NAV of ti and tj is well defined with identity substitution. If t is a unified NAV of t1, …, tn, then let E'i+1 be an environment such that all occurrences of x in E'i-{w} are replaced by t, and S'i+1=S'i∪{x/t}. Applying this procedure repeatedly, we can obtain E'm=ϕ and S'm.

Definition: Lossless unifier
As a necessary condition of unification is compatibility, in the above procedure of the theorem, we can also select t'(≦t) instead of t as a substituted NAV. When t'=t, unification is called lossless.

Corollary. There exists a unique lossless mgu if two NAVs are unified.

Proof: The above procedure in the theorem generates such a unifier. It is easy to verify the conditions.

Next we consider the semantics of unification of NAVs:
Corollary: if θ is a unifier of p and q, and I[η] is assignment then I[η](pθ)∪I[η](qθ)=I[η](uni(p,q,θ)).

Proof: Clearly, in cases of p,q∈P(D), and p∈V or q∈V. And also in the case of proper p and q, recursive definitions of unification and assignment prove the corollary.

5. PROGRAM

We define a program based on NAVs and its semantics:

Definition: Program
A pair (p,B) consisting of a NAV p and a set of NAVs B is a program clause. (p,{p1, … ,pn}) is written as p←p1,…,pn. And (p,ϕ) is called a unit clause and written simply as p. A ground unit clause (a NAV without variables) is called a fact. A program is a set of program clauses.

Example: A program for parent and ancestor relationship:

```
parent/{jack,betty}*child/{john,cathy}.
parent/{jack,nancy}*child/{mary}.
parent/{john,kate}*child/{bob}.
parent/{george,mary}*child/ω.
parent/{james,ann}*child/{betty,paul}.
ancestor/X*descendant/Y←parent/X*child/Y.
ancestor/X*descendant/Y←parent/X*child/Z,ancestor/Z*descendant/Y.
```

Definition: Model

A non-empty set M of PTTs is a model iff for any assignment η and p←p1, …, pn∈P, if I[η](pi)=si⊊M for each 1≦i≦n then I[η](p)⊆M.

Example: A model of the program

Consider the subset of the above program:

```
        parent/{betty}*child/{john,cathy}.
        parent/{james,ann}*child/{betty}.
        ancestor/X*descendant/Y←parent/X*child/Y.
        ancestor/X*descendant/Y←parent/X*child/Z,ancestor/Z*descendant/Y.
```

Let a PTT ({<>,parent,child},{(parent,u),(child,v)}) be denoted by pc(u,v) and ({<>,ancestor,descendant},{(ancestor,u),(descendant,v)}) by ad(u,v). A model of the program includes the following:

```
        {pc(betty,john),pc(betty,cathy),pc(james,betty),pc(ann,betty),
          ad(james,john),ad(james,cathy),ad(ann,john),ad(ann,cathy)}.
```

Note that, according to the above definition, programs {a/{c,d}} and {a/{c},a/{d}} have the same model such as {({<>,a},{(a,c)}),({<>,a},{(a,d)})}. That is, a row-nest and a row-unnest operations do not change the meaning of the program.

Lemma: The intersection of models of a program P is also a model of P.

Proof: Clear from the above definition.

Theorem: There is the least model for a program P.

Proof: The least model is intersection of all models of P.

Definition: Logical consequence

A NAV p is a logical consequence of a program P iff a model of P is also a model of p.

Definition: Goal

A goal, Q (={q1,…,qn}), for a program, P, is a set of NAVs and written as ←q1,…,qn. An answer for a goal is a substitution θ such that all of q1θ,…, qnθ are logical consequences of the program P. Then θ is called an answer substitution.

Example: Consider the following goals (queries) of the above program:

```
        ←parent/X*child/john.
        ←parent/X*child/john,parent/X*child/mary.
        ←ancestor/X*descendant/bob.
```

For the first query X = {jack}, {betty} and {jack,betty}, for the second X =

{jack}, and for the third X = a non-empty element of a power set of {john, kate, jack, betty, james, ann} are answer substitutions.

Example: Column-nested case:

```
        student/(name/(last/john*first/ω)*affiliation/icot*hobby/{ski,tennis}
            *school/(school_name/mita*school_address/tokyo)).
        ←student/(name/X*school/Y).
```

An answer is X=last/john*first/ω and Y=school_name/mita*school_address/tokyo.

Example: Consider the following program (unnormalized relation):

```
        a/c1*b/{c3,c4}.
        a/c2*b/c3.
        a/c2*b/c4.
```

For a goal ←a/{c1,c2}*b/X, X = {c3}, {c4} and {c3,c4} are answer substitutions.

For a procedural semantics we must define some notions:

Definition: Rest goal
Let t0 and t0' be unifiable by a unifier θ. If for some termination path p, t0//p=S, uni(t0,t0',θ)//p=S0 and $S \supset S0$, t0 is called reduced in p by θ. If t0=p/S*t1 and t0'=p/S'*t1' then p/(S-S0)*t1 and p/(S'-S0)*t1' are called the rests. If t0 is used as a goal, p/(S-S0)*t1 is called the rest goal. Generally if p1/S1*⋯*pn/Sn is reduced in p1/S1'*⋯*pn/Sn'*t, the rest goal is p1/(S1-S1')*⋯*pn/(Sn-Sn').

Definition: Marked substitution
A pair x:u of a variable x and a NAV u is called a marked subterm. Instead of substitution of u for x, substitution of x:u for x is called marked substitution. Let t be a variable x and θ={x/x:u}, then tθ = x:u. If t does not include 'x' or 'x:', θ is identity. Let t be x:u and θ={x:u/x:v}, then tθ = x:uni(u,v,σ) if there exists $\sigma \subseteq \theta$. If u and v cannot be unified, the substitution is failed. If u,v∈P(D) and $u \cap v \neq \phi$, x:v is changed into x:($u \cap v$).

Once we can get substitution {x/v}, the corresponding marked substitution θ ={x/x:v} and the new marked substitution θ'={x:u/x:v} which includes a unifier between v and u must be generated. Let t=p/x*t' be a NAV and θ={x/x:u}. Then tθ=p/x:u*(t'θ). If another binding substitution σ is applied to the rest goal t' and a variable x in t' is changed to v, then a unifier σ between v and u is applied to goals and x:(uσ) has been replaced by x:uni(v,u,σ).

'x:' is not a variable but only a mark where there was x, and it cannot be substituted or unified. Marked substitution is used to control a history of its subterm uniformly. In the procedural semantics, marked substitution is used instead of usual substitution.

Definition: Unmarked rest goal
If p1/S1*…*pn/Sn is reduced in p1/S1'*…*pn/Sn'*t and there are marks in p1,…, pi, the unmarked rest goal is p1/S1'*…pi/Si'*pi+1/(Si+1−Si+1')*…*pn/(Sn−Sn'). This rest goal assures that under one mark there is a unique subterm even if the mark appears in some goals.

Definition: SLD-resolution
Let P be a program and G be a goal. We assume that a set of variables included in each program clause is disjoint from another. Let G0=G and E0 be an empty environment. Let Gi={q1,…,qn}. If qj∈Gi and pi for some pi←pi1,…,pik∈P are unified by unifier σ, then

$$Gi+1=\{q1\theta\,\theta',\cdots,qj-1\theta\,\theta',pi1\theta,\cdots,pik\theta,qj'\theta,qj+1\theta\,\theta',\cdots,qn\theta\,\theta'\},$$

where θ is a marked substitution corresponding to σ, θ' is the new generated marked substitution and qj'θ is the unmarked rest goal of qj generated by θ. If Ei' and Ei" are environments corresponding to θ and θ' respectively, Ei+1 is a transitive closure of Ei∪Ei'∪Ei". The sequence of (G0,E0)⇒… ⇒(Gi,Ei)⇒ … is SLD-resolution of P and G.

Example: Consider the following program (unnormalized relation):

    a/c1*b/{c3,c4}.
    a/c2*b/c3.
    a/c2*b/c4.

For a goal ←a/{c1,c2}*b/X. we can obtain X=c3 and X=c4, which are the same as X={c3,c4}, as the answer substitution.

Theorem: Given a program P and a goal G, if there exists n such that Gn=ϕ in the sequence of SLD-resolution, then the corresponding substitution θ with En is an answer substitution such that qiθ for $1 \leq i \leq n$ is a logical consequence of P.

6. CRL AND NESTED RELATIONS

A NAV is an extension of a first order predicate (term) as follows:

$$p(t1,\cdots,tn) \Rightarrow p/(p\$1/t1*\cdots*p\$n/tn).$$

On the right side, p is an attribute and p\$1, … ,p\$n are attributes corresponding to the position of arguments of p. A set of usual predicates is isomorphic to a set of NAVs, whose domain is restricted to a set of singletons.

If we fix a set of top-level attributes and use them as relation names, then we can relate a NAV to a nested record in a relation whose name is the same as the top-level attribute. Like Prolog, a CRL program is considered as a database with multiple values. Any database can be transformed and divided into an extensional database (EDB) and an intensional one (IDB) like a Prolog database. EDB is a usual nested database. Its unique representation of EDB cannot be guaranteed generally; however, given EDB, its meaning is unique independently of its representation.

If we see CRL as a programming language, then its evaluation mechanism may be based on modified SLD-resolution of the above. The mechanism returns answers according to the representation of EDB, and a set of answers has a unique meaning independent of the representation. However, from the viewpoint of the deductive database, the evaluation mechanism should be reconsidered for termination and efficiency of query processing. Many optimization strategies have been proposed for a deductive database for a relational database [Bancilhon 86b], and most of them are resolved into a procedure using relational algebra. However, extended relational algebra for nested relations depends on their nested form.

For example, the following relation

DB1: a/c1*b/c4, a/c2*b/c4, a/c2*b/c5

can be transformed into

DB2: a/{c1,c2}*b/c4, a/c2*b/c5, or

DB3: a/c1*b/c4, a/c2*b/{c4,c5}.

If we consider a goal ←a/{c1,c2}*b/X, then the above strategy returns X=c4 for any of DB1, DB2 or DB3. However, in a case of a selection such as a⊇{c1,c1} only DB2 returns a correct answer.

If a base relation consists of nested records, all of which are nested in uniform sequence, then we should transform a goal for a base relation into the corresponding form. In the above example, the goal can be transformed into ←a/{c1}*b/X, a/{c2}*b/X for DB3. If a base relation is not nested uniformly, then a goal with a set value should be decomposed into a set of goals with a set of goals with a singleton. For the following database

a/c1*b/{c3,c4}, a/{c1,c2}*b/c5,

a goal ←a/X*b/{c4,c5} should be decomposed into ←a/X*b/c4, a/X*b/c5. If metadata of a base relation has information about nested sequences, then we can transform a given goal.

There are no problems in non recursive queries, because transformation of a goal corresponding to base relations and bottom-up evaluation assures the result. We have not yet investigated the strategy for recursive queries; however, we consider that a combination of strategies for Prolog and transformation of goals will give an answer.

The CRL database should be considered also from the viewpoint of the universal relation. First, if the top-level attribute (relation name) is omitted, then it is a universal relation. Second, a subclass of a nested relation satisfies universal relation schema assumption [Abiteboul 84]. The relation between the CRL database and universal relation has not been discussed enough yet. When we consider a set of NAVs as a database, there is another problem in duplication of information, which causes an update anomaly. A simple solution is to enforce a constraint as key attributes to a base relation.

7. EXTENSIONS

This section considers some extensions to CRL, such as deep nesting, negation and type hierarchy.

For deep row-nesting, we consider a constant domain as $D \cup \{\omega\}$, add a function symbol '+', and generalize a definition of a NAV as follows:

i) A constant or a variable is a NAV,

ii) (a1/t1*$\cdots$*an/tn) is a NAV, where ai,$\cdots$,an$\in$ATTR, t1,$\cdots$,tn are NAVs and for i, j ($i \neq j$), ai$\neq$aj.

iii) If t1,$\cdots$,tn are NAVs, (t1+$\cdots$+tn) is a NAV.

A NAV defined is called single when only by applications of i) and ii).

We add new axioms for NAVs:

$$a/(t1+t2)=a/t1+a/t2,$$
$$t1+(t2+t3)=(t1+t2)+t3,$$
$$t1+t2=t2+t1,$$
$$t+t=t,$$
$$t+\omega=t,$$
$$t1*(t2+t3)=(t1*t2)+(t1*t3).$$

We write {a1,$\cdots$,an} for (a1+$\cdots$+an) because '+' is ACI. A NAV defined in Section 3 should be rewritten as c1+$\cdots$+cn instead of an element s={c1,$\cdots$,cn} of P(D).

Definition: Normal form

A NAV t is in a normal form iff t=t1+$\cdots$+tn where t1,$\cdots$,tn are single NAVs.

Theorem: A NAV can be transformed into a normal form.

We give semantics to a normal form of a NAV. Let η be a variable assignment from V to a set of PTTs. Then assignment $I[\eta]$ is defined as follows:

$$\begin{aligned} I[\eta](p) &= \{(\langle\rangle, \phi)\} && \text{if } p=\omega, \\ &= \{(\langle\rangle, \{(\langle\rangle, c)\})\} && \text{if } p=c\in D, \\ &= \eta(x) && \text{if } p=x\in V, \end{aligned}$$

n

$= \{\sum_{i=1} ai.I[\eta](pi)\}$ if p is a NAV (a1/p1*⋯*an/pn),

$= \bigcup_{i=1}^{n} I[\eta](pi)$ if a normal form of p is (p1+⋯+pn).

In Section 3, a NAV was restricted so that its set value does not contain variables. We can consider only a simple case; however, in the generalized NAV, there might be variables and there are problems in the scope of variables. That is, row-unnesting causes a variable over multiple records in a database, and it works as a null value with equality constraint. It also needs unification under a distributive law.

Another extension is to introduce negative information. Like Prolog, a program clause of CRL is extended to a set of literals (a positive NAV or a negative NAV). The evaluation of a ground negative NAV is the same as SLDNF. The same technique as [Warren 87] for floundering can be applied to CRL.

Attribute-value pair representation is not only for formalism of nested relation, but also for various kinds of structured data. [Maier 86], [Bancilhon 86], [Abiteboul 86] and other recent research deal with formalism of objects, complex objects and so on. We would like to extend a 'term' defined by a function from a tree domain of attributes (labels) to a domain of constants (or types or objects), for representing more general structured data such as type hierarchy and complex objects.

8. RELATED RESEARCH

CIL (Complex Indeterminates Language) is a logic programming language motivated by situation semantics, which aims at representing uniformly various fields in linguistics [Mukai 87]. It is also the implementation language for a discourse understanding system called DUALS. It focuses on record structures appearing in the domain and represents them in attribute-value pairs. From the viewpoint of the nested relation, CIL handles a general column-nesting with infinite trees but not row-nesting. Section 3 described the relation between CIL and CRL. Although they are in different domains, they are common in handling more general record structures.

Ait-Kaci proposed the ϕ-term for type inheritance, which is an extension of first order terms by type structure represented in attribute-type pairs with an equality constraint called a tag. As notational convenience, he considered the ε-term for a set of ϕ-terms [Ait-Kaci 84,86]. The ϕ-term is very flexible and its type can be interpreted in various ways from the viewpoint of the database. Bancilhon proposed a calculus for complex objects, where a type corresponds to a complex object with set and tuple constructors, which are linked by a part-of relation [Bancilhon 86]. Maier proposed a logic for objects, where a class and an object correspond to a type and a tag respectively [Maier 86]. The ϕ-type also can be applied to a nested relation if we consider a type as a relation (which might be called a relation object). That is, a nested relation is defined as a pair of a tree domain of attributes, and a partial function from the tree domain to a set of relation objects. However, as the object is different from a usual relation, we must introduce a set constructor as in [Bancilhon 86].

Beeri et al proposed LDL1 (Logic Database Language), which introduces extensional and intensional definitions of a set, that is, set enumeration and set grouping into Prolog for resolving 'mismatch' between tuple-at-a-time in Prolog and set-at-a-time in a database [Beeri 87]. In LDL1.5, deep row-nesting can be expressed by set grouping and transformed into LDL1. However, a nested relation is not considered as a base relation but as a derived relation. In the framework of Prolog, it does not seem to be appropriate for formalizing nested relations, from the viewpoints of representation of base nested relations and a user language independent of nested sequence. Kuper also proposed LPS (Logic Programming with Sets) with the explicit motivation of integration of a nested relational model and a deductive database [Kuper 87]; however, it has been proven to be a subclass of LDL1 in [Beeri 87].

Another significant research report was that of W.C.Rounds et al, which considered a complete logical calculus for record structures for linguistic information. The calculus is not related to a database, but it seems very valuable in introducing logical meaning such as disjunction and conjunction between occurrences of row-nesting.

The research on integration of a nested relation and a logic programming language, that is, a deductive database based on nested relations, is a very recent trend with many possibilities. From the viewpoint of knowledge representation, it relates structured data such as objects, complex objects, class taxonomy and other structured objects in a semantic model.

From the viewpoint of the universal relation, we should consider the following points. First, a logic programming language based on attribute-value pairs would be appropriate for a query language because it admits partial specification and is more independent of the logical structure in a database. Secondly, a nested relation itself relates a universal relation, like Verso model and a universal relation schema assumption [Abiteboul 84]. A nested database based on CRL should be considered in the point.

9. CONCLUDING REMARKS

This paper defined an attribute-value paired term called a NAV, and a logic programming language based on NAVs, called CRL. A NAV is an extension of a first order term and is appropriate for representation of more general record structures, especially nested relations. It is also suitable for a user language for a database. A CRL program can be considered as a nested database consiting of an extensional database and an intensional database, like Prolog database.

The main characteristics of CRL are:

i) The CRL program is data-independent, that is, even if database schema is modified, the program does not need to be modified. This makes it easy for a user to write inference rules and a query.

ii) The CRL database is independent of row-nest and row-unnest operations, that is, it returns the same answer even if its nesting sequence is different. From the viewpoint of the user language, it is difficult to enforce nested sequences for a normal form of nested relations to a user query.

iii) If a relation is uniformly nested, then CRL processes a user query more efficiently than Prolog, because nested records and rules based on them reduce the size of the database, and the number of database operations to a base relation and evaluation paths will become smaller.

iv) A user can specify only necessary information with a base relation name as a NAV in his query. It helps him to write a query or a program.

We have already implemented the prototype system of KAPPA, and been designing and planning a new system based on the prototype. CRL and its extended version is intended to be one of the knowledge representation languages in the system. There remain many problems in CRL for its extensions, which also occur in usual nested relations. Our approach will help clarify them.

ACKNOWLEDGMENTS

I would like to thank Dr. Toshio Yokoi, Dr. Shunichi Uchida and Dr. Kuniaki Mukai for useful discussion and suggestion. I must thank to Mr. Minoru Kiyama for invaluable comments. I also wish to express my gratitude to members of KAPPA project.

REFERENCES

[Abiteboul 84] Abiteboul, S. and Bidoit, N., "Non First Normal Form Relations: An Algebra Allowing Data Restructuring", INRIA, TR-347, 1986

[Abiteboul 86] Abiteboul, S. and Hull, R. "Restructuring of Complex Objects and Office Forms", ICDT'86, LNCS-243, pp.54-72, 1986

[Ait-Kaci 84] Ait-Kaci, H., "A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures", Dissertation, Univ. of Pennsylvania, 1984

[Ait-Kaci 86] Ait-Kaci, H. and Nasr, R., "LOGIN: a Logic Programming Language with Built-in Inheritance", J. Logic Programming, vol.3, pp.185-215, 1986

[Bancilhon 86a] Bancilhon, F. and Khoshafian, S., "A Calculus for Complex Objects", PODS'86, pp.53-59, 1986

[Bancilhon 86b] Bancilhon, F. and Ramakrishnan, R., "An Amateur's Introduction to Recursive Query Processing Strategies", SIGMOD'86, pp.16-52, 1986

[Beeri 87] Beeri, C., Naqvi, S., Ramakrishnan, R., Shmuelli, O. and Tsur, S., "Sets and Negation in a Logic Database Language (LDL1)", PODS'87, pp.21-37, 1987

[Fischer 83] Fischer, P.C. and Thomas, S., "Operations for Non-First-Normal Form Relations", COMPSAC'83, pp.464-475, 1983

[Gallaire 78] Gallaire, H. and Minker, J. (ed), Logic and Data Bases, Plenum, 1978

[Gallaire 84] Gallaire, H., Minker, J. and Nicolas, J.-M., "Logic and Databases: a Deductive Approach", Computing Surveys, vol.16, no.2, pp.153-185, 1984

[Kambayashi 83] Kambayashi, Y., Tanaka, K. and Takeda, K., "Synthesis of Unnormalized Relations Incorporating More Meaning", Int.J.Information Science, vol.29, pp.201-247, 1983

[Kuper 87] Kuper, G.M., "Logic Programming with Sets", PODS'87, pp.11-20, 1987

[Lloyd 84] Lloyd, J.W., Foundations of Logic Programming, Springer, 1984

[Maier 86] Maier, D., "A Logic for Objects", Proc. of the Workshop on Foundation of Deductive Database and Logic Programming, pp.6-26, 1986

[Makinouchi 77] Makinouchi, A., "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", VLDB'77, pp.447-453, Tokyo, 1977

[Mukai 87] Mukai, M., "Anadic Tuples in Prolog", ICOT, TR-239, 1987

[Pistor 86] Pistor, P. and Traunmueller, R., "A Database Language for Sets, Lists and Tables", Inform. Systems, vol.11, no.4, pp.323-336, 1986

[Reiter 84] Reiter, R., "Toward a Logical Reconstruction of Relational Database Theory", in Conceptual Modeling, ed. by Brodie, M.L. Mylopoulos, J. and Schmidt, J.W., pp.191-238, Springer, 1984

[Rounds 86] Rounds, W.C. and Kasper, R., "A Complete Logical Calculus for Record Structures Representing Linguistic Information", 1986

[Schek 82] Schek, H.-J. and Pistor, P., "Data Structures for an Integrated Data

Base Management and Information Retrieval System", VLDB'82, pp.197-207, Mexico City, 1982

[Scholl 87] Scholl, M.H. and Schek, H.-J. (ed.), Theory and Applications for Nested Relations and Complex Objects, Workshop Material, Darmstadt, April 6-8, 1987

[Wallace 87] Wallace, M, "Negation by Constraints: a Sound and Efficient Implementation of Negation in Deductive Databases", SLP'87, pp.253-263, 1987

[Yokota 87] Yokota, K., "Knowledge Base Management System KAPPA", 5th Symposium on FGCS, June.9-10, 1987 (in Japanese)