Nested Relation Based Database Knowledge Representation

Qiming Chen

Computer Science Dept.
University of California
Los Angeles, USA
qiming@cs.ucla.edu

Yahiko Kambayashi

Computer Science Dept.
University of Kyoto
Kyoto, Japan
yahiko@kuis.kyoto-u.ac.jp

Abstract

The previous approaches of extending logic programming for handling complex objects introduced new LP notions but without changing the underlying set manipulations. As a result, they were insufficient for handling nested relations and could not consistently maintain the canonical semantics of logic programming. In order to remedy this problem we propose to develop algebraic tools at set manipulation level, such as partial inclusion and ungrouping, for providing a canonical formal framework of nested relation based knowledge representation.

1. Introduction

Knowledge representation admits not only declarative semantics, but also imperative semantics. Nested relation based knowledge representation should provide not only the means of specifying relevant facts, but also the means of generating and reconstructing nested relations. Unlike manipulating normalized relations where a set of standard relational operations are available, manipulating nested relations with few standard operations such as intersection-join is far more restricted. Instead rule-based inference potentially can provide richer expressive power than nested relation algebra in knowledge representation.

Furthermore, any knowledge representation formalism should be based on a consistent semantics framework. The foundation of PROLOG based knowledge representation is Logic Programming (LP) theory based on First-Order Logic(FOL) [LLO83], which can be used to deal with deductive databases [GAL84] [REI84], but has many limitations in handling complex structures. In order to support complex object reasoning many LP extentions have been proposed [ZAN85] [BEE86] [TSU86] [KUP87] [GAR87] [MAI86] [CHEN86] [APT86] [HULL89] [ABI89] [TAKE89] [KAM89]. However, they could not well meet the following key requirements for nested relation reasoning.

a) dealing with hierarchically nested tuples and relations (sets of tuples),
b) allowing nested relations to be unnested and uniformly handled at any level of their configuration hierarchies, and
c) handling named attributes to relax the constraint of *fixed position of arguments* in LP systems.

The intuitive reason causing the above limitations is that these systems generally treat complex objects as nested function terms to preserve the traditional FOL framework, and therefore fail to provide a unified framework for manipulating nested relations at arbitrary levels. Under such a philosophy, the top-level relations are treated as *predicates* which can be included in the Herbrand base and interpretations of logic programs, but the nested ones are treated as *function terms*, which cannot stand alone in program interpretations [TSU86]. This problem restricts the deductive retrieval at arbitrary levels of the nested relation. As a result many of the above systems cannot fully support mutually nested tuples and sets. For example, LPS [KUP87] is limited to one level set, [ZAN85] allows only nested tuples rather than nested sets. LDL1 [BEE86] allows nested sets but it does not allow the set elements to be tuples.

The more fundamental reason causing the above limitations is that the previous LP extentions still have their notions set on *single-level setwise manipulations*. As indicated by [TSU86] [ABI89], the inadequacy of applying such manipulations to nested structures may cause the inconsistency of some basic LP theorems. For example, as pointed out in [BEE86], with complex structures, some logic programs have no model; when a program does have a model, it does not necessarily have a unique least model; and that the intersection of models may not be a model. These facts imply the mismatch between the conventional set theory and LP notions involving hierarchical structures.

We shall introduce a two level view to LP model theory : at the higher level the basic LP notions are described, at the lower level the fundamental set operations are provided to underlie LP notions at the higher level. Accordingly we can call that higher level as the **LP theory level**, and the lower level as the **set theory level**. To develop extended LP systems for supporting nested relations, both levels are concerned.

The previous LP extentions introduced new LP notions to the LP theory level without changing the fundamental notions used in the set theory level. Since many notions in LP model theory are based on *single-level setwise manipulations*, and single-level setwise manipulations are incapable of dealing with structures, in applying them directly to nested relations two prob-

lems may arise : first, the system does not have full expressive power for handling mutually nested tuples and sets and for dealing with nesting/unnesting; second, the system is not canonical since certain typical LP theorems such as the uniqueness of least model, do not hold.

In contradistinction to the above we propose to **extend the fundamental notions at set theory level to underlie the notions at LP theory level.** The framework built in this way meets the key requirements mentioned above, is rather *canonical* in maintaining the major LP notions, and suitable for handling nested relations. For instance, we still preserve the existence and uniqueness of least model of a LP program for handling nested relations, but our model comparison is based on a newly extended notion of set inclusion. Other typical LP notions are also maintained but based on extended set manipulation notions. Such a treatment also makes the proposed framework compatible with first order LP as it covers the latter as its special case. In addition, the proposed approach enable us to handle named attributes to relax the constraint on fixed position of arguments in LP formulas.

This work extends our previous research shown in [CHEN88] and [KAM89]. We shall describe our framework in terms of the LP language with high-order constructs named HILOG-R, which is extended from HILOG presented in [CHEN89]. The rest of this paper describes how to represent nested relations and their manipulations in terms of logic formulas and rules, followed by the semantic framework underlying the proposed nested relation based knowledge representation.

2. Nested Relation based Knowledge Representation

Our first step consists in turning nested relations to logic formulas with high-order constructs. Symbols utilized in our framework include *constants, variables, types, attribute names, fillers, logical connectives* and *comparison operators*. We identify two basic kinds of structure types : the *tuple-type* with a list of attribute types, and the *set-type* with a unique member type. They may be mutually nested to form more complex types. To be clear throughout this paper set-types are denoted by capital symbols. The existence of a set of atomic types and the existence of an infinite set of variables for each type are assumed, the fixed-position constraint of attributes is relaxed, and specifying an attribute and a type by the same name is allowed. Types are defined as follows.

[Type] Types are defined recursively by the following :

a) An atomic type.
b) A **tuple-type** $\mathbf{p}(s_1{:}p_1,..., s_n{:}p_n)$ where $p_1,..., p_n$ are types and $s_1,..., s_n$ are attribute names, which can be abbreviated as **p**.
c) A **set-type** $\mathbf{R}\{p\}$ where p is a type, which can be abbreviated as **R**.

For example, a tuple type **prof** with an attribute of set-type, is specified as

prof(*name*:**name**, *fields*:**SUBJECTS**{**subject**}).

A nested relation schema **PROFS** is specified by the following type :

PROFS{ **prof**(*name*:**name**, *fields*:**SUBJECTS**{**subject**})}.

A type **project** representing a M:N relationship is specified by the following, where each project involves multiple professors and students.

project(*title*:**contract#**,
researchers:**PROFS**{
prof(*name*:**name**, *fields*:**SUBJECTS**{**subject**})},
assistants:**STUDENTS**{
student(*name*:**name**, *major*:**subject**)}).

The instances and variables of types are expressed by terms as defined below.

[Term] The set of terms of type **p**, T(p), is defined recursively by the following :

a) For a constant value c of atomic type **p**, $c \in T(p)$.
b) For a variable v of type **p**, $v \in T(p)$.
c) For a tuple-type $\mathbf{p}(s_1{:}p_1,..,s_n{:}p_n)$, $\forall i \in \{1,..,n\}\ t_i \in T(p_i)$ $\rightarrow \mathbf{p}(s_1{:}t_1,.., s_n{:}t_n) \in T(p)$ (**tuple-term**).
d) For a set-type $\mathbf{R}\{p\}$, $\forall i \in \{1,..,n\}\ t_i \in T(p)$ $\rightarrow \mathbf{R}\{t_1,..,t_n\} \in T(R)$ (**set-term**).
The special notation $\mathbf{R}\{x\}$, where x is a variable or a term containing variables, denotes a set-term of arbitrary cardinality.

For example, an instance of the type **project** shown above is expressed as :

project(*title*:IBM89,
researchers:**PROFS**{
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB}),
prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB})},
assistants:**STUDENTS**{
student(*s#*:s102, *name*:Linda, *major*:DB)}).

A ground term is one that is free of variables. Two ground terms t_1 and t_2 are equal, denoted as $t_1 = t_2$, iff they are the same atomic value, or they are set-terms with the same set-type and pairwise equal elements, or they are tuple-terms with the same tuple-type and equal elements filling each attribute.

Strong typing and *Unique Name Assumption (UNA) on types* are two key syntactical constraints on terms. By strong typing we mean that a tuple-term or a set-term must be tagged explicitly with a type symbol such as $\mathbf{p}(s{:}t)$ or $\mathbf{R}\{t\}$. On the contrary, terms like {{a,b}} and $\mathbf{R}\{\{a,b\}\}$ are illegal since they are not strongly typed, $\mathbf{p}(s{:}\mathbf{p}(s_1{:}a, s_2{:}b))$ is illegal since it violates UNA on types. In fact, these constraints emphasize the *context* of object modeling, which are not unnatural for relational databases since a tuple must belong to a particular relation, and all the horizontal fragments of a relation logically belong to the relation. Under these constraints, all the subsets of a same type can be grouped together and thus two sets **SUBJECTS**{AI, DB, KB} and **SUBJECTS**{OS, PL} are viewed as parts of the set of type **SUBJECTS**. From this sense, **SUBJECTS**{x} can be substituted by the instance set of type **SUBJECTS** of arbitrary cardinality, and **SUBJECTS**{t}, where t is a non-variable term, can be interpreted as "t is a member of the set of type **SUBJECTS** ".

In order to link our framework to logic it is necessary to introduce the notion of **formula**. We shall define a *top-level* structure, namely, top-level set-term or tuple-term, or a comparison operator with arguments, as an atomic formula, or simply an *atom*. Such a treatment follows two considerations : to deal with a nested relation as a hierarchically structured and integrated object, and to allow a relation (possibly nested) which is a part of one or more other nested relations to be handled individually. The conjunction of two formulas is also a formula. With the notion of formula, we can syntactically define a HILOG-R rule as **head** ← **body**, where the **body** is a formula and the **head** is a positive atomic formula. A fact is a ground rule without a body, denoted as **head**← (or simply as **head**). A HILOG-R program is a finite set of valid rules. A query is a rule without a head denoted as **?body**. These notions provide us the means of manipulating nested relations, representing knowledge associated with nested relations, and deriving new knowledge from the existing knowledge.

Some HILOG-R example programs are given below showing the reasoning with nested relations which essentially consists in the reconstruction of new nested relations from existing ones. In order to demonstrate this point we shall use HILOG-R logic programs to realize three kinds of join operations, namely, the 1-1 join where both join attributes are of atomic types, the 1-N join where one join attribute is of atomic type and the other is of set type, the M-N join where both join attributes are of set types. In the figures illustrating these examples, a set type is depicted by an elliptic symbol with a unique member type, and a tuple type is depicted by a rectangular symbol with several attribute types.

An HILOG-R Program for 1-1 Join

As shown in Figure 1, this program is based on the relation schemes

PROFS{**prof**(*name*:**name**, *fields*:**SUBJECTS**{**subject**})} and
ASSIGNMENTS{**assignment**(*name*:**name**,
projects:**CONTRACTS**{**contract**})}

to generate the nested relation **PROFS_PROJECT** with the following scheme

PROFS_PROJECT{**prof_project**(*name*:**name**,
fields:**SUBJECTS**{**subject**}, *projects*:**CONTRACTS**{**contract**})}.

It is the rule-based representation of join on attributes *name* of atomic type in both relations.

{ ***rule*** (Program P_1)

PROFS_PROJECT{**prof_project**(*name*:x, *fields*:**SUBJECTS**{y},
projects:**CONTRACTS**{z})} ←
PROFS{**prof**(*name*:x, *fields*:**SUBJECTS**{y})} &
ASSIGNMENTS{**assignment**(*name*:x, *projects*:**CONTRACTS**{z})}.

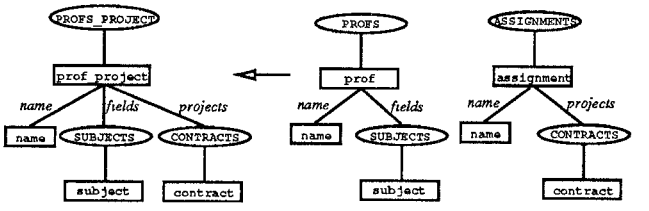
facts** (nested relations)

PROFS{
prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}),
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

ASSIGNMENTS{
assignment(*name*:Smith, *projects*:**CONTRACTS**{IBM89, NEC90}),
assignment(*name*:Parker, *projects*:**CONTRACTS**{NASA90,IBM89}),
assignment(*name*:John, *projects*:**CONTRACTS**{NEC90})}.
}

As a result of executing P_1 the following nested relation is derived.

PROFS_PROJECT{
prof_project(*name*:Parker, *fields*:**SUBJECTS**{DB, KB},
projects:**CONTRACTS**{NASA90, IBM89}),
prof_project(*name*:Smith, *fields*:**SUBJECTS**{AI, KB},
projects:**CONTRACTS**{IBM89, NEC90})}.



PROFS_PROJECT

name	*fields*	*projects*
Parker	DB KB	NASA90 IBM89
Smith	AI KB	IBM89 NEC90

PROFS

name	*fields*
Parker	DB KB
Smith	AI KB

ASSIGNMENTS

name	*projects*
Parker	NASA90 IBM89
Smith	IBM89 NEC90
John	NEC90

Figure 1: An 1-1 Join Example

An HILOG-R Program for 1-N Join

As shown in Figure 2, this program is based on the relation schemes

PROFS{ **prof**(*name*:**name**, *fields*:**SUBJECTS**{**subject**})} and
GROUPS{**group**(*project*:**contract**, *namelist*:**NAMES**{**name**})}

to derive the nested relation **PROF_GROUPS** with the following scheme for obtaining information about professors participating each group.

PROF_GROUPS{**prof_group**(*project*:**contract**, *members*:**PROFS**{
prof(*name*:**name**, *fields*:**SUBJECTS**{**subject**})})}.

It is the rule-based representation of join on attribute *name* of atomic type in relation **PROF** and attribute *namelist* of set type in relation **GROUP**.

{ ***rule*** (Program P_2)

PROF_GROUPS{**prof_group**(*project*:z, *members*:**PROFS**{
prof(*name*:x, *fields*:**SUBJECTS**{y})})} ←
PROFS{**prof**(*name*:x, *fields*:**SUBJECTS**{y})} &
GROUPS{**group**(*project*:z, *namelist*:**NAMES**{x})}.

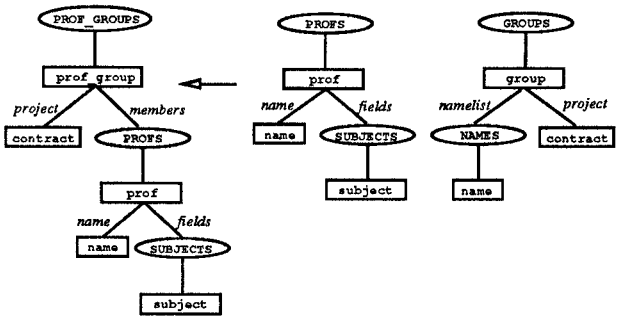
facts** (nested relations)

PROFS{
prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}),
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

GROUPS{
  group(*project*:NEC90, *namelist*:NAMES{Smith, John, Won}),
  group(*project*:IBM89, *namelist*:NAMES{Smith, Parker, Linda}),
  group(*project*:NASA90, *namelist*:NAMES{Parker, Lee})}.
}

The fact (relation) listed below can be derived from this program :

PROF_GROUPS{
  prof_group(*project*:NEC90, *members*:PROFS{
      prof(*name*:Smith, *fields*:SUBJECTS{AI, KB})}),
  prof_group(*project*:IBM89, *members*:PROFS{
      prof(*name*:Smith, *fields*:SUBJECTS{AI, KB}),
      prof(*name*:Parker, *fields*:SUBJECTS{DB, KB})}),
  prof_group(*project*:NASA90, *members*:PROFS{
      prof(*name*:Parker, *fields*:SUBJECTS{DB, KB})})}.



PROF_GROUPS

project	members	
	name	fields
NEC90	Smith	AI KB
IBM89	Parker	DB KB
	Smith	AI KB
NASA90	Parker	DB KB

PROFS

name	fields
Parker	DB KB
Smith	AI KB

GROUPS

namelist	project
Smith John Won	NEC90
Smith Parker Linda	IBM89
Parker Lee	NASA90

Figure 2 : An 1-N Join Example

An HILOG-R Program for M-N Join

As shown in Figure 3, this program is based on the relation schemes

GROUPS{**group**(*project*:**contract**, *namelist*:**NAMES**{**name**})} and
DIVISIONS{**division**(*field*:**subject**, *persons*:**NAMES**{**name**})}

to reconstruct the nested relation **CO_WORKERS** with the following scheme which is used to represent persons on the same subject and in the same group :

CO_WORKERS{**co_workers**(*persons*:**NAMES**{**name**},
field:**subject**, *project*:**contract**).

It is the rule-based representation of join on attribute *namelist* of set type in both relations.

{ ***rule*** (Program P_3)

CO_WORKERS{**co_workers**(*persons*:**NAMES**{x}, *field*:y, *project*:z)}
← **GROUPS**{**group**(*project*:z, *namelist*:**NAMES**{x})} &
DIVISIONS{**division**(*field*:y, *persons*:**NAMES**{x})}.

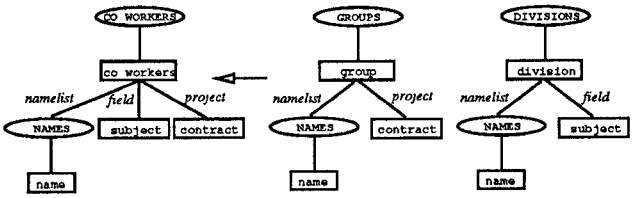
Facts

DIVISIONS{
  division(*field*:DB, *persons*:NAMES{Parker, Linda, Lee}),
  division(*field*:KB, *persons*:NAMES{Parker, Smith, John}),
  division(*field*:AI, *persons*:NAMES{Smith, Won})}.

GROUPS{
  group(*project*:NEC90, *namelist*:NAMES{Smith, John, Won}),
  group(*project*:IBM89, *namelist*:NAMES{Smith, Parker, Linda}),
  group(*project*:NASA90, *namelist*:NAMES{Parker, Lee})}.
}

The fact (relation) listed below can be derived from this program :

CO_WORKERS{
  co_workers(*persons*:NAMES{Parker, Linda},
      field:DB, *project*:IBM89),
  co_workers(*persons*:NAMES{Parker, Lee},
      field:DB, *project*:NASA90),
  co_workers(*persons*:NAMES{Smith, Parker},
      field:KB, *project*:IBM89),
  co_workers(*persons*:NAMES{Smith, John},
      field:KB, *project*:NEC90),
  co_workers(*persons*:NAMES{Parker},
      field:KB, *project*:NASA90),
  co_workers(*persons*:NAMES{Smith},
      field:AI, *project*:IBM89),
  co_workers(*persons*:NAMES{Smith, Won},
      field:AI, *project*:NEC90)}.



CO_WORKERS

namelist	field	project
Parker Linda	DB	IBM89
Parker Lee	DB	NASA90
Smith Parker	KB	IBM89
Smith John	KB	NEC90
Parker	KB	NASA90
Smith	AI	IBM89
Smith Won	AI	NEC90

GROUPS

namelist	project
Smith John Won	NEC90
Smith Parker Linda	IBM89
Parker Lee	NASA90

DIVISIONS

namelist	field
Parker Linda Lee	DB
Parker Smith John	KB
Smith Won	AI

Figure 3 : An M-N Join Example

3. Semantics of Nested Relation Reasoning

As proposed in section 1, we shall develop extended algebraic tools at set manipulation level to reformulate LP notions for underlying the semantics of nested relation reasoning. We shall extend the following LP notions : *Herbrand interpretation, satisfaction* and *model, model comparison, model intersection property*, and *least fixpoint characteristics*. In first-order LP, the typical algebraic operations involved in the above notions are *set manipulations*, such as membership (in depicting logic satisfac-

tion), containment (in model comparison), intersection (in model intersection), union (in least fixpoint computation) and so on. Since FOL is a symbolic framework without notions about structures, its set operations are single-leveled. This can be considered as the reason causing the weakness of conventional LP systems in handling complex objects.

The mismatch of the conventional LP model theory based on setwise manipulations and the HILOG-R semantics can be illustrated by the following examples. For simplicity the extention of usual set operations to typed sets is assumed. We decompose the fact

PROFS{
prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}),
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

into the following four individual **first-order LP** formulas :

prof(*name*:Parker, *fields*:DB).
prof(*name*:Parker, *fields*:KB).
prof(*name*:Smith, *fields*:AI).
prof(*name*:Smith, *fields*:KB).

Then assuming I_0 as a first-order LP interpretation containing as its set members the above four formulas, we have

prof(*name*:Parker, *fields*:DB) $\in I_0$,
:

and so on. Since in the conventional first-order LP the *satisfaction* of an atom F by an (Herbrand) interpretation I is defined as F $\in$ I, we have

$I_0 \models$ **prof**(*name*:Parker, *fields*:DB),
:

and so on. Then the individual constants DB, KB, AI are the answer to the query "What are the fields of Parker and Smith ?".

However, in the original HILOG-R formula given above the facts described by these four formulas are expressed together in a nested fashion. In this case if an interpretation I contains the above HILOG-R formula, under the conventional first-order LP notion of satisfaction, the HILOG-R formulas

prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}) or
SUBJECTS{AI, KB}

may not be interpreted by I, since under the traditional set membership notion $\in$ they may not be members of I.

Furthermore, under the UNA on types, it is assumed that the satisfaction of **SUBJECTS**{AI, KB} by an interpretation should imply the satisfaction of **SUBJECTS**{KB} and **SUBJECTS**{AI} by that interpretation. This point is also not captured by the conventional first-order LP model theory.

Another example showing the above mismatch concerns with the LP notion of *model comparison* and *model intersection*. In the conventional first-order LP, for a program, a model M is said to be smaller than a model N iff M $\subseteq$ N, and the intersection of models is a model. Suppose we have a program

P_4 = { **ISSUES**{x} $\leftarrow$ **seminar**(*title*:x), **seminar**(*title*:KB)}.

Under UNA on types

M = { **ISSUES**{KB}, **seminar**(*title*:KB) },
M_1 = { **ISSUES**{KB, DB}, **seminar**(*title*:KB) } and
M_2 = { **ISSUES**{KB, AI}, **seminar**(*title*:KB) }

are models of P. Further, M has the least redundancy. However, based on the conventional first-order LP notions, there is no way to compare them with each other and the intersection of M_1 and M_2 is not a model.

These examples show that the mismatch between the conventional LP theory and the semantics of nested relation reasoning essentially consists in the *inadequacy of the algebraic tools* used in underlying the theory, namely, single-level set membership, containment and other setwise manipulations. To develop HILOG-R semantics, it is necessary to introduce additional mathematical tools, particularly in the area of extended notions on set membership (e.g. for checking whether a HILOG-R atom is *included* in an interpretation), containment (e.g. for model reduction and comparison) and the corresponding set manipulations. These requirements constitute the background of the following formal discussions.

We plan to discuss two kinds of algebraic properties relating to the model theory of nested relation reasoning. First, we shall study the structural relationships between HILOG-R *terms* which is based on the notion of **partial inclusion**. Second, we shall develop the notion of **ungrouping** to allow various structural relationships between HILOG-R terms or sets of formulas to be handed under the unique notion of partial inclusion. Since an interpretation of a program is just a set of formulas, and most LP notions about model theory relate to certain set inclusion relationships, the above notions provide algebraic tools for extending LP model theory for handling nested relations.

Before formally introducing those algebraic notions we shall first extend the notion of the **base** of a program. For a HILOG-R program P, the *universe* U is formed from all the ground terms appearing in P, the *base* Bp is the set of all ground atoms which can be formed by the type symbols appearing in P with the ground terms in U, and the *interpretation* I is a finite subset of Bp. This extention causes the following basic difference between HILOG-R and first-order LP. In a FOL language such as DATALOG, *atoms* and *terms* are structurally distinct since an atom is preceded by a predicate symbol while a term is not. Further, nested predicates are not allowed. The nested constructs may only be considered as function terms which do not appear as individual atoms in interpretations. However, in HILOG-R, both atoms and tuple/set-terms are explicitly typed and preceded by type symbols and a HILOG-R tuple-term or set-term can be either contained in other atoms or as a top-level atom itself.

3.1 Partial Inclusion

First we shall focus to a typical relationship between terms: the part-term relationship. Then based on the part-term relationship, the notions of partial-membership and partial-containment can be introduced, where the lattice property of partial-containment allows us to further induce the operations of partial-union and partial-intersection. We generally call these notions as *partial inclusion* notions.

[**part-term**] A term t is the part-term of a term t', denoted as $t \leq: t'$, is defined recursively as follows :

a) For each term t, $t \leq: t$.
b) For a tuple-term $t = \mathbf{p}(s_1{:}t_1,...,s_n{:}t_n)$, $\forall i \in \{1,...,n\}\ t_i \leq: t$.
c) For a set-term $t = \mathbf{R}\{t_1,..., t_n\}$, $\forall i \in \{1,...,n\}\ t_i \leq: t$.
d) $t_1 \leq: t_2$ and $t_2 \leq: t_3 \rightarrow t_1 \leq: t_3$.

For example,
SUBJECTS{DB, KB} $\leq:$
prof(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}).

The above definition also applies to atoms which are the top-level terms. With the part-term concept, the special notions called set partial-membership $\in:$ and partial-containment $\subseteq:$ can be defined as follows.

[**partial-membership $\in:$ and partial-containment $\subseteq:$**]

a) Let b be an atom and A be a set of atoms, $b \in: A$ iff $(\exists\ a \in A)\ b \leq: a$.
b) Let A and B be two sets of HILOG-R atoms, $B \subseteq: A$ iff $(\forall\ b \in B)\ b \in: A$.

For example,
SUBJECTS{DB, KB} $\in:$
{**prof**(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}),
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

{**SUBJECTS**{DB, KB}} $\subseteq:$
{**prof**(*name*:Parker, *fields*:**SUBJECTS**{DB, KB}),
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

In order to show that $\subseteq:$ underlies a lattice, as $\subseteq$ does, it is necessary to introduce the notion of **p-reduction**. A set of atoms with none of its elements a part-term of another is called a *p-reduced* set. If a set is not p-reduced, we say that it contains part-term redundancy which destroys the *antisymmetry property* of $\subseteq:$ over the set of HILOG-R atoms sets. For example, consider two sets

A = { **MEETING**{**seminar**(*title*:KB), **seminar**(*title*:AI)},
seminar(*title*:KB) }.
B = { **MEETING**{**seminar**(*title*:KB), **seminar**(*title*:AI)} }.

Set A contains part-term redundancy since

seminar(*title*:KB) $\leq:$
MEETING{**seminar**(*title*:KB), **seminar**(*title*:AI)}.

As a result, both $A \subseteq: B$ and $B \subseteq: A$ hold but $A \neq B$. The part-term redundancy in a set of atomic formulas can be filtered out by dropping those elements that are part-terms of others. The function of **p-reduction** ε can be described by the following :

$\varepsilon: S \rightarrow \{\ x \mid x \in S \wedge (\forall\ y \in S)[x \neq y \rightarrow \neg\ (x \leq: y)]\ \}$

Based on the set of p-reduced sets of atoms, it can be proved that the $\subseteq:$ relationship is reflexive, transitive, and also antisymmetric. Furthermore we can show that the set of p-reduced sets of atoms form a partial order lattice under the $\subseteq:$ relationship.

The least upper bound (lub) and the greatest lower bound (glb) of the two p-reduced sets of atoms are referred to as their partial-union and partial-intersection respectively. For example, given

S_1 = {**MEETING**{**seminar**(*title*:KB), **seminar**(*title*:AI)},
tutorial(*title*:KB) } and
S_2 = { **seminar**(*title*:KB), **tutorial**(*title*:KB), **tutorial**(*title*:AI) },

we have

$S_1 \cup: S_2$ = { **MEETING**{**seminar**(*title*:KB), **seminar**(*title*:AI)},
tutorial(*title*:KB), **tutorial**(*title*:AI) }.
$S_1 \cap: S_2$ = {**seminar**(*title*:KB), **tutorial**(*title*:KB) }.

There are also other structural relationships between HILOG-R terms which cannot be directly described by the partial inclusion notions, such as the relationship between formulas

prof(*name*:Smith, *fields*:**SUBJECTS**{AI}) and
prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB}),

the relationship between formulas

SUBJECTS{DB} and
research(*division*:info_sys, *fields*:**SUBJECTS**{DB, KB}),

or the relationship between sets of formulas

{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI})} and
{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})}.

As it is unreasonable to handle atoms or interpretations based on multiple relationships, we shall introduce appropriate mappings to convert atoms and sets of atoms into a form that can be compared based on the unique partial inclusion relationships. In the next section, we shall discuss the mechanism used for transforming containment relationships between sets of HILOG-R atoms into the unique context of partial-containment.

3.2 Ungrouping

Ungrouping and **grouping** are used to handle formulas containing set-terms. Formulas in the ungrouped form can be generally compared in terms of the partial containment relationship, which serves as a fundamental notion of the HILOG-R model theory.

In general an atom with a typed component set-term can be ungrouped into a set of atoms with a single-element component set-term of the same type; an atom with multiple set-terms is stepwise ungrouped; a set of atoms is ungrouped into the union of sets resulted from ungrouping each atom. Ungrouping a single atom or a set of atoms always returns a set of atoms, this resulting set has any set-term at any level of any atom has one element. Such a treatment is iteratively applied to more complex formulas. Below are some examples. The formal definition of ungrouping is given in appendices.

σ **course**(*c#*:CS231, *title*:OS) = { **course**(*c#*:CS231, *title*:OS) }.

σ {**course**(*c#*:CS231, *title*:OS)} = { **course**(*c#*:CS231, *title*:OS) }.

σ **prof_project**(*name*:Smith, *fields*:**SUBJECTS**{AI, KB},
projects:**CONTRACTS**{IBM89, NEC90}) =
{**prof_project**(*name*:Smith, *fields*:**SUBJECTS**{AI},
projects:**CONTRACTS**{IBM89}),
prof_project(*name*:Smith, *fields*:**SUBJECTS**{AI},
projects:**CONTRACTS**{NEC90}),
prof_project(*name*:Smith, *fields*:**SUBJECTS**{KB},
projects:**CONTRACTS**{IBM89})
prof_project(*name*:Smith, *fields*:**SUBJECTS**{KB},
projects:**CONTRACTS**{NEC90})
}.

Let us denote the "combine" operation as ψ (Note ∈ is extended to the membership of typed set-terms) such that

$$\psi\{S_1,...,S_n\} = \{S_1,...,S_n\} \text{ if } S_i\text{'s are not sets;}$$
$$\{s \mid s \in S_1 \vee ... \vee s \in S_n\} \text{ otherwise.}$$
$$\psi\,\mathbf{R}\{S_1,...,S_n\} = \mathbf{R}\{S_1,...,S_n\} \text{ if } S_i\text{'s are not sets;}$$
$$\mathbf{R}\{s \mid s \in S_1 \vee ... \vee s \in S_n\} \text{ otherwise.}$$

and denote $\{f{:}t_1,..,\ f{:}t_n\}$ as $@f : \{t_1,...,\ t_n\}$. Ungrouping is recursively defined as follows :

[Ungrouping]

a) The ungrouping mapping σ on an atom f is resulted by applying the mapping on all the part-terms of f in the top-down order as follows:
- For an atomic constant A, $\sigma : A \rightarrow A$
- For a set-term, $\sigma : \mathbf{R}\{S_1,...,S_m\} \rightarrow \beta(\psi\ \mathbf{R}\{\sigma S_1,...,\sigma S_m\})$, where $\beta : \mathbf{Q}\{s_1,...,s_m\} \rightarrow \bigcup_{k=1}^{m} \{\mathbf{Q}\{s_k\}\}$
- For a tuple-term, σ: $\mathbf{p}(u_1{:}a_1,..,u_n{:}a_n) \rightarrow \delta\mathbf{p}\,(u_1{:}\sigma\,a_1,..,\ u_n{:}\sigma\,a_n)$
where for a given $t = \mathbf{q}(u_1{:}a_1,..,u_i{:}a_i,..,u_m{:}a_m)$, then
$\delta_i t = \text{not-set}\,(a_i) \rightarrow \{t\};\ a_i = \mathbf{S}\{s_1,..,s_d\} \rightarrow \bigcup_{k=1}^{d} \{\mathbf{q}(u_1{:}a_1,..,u_i{:}\mathbf{S}\{s_k\},..,u_m{:}a_m)\}$
$\delta_{i,j}t = \psi@\delta_j\,(\delta_i t)$, $\delta_{i,j,..,k} = \delta_{i,(j,..k)}$, and $\delta t = \delta_{1,..,m}\ t$.

b) For a set of HILOG atoms $I = \{F_1,...,F_n\}$, ungrouping is defined as $\sigma : I \rightarrow \bigcup_{i=1}^{n} \sigma F_i$.

Grouping, denoted as π, is the transformation defined on a set of atoms for converting the set into the grouped form. *Grouped formulas are preferred in representing nested relations.* Although in certain cases grouping may not yield a unique result, in these cases the results represent the same semantics under UNA on types. Grouping is not simply the inverse of ungrouping. To group a set of atoms I, we first ungroup it to I', then followed by grouping I'. Therefore a set of atoms may be neither grouped nor ungrouped. In general, to group a set which is not in the ungrouped form, we should first ungroup it.

The grouping/ungrouping conversion is made meaningful under the assumption of UNA on types, since terms of the same type can be identified and traced through type symbol. For example, having said that the existence of **staff**{a,b,c} implies the existence of **staff**{a} is due to the same type of formula. If another specific grouping of a and b is of interest, {a,b} must be specifically typed to catch the meaning of that grouping. Grouping/ungrouping untyped sets is semantically meaningless.

Ungrouping makes it possible to handle various memberships between an atom and a set of atoms, and containment relationships between sets of atoms (e.g. interpretations) under the unique context of partial-containment, since ungrouped atoms contain only single-element set-terms. This feature is used in developing the HILOG-R model theory shown below.

3.3 Models of A Program

In first order-LP, the satisfaction of a ground atom F by a (Herbrand) interpretation I is defined as $F \in I$. A rule instance $h \leftarrow b_1 \,\&\, ... \,\&\, b_n$ is satisfied by I iff $b_1,..,\ b_n \in I$ implies $h \in I$. It is easy to see that with this notion the question we raised at the beginning of this section cannot be answered, that is, how to explain that

SUBJECTS{AI, KB} and
prof(*name*:Smith, *fields*:**SUBJECTS**{AI})

are satisfied by any interpretation I containing as its set member the following formula

prof(*name*:Smith, *fields*:**SUBJECTS**{AI, KB})

where AI, DB are *grouped* under a (unique) set-type **SUBJECTS**. Intuitively we can see that these formulas are indeed somehow *included* in I. However the inclusion relationships involved are not all the same, and it is not reasonable to have the notion of satisfaction expressed in terms of various inclusion relationships. Instead, it should be defined under a unique relationship. Based on the algebraic notions introduced above, we can solve this problem and develop Herbrand-like models of HILOG-R programs.

As discussed above, ungrouping an atom yields a set of atoms, and ungrouped sets of atoms can be compared under the unique relationship of partial-containment $\subseteq$:. Thus, checking the satisfaction of an atom F by an interpretation I can be made through checking whether the set σ F is partial-contained in σ I. In other words, the concept of satisfaction is generally expressed by using the notions of ungrouping and partial-containment.

In fact, the ungrouped form of the above interpretation σ I includes (as its set elements) the following formulas :

prof(*name*:Smith, *fields*:**SUBJECTS**{AI}).
prof(*name*:Smith, *fields*:**SUBJECTS**{KB}).

Then the above relationship can be uniformly represented in terms of $\subseteq$: relationship as

σ**SUBJECTS**{AI, KB} $\subseteq$: σ I.
σ**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI}) $\subseteq$: σ I.

Thus the notion of satisfaction $I \models F$ extends membership in FOL to partial-containment in HILOG-R, that is, σ F $\subseteq$: σ I. Note that σ F $\subseteq$: σ I can cover $F \in I$, F ∈: I and other forms of inclusion relationships, since in all those cases the ungrouping forms of F and I can be generally compared under $\subseteq$:.

[**Satisfication**] Let I be an interpretation. The notion of satisfaction, denoted as |=, is defined as

a) For a ground atom F, if F is a comparison formula then

I ⊨ F iff it is a tautology, otherwise I ⊨ F iff $\sigma F \subseteq: \sigma I$.
b) For $H \leftarrow B_1 \& ... \& B_k$, I ⊨ $(H \leftarrow B_1 \& ... \& B_k)$
iff $I \models B_1,..,I \models B_k$ implies I ⊨ H.
c) For a program P, I ⊨ P iff for each rule r in P, I ⊨ r.

[Model of Program] A model of program P is an interpretation of P which satisfies P.

The least model issue is essential to all LP systems. In the following discussions we shall first extend the notion of model comparison, then we shall show that a HILOG-R program has a unique least model called the **immediate least model**, and there exists a so called **least model equivalent class** containing those models whose ungrouped form are the same and equal to the immediate least model of that program under our extended model comparison notion. We shall also demonstrate that the model intersection property is preserved and extended to the **model partial-intersection property**, and the immediate least model of a program is computable through the **least fixpoint computation.**

In FOL, a model M is said to be smaller than a model N if $M \subseteq N$, M is said to be the least iff for any model N, $N \subseteq M$ means N = M. However, for the models consisting of hierarchically structured HILOG-R atoms, such a one-level set containment comparison is not applicable. The non-minimality of a model for a HILOG-R program may be caused by the existence of extra atoms or "bigger" atoms that "contain" other atoms. In order to capture such semantics and to provide a unified formalism for HILOG-R model comparison, we shall map models of a HILOG-R program to their ungrouped forms for comparison to each other, and use partial-containment as the comparison operator. Ungrouping makes it possible to handle various containment relationships between sets of atoms (e.g. interpretations) under the unique context of partial-containment. Atoms in an ungrouped set only contain single-element set-terms. Therefore, for two sets of atoms, regardless of their original containment relationship, their ungrouped forms can be compared under $\subseteq:$ relationship. For example, consider a simple HILIG-R program

P_5 = { **research**(*division*:info_sys, *fields*:**BRUNCHES**{x})←
SUBJECTS{x}, **SUBJECTS**{DB, KB} }

and the following models of this program

N_1 = {**research**(*division*:info_sys, *fields*:**BRUNCHES**{DB, KB}),
SUBJECTS{DB, KB} }

N_2 = {**director**(*name*:Hauss, *duty*:**research**(*division*:info_sys,
fields:**BRUNCHES**{DB, KB})),
SUBJECTS{DB, KB} }

N_3 = {**research**(*division*:info_sys, *fields*:**BRUNCHES**{DB, KB, AI}),
SUBJECTS{DB, KB, AI} }

Although N_2 and N_3 seem "bigger" than N_1, they may not be comparable under either $\subseteq$ or $\subseteq:$ directly. However, their ungrouped forms can be compared under the unique relationship $\subseteq:$ as

$$\sigma N_1 \subseteq: \sigma N_2, \qquad \sigma N_1 \subseteq: \sigma N_3 .$$

This fact indicates the use of ungrouping in the notion of HILOG-R model comparison which can be defined by the following.

[Model Comparison] For a program P, a model M is said to be smaller than N, iff $\sigma M \subseteq: \sigma N$. Further, M is the immediate least model iff for any model N, $\sigma N \subseteq: M$ means $\sigma N = M$.

From the above definition it is easy to see that the immediate least model of a HILOG-R program must be in the ungrouped form. The fact that a HILOG-R program may have a set of models with the same minimal ungrouped form as the immediate least model, is very different from that of the conventional first-order LP. Such a set of models is referred to as the **least model equivalent class**, where the immediate least model is the common ungrouped form of all the models in the least model equivalent class. That is, let E be the least model equivalent class of a program and MM be the immediate least model of the program, then

$$(\forall M \in E)\, \sigma M = MM.$$

For example, for the above simple HILIG-R program P_5, its immediate least model

M_0 = {**research**(*division*:info_sys, *fields*:**BRUNCHES**{DB}),
research(*division*:info_sys, *fields*:**BRUNCHES**{KB}),
SUBJECTS{DB}, **SUBJECTS**{KB} }

and the following models are in its least model equivalent class :

M_1 = {**research**(*division*:info_sys, *fields*:**BRUNCHES**{DB, KB}),
SUBJECTS{DB, KB} }

M_2 = {**research**(*division*:info_sys, *fields*:**BRUNCHES**{DB, KB}),
SUBJECTS{DB}, **SUBJECTS**{KB} }

M_3 = {**research**(*division*:info_sys, *fields*:**BRANCHES**{DB}),
research(*division*:info_sys, *fields*:**BRUNCHES**{KB}),
SUBJECTS{DB, KB} }

where M_0 is the common ungrouped form of all of them (Figure 4), and M_1 is the preferred model since it is fully grouped.

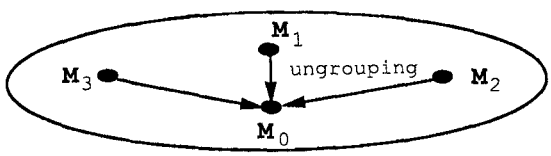


Figure 4 : Least Model Equivalent Class

It is worth noting the relevance of the minimal model semantics between HILOG-R programs and FOL programs. Note that the traditional notion of model comparison is only a special case in our definition. For a FOL program, the least model equivalent class is reduced to a unique element MM and $\sigma MM = MM$.

In the conventional first-order LP, the intersection of two models of a program is still a model of that program. The least model can be defined as the intersection of all the models of that program. This model intersection property is considered a key issue of LP semantics. In HILOG-R it can be extended to the so-called **model partial-intersection property**. For each model, the part-term redundancy can be removed by p-reduction, other

redundancy caused by structural containment can be either reduced by ungrouping or converted to the part-term redundancy by ungrouping. For the set of all models, the extra-atom redundancy can be removed by the model intersection. In general, redundancies can be removed by partial-intersection. Below is the extended model partial-intersection property where the usual set-wise intersection is just a special case of partial-intersection.

Theorem 1 : Model Partial-intersection Property
Let P be a HILOG-R program and $\{M_i\}$ be the set of all models for P, then $\cap$: [$\varepsilon \bullet \sigma M_i$] is the immediate least model of P.

Proof Outline : (a) After performing p-reduction (by applying ε) on the ungrouped models, $\cap$: is a legal operation on them. Clearly MM = $\cap$: [$\varepsilon \bullet \sigma M_i$] is an ungrouped interpretation and also is a model. (b) If MM is not the least, then we can assume there exists a model σ N $\neq$ MM such that σ N $\subseteq$: MM. The fact that MM = MM $\cap$: $\varepsilon \bullet \sigma$ N implies MM $\subseteq$: $\varepsilon \bullet \sigma$ N. Since $\varepsilon \bullet \sigma$ N $\subseteq$: σ N, we have MM $\subseteq$: σ N which contradicts the above assumption. Thus MM is the immediate least model among all the models of P.

One of the key points in LP model theory is the existence of a unique least model for a program. This may not hold for some extended LP systems but it holds for a HILOG-R program, as within the least model equivalent class of a HILOG-R program the immediate least model is unique.

Theorem 2 : Uniqueness of Immediate Least Model
The immediate least model of a HILOG-R program is unique.

Proof Outline : It is straightforward. By definition if M is the immediate least model σ M = M. The existence of two immediate least models M and N would cause either σ M = N or σ N = M, in both cases M = N can be concluded.

3.4 Least Fixpoint Characteristics of the Immediate Least Model

In the conventional first-order LP, the least model MM of a program P has the fixpoint characteristics, namely, MM = lfp(T_P). The function T_P for fixpoint computation is based on the rules in P and the interpretation I. As described in [LLO83],

T_P(I) = I $\cup$ {H | H $\leftarrow B_1$ & ... & B_n is a ground instance of a rule in P and B_1,.., $B_n \in$ I}.

For HILOG-R programs, clearly such a set-wise union of I and {h} may not remove the structural redundancies discussed before and thus may not result in the immediate least model. Therefore we redefine the function T_P as follows:

[**Mapping Tp**] Let P be a HILOG-R program and I be the interpretation, then

T_P(I) = $\varepsilon \bullet \sigma$ I $\cup$: $\varepsilon \bullet \sigma$ {H | H $\leftarrow B_1$ & ... & B_n is a ground instance of a rule in P and σB_1, ..., $\sigma B_n \subseteq$: σ I }.

Theorem 3 : Least Fixpoint Characteristics of Immediate Least Model
Let P be a HILOG-R program, then its immediate least model MM = lfp(T_P).

Proof Outline : (a) Under $\subseteq$: relationship the set of σ I constitute a lattice. According to the definition clearly T_P is monotonic and continuous. (b) I is a model of P iff I is a model for each ground instance H $\leftarrow B_1$ & ... & B_n in P, for i=1,..,n, $\sigma B_i \subseteq$: σ I implies I |= H iff σ (T_P(I)) $\subseteq$: σ (I). (c) According to the model partial-intersection property MM = glb{I | I |= P} which means MM = glb{I | σ (T_P(I)) $\subseteq$: σ (I)}. Then MM = lfp (T_P) (refer to [LAS82], [LLO83]).

This theorem reveals that the immediate least model of a program P can be computed by applying T_P to an empty set $\varnothing$ a finite number of times up to its saturation, then converting the result to the grouped form by π mapping.

We show below the use of function T to compute the immediate least model of the simple program P_2 given before. This example also shows the sense of ungrouping. It is easy to see that $T_{P_2}(\varnothing)$ yields the original facts given in the program which are expressed in ungrouped form. Then each application of T_{P_2} to the current interpretation extends the interpretation by introducing the derived facts on the basis of that interpretation, until no more new facts can be yielded. The results of this example are shown below.

$T_{P_2}(\varnothing)$ results in the following interpretation M_0

{ **PROFS**{**prof**(*name*:Parker, *fields*:**SUBJECTS**{DB})}.
PROFS{**prof**(*name*:Parker, *fields*:**SUBJECTS**{KB})}.
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI})}.
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{KB})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{Smith})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{John})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{Won})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Smith})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Parker})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Linda})}.
GROUPS{**group**(*project*:NASA90, *namelist*:**NAMES**{Parker})}.
GROUPS{**group**(*project*:NASA90, *namelist*:**NAMES**{Lee})}.
}

$T_{P_2}(T_{P_2}(\varnothing))$ results in the following interpretation M_1

{ **PROFS**{**prof**(*name*:Parker, *fields*:**SUBJECTS**{DB})}.
PROFS{**prof**(*name*:Parker, *fields*:**SUBJECTS**{KB})}.
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI})}.
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{KB})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{Smith})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{John})}.
GROUPS{**group**(*project*:NEC90, *namelist*:**NAMES**{Won})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Smith})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Parker})}.
GROUPS{**group**(*project*:IBM89, *namelist*:**NAMES**{Linda})}.
GROUPS{**group**(*project*:NASA90, *namelist*:**NAMES**{Parker})}.
GROUPS{**group**(*project*:NASA90, *namelist*:**NAMES**{Lee})}.

PROF_GROUPS{**prof_group**(*project*:NEC90, *members*:
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI})})}.
PROF_GROUPS{**prof_group**(*project*:NEC90, *members*:
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{KB})})}.
PROF_GROUPS{**prof_group**(*project*:IBM89, *members*:
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{AI})})}.
PROF_GROUPS{**prof_group**(*project*:IBM89, *members*:
PROFS{**prof**(*name*:Smith, *fields*:**SUBJECTS**{KB})})}.
PROF_GROUPS{**prof_group**(*project*:IBM89, *members*:

PROFS{prof(*name*:Parker, *fields*:SUBJECTS{DB}))}.
PROF_GROUPS{prof_group(*project*:IBM89, *members*:
PROFS{prof(*name*:Parker, *fields*:SUBJECTS{KB}))}.
PROF_GROUPS{prof_group(*project*:NASA90, *members*:
PROFS{prof(*name*:Parker, *fields*:SUBJECTS{DB}))}.
PROF_GROUPS{prof_group(*project*:NASA90, *members*:
PROFS{prof(*name*:Parker, *fields*:SUBJECTS{KB}))}.
}

$T_{P_2}(T_{P_2}(T_{P_2}(\varnothing))) = T_{P_2}(T_{P_2}(\varnothing))$.

Thus the ungrouped immediate least model is

$MM = T_{P_2}(T_{P_1}(\varnothing)) = M_1$.

Its grouped form MM' is

{ PROFS{
prof(*name*:Parker, *fields*:SUBJECTS{DB, KB}),
prof(*name*:Smith, *fields*:SUBJECTS{AI, KB})}.

GROUPS{
group(*project*:NEC90, *namelist*:NAMES{Smith, John, Won}),
group(*project*:IBM89, *namelist*:NAMES{Smith, Parker, Linda}),
group(*project*:NASA90, *namelist*:NAMES{Parker, Lee})}.

PROF_GROUPS{
prof_group(*project*:NEC90, *members*:PROFS{
prof(*name*:Smith, *fields*:SUBJECTS{AI, KB}))},
prof_group(*project*:IBM89, *members*:PROFS{
prof(*name*:Smith, *fields*:SUBJECTS{AI, KB}),
prof(*name*:Parker, *fields*:SUBJECTS{DB, KB}))},
prof_group(*project*:NASA90, *members*:PROFS{
prof(*name*:Parker, *fields*:SUBJECTS{DB, KB}))}.
}

It is easy to see that MM and MM' are both in the least model equivalent class of program P_2.

The discussion given in this chapter indicates the computability of a HILOG-R program used in the inference of nested relations. In fact, the execution of HILOG-R programs is based on the least fixpoint computation demonstrated here. The extended least fixpoint characteristics also underlies our implementation model [CHEN88] of LP with high-order constructs.

The notion of least model equivalent class, model partial-intersection property, and least fixpoint characteristics, have represented both the semantic difference and the semantic analogy between HILOG-R and the conventional first-order LP. While the major notions of the conventional first-order LP have their counterparts in HILOG-R constructed in terms of extended set manipulation notions, those notions can be represented as the special cases of the corresponding HILOG-R notions. Such analogy can be kept due to that the extention is made at the fundamental set theory level.

4. Conclusions

We have developed the formal semantics for nested relation based knowledge representation. Since our approach is characterized by **extending the underlying notions at set theory level** the resulting framework is capable of dealing with hierarchically nested tuples and relations, and supporting flexible nesting/unnesting at any level of a hierarchy. Different from the previous LP extentions, our framework is **canonical and analogous to first-order LP framework.**

References

[ABI89] S. Abiteboul, C. Beeri, M. Gyssens & D. Gucht, "An Introduction to the Completeness of Languages for Complex Objects and Nested Relations," In Lecture Notes in Computer Science (361), Springer-Verlag, 1989.

[APT86] K.R. Apt, H. Blair, A. Walker, "Towards a Theory of Declarative Knowledge," Proc. of Workshop on Foundations of Deductive Databases and Logic Programming, USA, 1986. Also in J. Minker, editor, Deductive Databases and Logic Programming, Morgan Kaufmann.

[BEE86] C. Beeri et.al, "Sets and Negation in a Logic Database Language LDL1," MCC Rep,1986.

[CHEN86] Q. Chen, "A Rule-based Object/Task Modeling Approach," ACM SIGMOD Record, Vol.15, No. 2, 1986.

[CHEN88] Q. Chen and G. Gardarin, "An Implementation Model for Reasoning with Complex Objects," ACM SIGMOD Record Vol.17, No.3, 1988.

[CHEN89] Q. Chen, "A High Order Logic Programming Framework for Complex Objects Reasoning," Proc. COMPSAC'89, 1989, USA.

[GAR87] G. Gardarin, E. Simon, "Les Systemes de Bases de Donnees Deductives," TSI, Dunod Ed., A paraitre.

[GAL84] H. Gallaire, J. Minker, J. Nicolas, "Logic and Databases : A Deductive Approach," ACM Computing Surveys, Vol. 16, No 2, June 1984.

[HULL89] R. Hull, "Four Views of Complex Objects : A Sophisticate's Introduction," In Lecture Notes in Computer Science (361), Springer-Verlag, 1989.

[KAM83] Y. Kambayashi, K. Tanaka and K. Kakeda, "Synthesis of Unnormalized Relations Incorporating More Meaning," Info. Sci. Vol.29, 1983.

[KAM89] Y. Kambayashi, T. Furukawa and H. Yamamoto, "Realization of Nested Relation Interfaces for Relational and Network Databases," In Lecture Notes in Computer Science (361), Springer-Verlag, 1989.

[KUP87] G. Kuper "Logic Programming with Sets," Proc. PODS, 1987.

[LAS82] J. Lassez, V. Nguyen and E. Sonenberg, "Fixed Point Theorems and Semantics: A Folk Tale, IPL, 14,3, 1982.

[LLO83] J. Lloyd, "Foundation of Logic Programming," Springer-Verlag, 1983.

[MAI86] D. Maier, "A Logic for Objects," Proc. of Workshop on Foundations of Deductive Databases and Logic Programming, USA, 1986.

[REI84] R. Reiter, "Towards a Logical Reconstruction of Relational Database Theory," in On Conceptual Modeling, pp 191-234, Springer- Verlag Ed ., 1984.

[TAKE89] K. Takeda, "On the Uniqueness of Nested Relations," In Lecture Notes in Computer Science (361), Springer-Verlag, 1989.

[TSU86] S. Tsur and C. Zaniolo, "LDL: A Logic Based Data Language," Proc. VLDB 12, 1986.

[ZAN85] C. Zaniolo, "The Representation and Deductive Retrieval of Complex Objects," Proc. VLDB 11, 1985.