Sets and Negation in a Logic Database Language (LDL1)

Catriel Beeri
Shamim Naqvi
Raghu Ramakrishnan
Oded Shmueli
Shalom Tsur

MCC
P O Box 200195
Austin, TX 78720

ABSTRACT

In this paper we extend LDL, a Logic Based Database Language, to include finite sets and negation The new language is called LDL1 We define the notion of a model and show that a negation-free program need not have a model, and that it may have more than one minimal model We impose syntactic restrictions in order to define a deterministic language These restrictions allow only layered (stratified) programs We prove that for any program satisfying the syntactic restrictions of layering, there is a minimal model, and that this model can be constructed in a bottom-up fashion Extensions to the basic grouping mechanism are proposed We show that these extensions can be translated into equivalent LDL1 programs Finally, we show how the technique of magic sets can be extended to translate LDL1 programs into equivalent programs which can often be executed more efficiently

1 Introduction

LDL (Logic Data Language) is an attempt to combine the benefits of Logic Programming with those of relational query languages The motivation and basic features of this language were described in [TZ85,TZ86] While the style of presentation in those documents was intuitive and was primarily motivated by our desire to present the basic ideas, this paper concentrates on formal aspects In particular, we provide a unified framework in which we describe the semantics of Horn-clauses with sets and negation Before doing so however, we will, by means of some examples, introduce the essential features

An LDL program consists of a set of rules that, like Horn clauses, have a single head predicate and a body which consists of a conjunction of one or more predicates The rules serve to specify *derived relations* from a set of relations Queries are specified by the conjunction of one or more predicates which correspond to rule heads or base relations and the specification of appropriate argument bindings Unlike PROLOG, which we shall use here for comparison purposes, the LDL programmer does not have explicit control over the order of execution of the predicates within a rule or the order of execution of the rules Hence, within certain restrictions which we shall outline in the sequel, the semantics of an LDL program is like that of an assertional language —devoid of any user control

A subset of LDL programs, denoted *simple programs* consists of rules without any negated predicates or set terms The ancestor example is by now a classical representative of this subset.

```
ancestor(X, Y) <-- ancestor(X, Z), parent(Z, Y)
ancestor(X, Y) <-- parent(X, Y)
```

An *admissible LDL program* consists of a set of rules which may include negated predicates or set terms subject to the restrictions of layering (or stratification) [ABW86, GELD86, NAQV86] The following is an admissible LDL program which derives an "exclusive ancestor" relation i e , all ancestors but not those of a particular individual (the binding to Z)

```
ancestor(X, Y) <-- parent(X, Y)
ancestor(X, Y) <-- parent(X, Z), ancestor(Z, Y)

excl_ancestor(X, Y, Z) <-- ancestor(X, Y), ¬ ancestor(X, Z)
```

This program consists of two "layers" one layer, consisting of the

first two *ancestor* rules and the *parent* base relation and a second layer consisting of the *excl_ancestor* rule The second layer may be computed once the first one has been computed

The following is an inadmissible LDL program since it cannot be stratified, the reason being that *even* must be in a layer below *even* (*s(X)* denotes the successor of *X*)

```
int(0)
int(s(X)) <-- int(X)

even(0)
even(s(X)) <-- int(X), ¬ even(X)
```

Set terms as arguments of LDL predicates can be used for two different purposes *set-enumeration* and *set-grouping* Set enumeration is the process of constructing a set from its elements as in the following example

```
book_deal({X, Y, Z}) <-- book(X, Px), book(Y, Py),
        book(Z, Pz), Px+Py+Pz<100
```

The example derives a relation on sets of book titles from the *book* base relation such that their total price does not exceed 100 Dollars The cardinality of the sets in the derived relation is bounded by 3, duplicate elements are eliminated during the set construction process so that books with the same title but a different price e g , paperbacks and hardcovers are eliminated Observe that *book_deal* may yield singleton and doublet sets

The following example is of set-grouping in which all of the parts supplied by a supplier are grouped with the supplier number

```
part_sets(S#, <P#>) <-- supplier(S#, P#)
```

Unlike set-enumeration, where the set is constructed by listing its elements, in grouping the set is constructed by defining its elements by a property (i e , a conjunction of predicates) that they satisfy It follows that the cardinality of the grouped set in a derived relation is unbounded (but finite) The last illustrative example of an admissible LDL program is included to demonstrate the power of the language

```
part(P#, <Subpart#>) <-- p(P#, Subpart#)
tc({X}, C) <-- q(X, C)
tc({X}, C) <-- part(X, S), tc(S, C)
tc(S, C) <-- partition(S, S1, S2), tc(S1, C1),
        tc(S2, C2), +(C1, C2, C)
result(P#, C) <-- tc({P#}, C)
```

The (normalized) base relations for this problem are *p(P#, Subpart#)* which contains tuples of the type (part number, immediate-sub-part number)[1] and *q(P#, Cost)* which contains tuples of the type (elementary part number, cost of part) [2] The result is the derived relation *result(P#, Cost)* which contains for each part, elementary or aggregate, the cost of that part. The cost of an aggregate part is the sum of the costs of its immediate subparts The first line derives the *part* relation which groups for each part number *P#* the set of immediate sub-part numbers <*Subpart#*> Thus, if base relation *p* would contain {*p(1,2), p(1,7), p(2,3), p(2,4), p(3,5), p(3,6)*} then the corresponding *part* relation would contain {*part(1,{2,7}), part(2,{3,4}), part(3,{5,6})*} Derived relation *tc* contains the total cost *C* for a set of constituent part numbers *S* The first *tc* rule derives for each elementary part *X* a singleton set {*X*} of that part and its cost The second *tc* rule derives for each aggregate part *X* a singleton set {*X*} and the cost *C* of its immediate subparts If base relation *q* would contain {*q(4,20), q(5,10), q(6,15), q(7,200)*} then the second *tc* rule would contribute the following tuples {*tc({3},25), tc({2},45), tc({1},245)*} The third *tc* rule is the recursive derivation of the cost of a part as the sum of the costs of its subparts The result is derived from the *tc* relation by selecting only those tuples which consist of singleton sets, i e , elementary part numbers or aggregate part numbers The *partition* predicate partitions a set *S* into two disjoint subsets, it can be realized by using the built-in predicate *union*.

While a semantics for the set-enumeration terms of LDL can still be provided within the Herbrand universe, the inclusion of set-grouping terms forces us to depart from this realm We define thus in section 2 the syntax and semantics which are based on an extended universe and the interpretation of the various constructs

[1] For simplicity we will assume that a part cannot have more than one identical immediate subpart.

[2] It is interesting to note that, if base relation *q* would be "impure" in the sense that it would also contain cost tuples for some of the aggregate parts, the derivation would still hold.

within this universe With this interpretation we define the notion of a model for an LDL program and show some properties, e g , that a negation-free program need not have a model, and that it may have more than one minimal model In section 3 we prove that for the class of admissible programs there is a "standard" minimal model and that it can be constructed in a bottom-up fashion Another class of LDL programs is that of negation-free (or *positive*) programs For admissible positive programs and hence for simple programs, the minimal model is unique

The version of LDL described in this paper is termed LDL1 In section 4 we propose various extensions which serve to enhance the usability of the language They do not however add any power to it These extensions are defined in terms of LDL1 and can be thought of as source rewriting rules or "macros" which can be expanded into LDL1 rules LDL1 plus its extensions is termed LDL1 5

Section 5 provides a comparison between LDL1 and the set-oriented logic programming language LPS [KUPE86] We show that LDL1 is more powerful than LPS Finally, in section 6 we sketch how the magic-sets techniques of [BMSU86] can be used in the compilation of LDL1 programs to improve execution efficiency Section 7 presents some open problems and summarizes our conclusions

2 Syntax and Semantics

In this section we present the detailed syntax of the language, and the semantics of the new constructs It is well known that the semantics of logic programs can be defined in several equivalent ways, e g , model-theoretic and through lattice-theoretic fixed points We show at the end of this section why a straight-forward extension of these standard approaches is prevented by the new constructs in LDL1

2 1 Syntax

Variables are denoted by capital letters, e g , X,Y,Z (Occasionally we shall use $\overline{X}$ to denote a sequence of variables) We use "_" to denote an anonymous variable Constants are denoted by lower-case letters, e g a,b,c Function symbols are denoted by strings of lower-case letters Associated with each function symbol is an integer —its arity The symbol *scons* denotes a built-in function of arity 2 Some function symbols are reserved LDL1 words

A *simple term* [LLOY84] is defined inductively as a variable, a constant, or if f is an n-ary function symbol and $t_1, \ldots, t_n$ are simple terms then $f(t_1, \ldots, t_n)$ is a simple term *LDL 1 terms* (or simply *terms*) are obtained from *simple terms* by adding {} to the set of constants, adding *scons* to the collection of functions symbols, adding <X>, where X is a variable, to the collection of terms, and by defining (inductively) $f(t_1, \ldots, t_n)$ to be a term where f is a n-ary function symbol, and the t_i, $1 \leq i \leq n$, are terms but not of the form <X>

Remark LDL1 has lists, and arithmetic and comparison predicates as built-in predicates Lists are handled in the usual manner as in logic programming, a treatment of arithmetic and comparison predicates is outside the scope of this paper

A *predicate symbol* is denoted by a lower-case letter, perhaps followed by other characters Some predicate symbols are reserved by LDL1, e g , *member, union* Associated with each predicate name there is an integer —its arity A (positive) *predicate* is a formula of the form $p(t_1, \ldots, t_n)$ where the t_i are LDL1 terms, and p is a predicate symbol of arity n A *negative predicate* is a formula of the form $\neg p$ where p is a predicate A *literal* is a positive or negative predicate A *rule* (or *clause*) is a formula of the form *head* <-- *body* where *head* is a predicate and *body* is a comma-separated sequence of literals, possibly empty A rule with an empty body is called a *fact* and is simply written as *head* A rule containing <> in the head is called a *grouping* rule

A rule is *well–formed* if it obeys the following syntactic restrictions

(1) the body contains no occurrence of the form <X>,

(2) the head contains at most one occurrence of the form <X>, such an occurrence must be an argument of the head predicate symbol

(3) All the predicates in the body of a grouping rule are positive

A *program* is a finite set of well-formed rules A program is

positive if none of its rules has an occurrence of a negative predicate in its body

The intended interpretation for the above syntax is defined formally in the next sub-section We briefly sketch the intended meaning of the syntactic constructs Intuitively, {} stands for the empty set, *scons* is a binary function *scons*(t,S), where t is a term and S is a set, that adds the element t to the set S A term constructed only from {} and scons is called an *enumerated set.* The angular brackets are used to group together, into a (finite) set, elements which satisfy some qualification specified in the body of the rule The predicate symbols *member* (of arity 2) and *union* (of arity 3) test respectively set membership and set union

2 2 Semantics

For a set S let F(S) denote the set of all finite subsets of S Let U_0 be the set of all simple variable-free terms Note that U_0 may be infinite and observe that terms involving *scons* are not contained in U_0 For $n>0$ U_n is defined inductively as follows

$G_{n,0} = U_{n-1} \cup F(U_{n-1})$

$G_{n,j} = G_{n,j-1} \cup \{f(t_1, \quad, t_k) \mid$ f is of arity k, f is not *scons*, and $t_i \in G_{n,j-1}$, $1 \leq i \leq k\}$

$U_n = \cup_{j=0}^{\infty} G_{n,j}$

Let $U = \cup_{i=0}^{\infty} U_i$ U is called the *LDL1 universe* It should be noted that the LDL1 universe is different from the Herbrand universe [LLOY84], i e , U_0 is the classical Herbrand Universe, and U_n is its ω-closure under subsets and function application An *interpretation* of an LDL1 program is defined in a manner analogous to Herbrand interpretations as follows (let I(t) denote the interpretation of t)

(1) U is the domain of interpretation

(2) Constants, c, are assigned to "themselves" in U, i e I(c)=c, except the constant {} which is assigned ϕ, i e , the empty set

(3) If f is not a built-in function symbol and has arity n, then f, restricted to U, is assigned a mapping from U^n to U which maps $(t_1, \quad, t_n)$ into $f(t_1, \quad, t_n)$, where for i=1, ,n, $t_i \in U$ If some $t_i \notin U$ then $f(t_1, \quad, t_n)$ is an object outside U Built-in functions must respect the restrictions below

(4) For each non-built-in predicate symbol of arity n there is an assignment of $\sim$ relation on U^n Built-in predicates must respect the restrictions below

Restrictions on the Interpretation of Built-in Functions and Predicates

(1) *scons*(t,S) is in U only when $S, t \in U$, S is a set, its value is $\{t\} \cup S$, otherwise it is an object outside of U

(2) *member*(t,S) is **true** only when $S, t \in U$, S is a set, and $t \in S$, otherwise it is **false**

(3) *union*(S_1,S_2,S_3) is **true** only when all of S_1,S_2,S_3 are sets in U, and $S_1 \cup S_2 = S_3$, otherwise it is **false**

(4) $=(a,b)$ is **true** if $a, b \in U$ and $a = b$, **false** otherwise $/=(a,b)$ is **false** if $a, b \in U$ and $a = b$, **true** otherwise

A *U-fact* is an object of the form $p(e_1, \quad, e_n)$ where p is an n-ary predicate symbol and for i=1, ,n $e_i \in U$ A subset of U_0-*facts* defines an interpretation in the classical Herbrand universe Similarly we shall take a subset of U-*facts* to define an interpretation in the *LDL 1 universe*

The notion of truth value for a formula induced by an interpretation I is defined as in [LLOY84] with the following modifications Consider first the formula $p(<Y>) <-- body(\bar{X}, Y)$ If there are infinitely many Y values for which $body(\bar{X}, Y)$ holds under I, then $body(\bar{X}, Y)$ evaluates to **false** and the formula evaluates to **true** Otherwise, suppose that there is a finite non-empty set of Y values for which $body(\bar{X}, Y)$ holds under I Then the formula evaluates to **true** if p holds on the set containing these values, otherwise the formula evaluates to **false** Note that when the set of elements to be grouped is empty, the formula evaluates to **true** even if p does not hold on the empty set

Generally, consider a formula of the form $p(t_1, \quad, t_n, <Y>) <-- body(\bar{X}, Y)$, where $\bar{Z}$ are all the variables within $t_1, \quad, t_n$ and $\bar{X}$ are the variables appearing in

the body except for Y, $\overline{Z}$ may however include Y Let $\overline{V}$ be the set of all variables in the above formula. Intuitively, one can view the body as evaluating a relation R Then, R is partitioned "horizontally" for each distinct combination of values in $\overline{Z}$ Next the values for the Y column in each finite partition are grouped into a set The formula is true if for each combination of $\overline{V}$ values for which there is a compatible tuple in R, p holds for this combination applied to $t_1, \quad , t_n$ with $<Y>$ replaced with the grouped set associated with the $\overline{Z}$ value combination, otherwise, it is **false**

Formally, the formula evaluates to **true** under I if for each combination of U-values, Γ, assigned to the variables $\overline{Z}$, for which there is an assignment for $\overline{V}-\overline{Z}$ such that there are a positive finite number of Y *values* for which *body* evaluates to **true**, there is a tuple $(s_1, \quad , s_n, S)$ in the interpretation of the p predicate such that

(1) s_i is the interpretation of t_i under I where variables are substituted for by values according to Γ, and

(2) S is the finite set of all Y value assignments for which $(\overline{V}-\{Y\})-\overline{Z}$ can be assigned values so that body evaluates to **true** [3]

If such a p-tuple is absent in p 's interpretation then the formula evaluates to **false**

Note that when a variable, say X, appearing in head of a rule also appears as $<X>$ in the same head then the grouped set is a singleton, we do not expect this situation to occur in most natural definitions

An interpretation I is a *model* for a set of LDL1 rules R if each rule in R is assigned **true** under I For clarity of presentation, when representing an interpretation as a set of $U-facts$ M we actually mean M' where M' is M augmented with all $U-facts$ for built-in predicates

Example. Consider the program P

[3] Alternatively, with a slight abuse of notation we can express when the formula evaluates to true by

$$\forall \overline{Z} / (\exists \textit{ finitely many } Y \exists (\overline{X}-\overline{Z}) \; body(\overline{X},Y)) \rightarrow$$
$$\exists S / p(t_1, \quad , t_n, S) \wedge$$
$$\forall Y (\exists (\overline{X}-\overline{Z}) \; body(\overline{X},Y) \leftrightarrow member(Y,S)) \; //$$

```
q(X) <-- p(X),h(X)
p(<X>) <-- r(X)
r(1)
h({1})
```

The set {r(1), h({1}), p({1}), q({1})} is a model whereas {r(1), h({1}), p({1,2})} is not a model []

2.3 Problems with the Classical Approach to Semantics of Programs

So far our definitions and concepts have closely followed the classical model-theoretic approach in which the semantics of logic programs are defined by considering models, and minimal models We shall show that simple extensions of such an approach for LDL1 are problematic For example, a well known property of classical logic programs is that the intersection of two models of a program is also a model This is not the case for LDL1 programs as the following example shows

Example: Consider the program [(p(<X>) <-- q(X))] Possible models for the above program are A={ q(1), q(2), p({1,2}) } and B={ q(2), q(3), p({2,3}) } However, A ∩ B is not a model as it does not contain $p(\{2\})$ []

Not all LDL1 programs have models

Example: Consider the program P [(p(<X>) <-- p(X)), (p(1))] Suppose P has a model M Let Z be the subset of U which interprets p Since M is a model of P we must have that $\{e \mid e \in Z\}$ is a subset of some element of Z But Z, in U, cannot be a subset of an element in Z Thus P does not have a model This is reminiscent of the Russel-Whitehead paradoxes []

Next, when a *positive* LDL1 program has a model it does not necessarily have a unique minimal model

Example: Consider the program P [(p(<X>) <-- q(X)), (q(Y) <-- w(S,Y),p(S)), (q(1)), (w({1},7))] Note, first, that M={q(1),w({1},7)} is not a model (Note that even if we add p({7}) to M it is still not a model) However, M_1=M ∪ {q(2),p({1,2})} is a model M_2=M ∪ {q(3),p({1,3})} is also a model It can be checked that M_1 and M_2 are minimal So P does not have a unique minimal model (taking the classical definition of minimality based on set inclusion) []

The idea of performing set-wise intersections to produce new "smaller" models also fails We are thus forced to conclude that a simple extension of the standard model-theoretic approach to semantics of logic programs will not work We thus define a non-standard notion of model minimality

2 4 Minimal Models

A $U-fact\ e = p(s_1, \ldots, s_n)$ is *dominated* by a $U-fact\ e' = p(s_1', \ldots, s_n')$, denoted $e \leq e'$, if for $i=1, \ldots, n$ if s_i is a set then $s_i \subseteq s_i'$ and otherwise $s_i = s_i'$

A function ρ from $U-facts$ to $U-facts$ is *preserving* if, for all e, $\rho(e) \leq e$ For a set of $U-facts$ M, let $\rho(M)$ denote the image of M under ρ

Let M, M' be sets of $U-facts$ which are models for P M' is a *submodel* of M, denoted $M' \leq M$, if there is $M'' \subseteq M$ and a preserving function ρ such that $\rho(M'') = M'$

A model M is *minimal* if there does not exist a model M' different than M such that $(M' - M) \leq (M - M')$ Observe that $(M' - M) \leq (M - M')$ implies $M' \leq M$ Note that in the absence of set terms this definition of minimality reduces to the usual definition based on set inclusion

Remark We may define a more elaborate notion of minimality as follows A $U-element\ e$ is *dominated* by a $U-element\ e'$, denoted $e \leq e'$, if either
(i) $e = e'$, or

(ii) $e = f(s_1, \ldots, s_n)$, $e' = f(s'_1, \ldots, s'_n)$ and for $i=1, \ldots, n$ $s_i \leq s'_i$, or

(iii) e and e' are sets and $\forall a \in e$, $\exists b \in e'$ such that $a \leq b$

Now a $U-fact\ p(s_1, \ldots, s_n)$ is dominated by a $U-fact\ p(s'_1, \ldots, s'_n)$ if for $i=1, \ldots, n$, $s_i \leq s'_i$ We claim that the results of this paper hold for this more elaborate definition of minimality as well

Example: Consider the program P

```
q(1)
p(<X>) <-- q(X)
q(2) <-- p({1,2})
```

$M_1 = \{q(1), q(2), p(\{1,2\})\}$ is a model for P However, M_1 is not minimal Consider $M_2 = \{q(1), p(\{1\})\}$ which is also a model for P $M_2 - M_1 = \{p(\{1\})\} \leq \{p(\{1,2\}), q(1)\} = M_1 - M_2$ M_2 is minimal as any model must include q(1) and any model must have a $U-fact$ of the form p({1, }) Thus, for any model M_3, $M_2 - M_3 \leq M_3 - M_2$ []

Observe that the program $P = [(p(<X>) <-- q(X)), (q(Y) <-- w(S,Y), p(S)), (q(1)), (w(\{1\},7))]$, shown previously not to possess a unique minimal model in the classical set-inclusion sense of minimality, also does not have a unique minimal model under our new definition of minimality

3 Semantics for Admissible LDL1 Programs

In this section we provide operational semantics for a subclass of LDL1 programs We begin by placing additional syntactical restrictions on LDL1 programs, thereby defining the class of admissible LDL1 programs The restrictions amount to splitting programs into distinct partitions, called *layers* or *strata* We shall show that for admissible programs there is always a minimal model and that this model can be computed (effectively if finite) in a repetitive fashion, the computation proceeding from the lowest to the highest layer For positive programs the minimal model is unique

3.1 Admissible Programs

Consider an LDL1 program P, define relations $>, \geq$ on predicate symbols appearing in P

(1) $p \geq q$ if there is a rule in P in which p is the head predicate symbol, there is no occurrence of the form <X> in the head and q appears non-negated in the body of the rule

(2) $p > q$ if there is a rule in P in which p is the head predicate symbol, there is an occurrence of the form <X> in the head and q appears in the body of the rule

(3) $p > q$ if there is a rule in P in which p is the head predicate symbol and q appears negated in the body of the rule

Program P is *admissible* if there is no sequence of predicate symbols in the rules of P of the form

$$p_1 \theta_1 p_2 \quad \ldots \quad \theta_{k-1} p_k$$

such that $p_1 = p_k$, for $i=1, \ldots, k-1$, $\theta_i \in \{>, \geq\}$ and some θ_j is $>$, $1 \leq j \leq k-1$ A *layering* for P is a partition $L_0, \ldots, L_m$ of the predicate symbols of P such that for all p, q

(1) if $p \geq q$, $p \in L_i$, $q \in L_j$ then $i \geq j$, and

(2) if $p > q$, $p \in L_i$, $q \in L_j$ then $i > j$

Observe that there may be more than one layering for a given program

Lemma 3 1 P is admissible iff there exists a layering of P

Proof Omitted []

3 2 Bottom-Up Semantics of Admissible Programs

Let M be a set of U-facts A *binding* θ is a set of pairs $\{X_1/t_1, \ldots, X_n/t_n\}$ where $X_1, \ldots, X_n$ are variables, and $t_1, \ldots, t_n$ are elements in U If A is a predicate with variables $X_1, \ldots, X_n$ and θ a binding then $A\theta$ denotes the simultaneous replacement of all the variables of A by the corresponding elements in θ *followed by the application of all functions in A to obtain a U-fact* or an object which is not a $U-fact$ in case an argument evaluates to an object outside of U For example, let A be the predicate $p(scons(a,X))$ and θ={X/{a}} Then $A\theta$ is the $U-fact$ p({a})

An LDL1 rule is *simple* if it contains no occurrence of the form <X> in its head and no negative literal in its body Let r be a simple rule

$$B <-- B_1, \ldots, B_m$$

Let M be a set of $U-facts$ A binding θ is *applicable* w r t r and M if

(1) Bθ and $B_i\theta$ are $U-facts$ $1 \leq i \leq m$

(2) $B_i\theta \in M$, $1 \leq i \leq m$

The *application of r to M* denoted $r(M)$, is defined as

$r(M)$ = {$B\theta$ | θ is applicable w r t r and M and B is the head of rule r}

For a set R of simple rules $R(M)$, the *application* of R to M, is defined as

$R_0(M)=M$

$R_{i+1}(M)=\cup_{r \in R} r(R_i(M)) \cup R_i(M)$

$R(M)=\cup_{i=0}^{\infty} R_i(M)$

Interpretation I' is a *sub-interpretation* of an interpretation I, denoted $I' \subseteq I$, if I and I' are identical except perhaps in predicates p such that $I'(p) \subset I(p)$ We say that an interpretation N is a *minimal model* of R *w r t an interpretation* M if

(i) $M \subseteq N$, and

(ii) N is a model of R, and

(iii) There does not exist a model M' of R different from N such that $M' \supseteq M$, and $(M' - N) \leq (N - M')$ Note that if M is empty then this definition reduces to our definition of minimality in section 2

Lemma 3 2 1 Let R be a set of simple LDL1 rules with a layering consisting of one layer and M a set of U-facts (i) $R(M)$ is a model for R containing M, and, (ii) $R(M)$ is the unique minimal model of R w r t M

Proof

(i) Assume $R(M)$ is not a model Then there is a rule $r \in R$ of the form

$$head <-- B_1, \ldots, B_n$$

and binding θ such that $\{B_1\theta, \ldots, B_n\theta\} \subseteq R(M)$, but $head\theta \notin R(M)$ Consider some $B_i\theta$, if $B_i\theta \in R_0(M)$ let $j = 0$, otherwise, since $B_i\theta \in R(M)$ there must be some j such that $B_i\theta \in R_{j+1}(M) - R_j(M)$ Let $\alpha(i)$ denote this j Let m be $\max\{\alpha(i) \mid 1 \leq i \leq n\}$ Clearly, $head\theta \in R_{j+2}$ and hence $head\theta \in R(M)$ Contradiction

(ii) $R(M)$ is minimal Otherwise there exists Ω such that $(\Omega - R(M)) \leq (R(M) - \Omega)$ w r t M If $R(M) \subseteq \Omega$ then $\Omega - R(M) \leq \emptyset$, $\Omega = M$, contradiction So, $Q = R(M) - \Omega$ is not empty Define, for $q \in Q$, $\alpha(q)$ to be the integer such that $q \in R_{j+1}(M) - R_j(M)$ Let m be $\min\{\alpha(q) \mid q \in Q\}$ Clearly, $m > 0$ (since $M \subseteq \Omega$, and $M = R_0(M)$) Thus $R_{m-1}(M) \subseteq \Omega$ Consider some $q \in Q$ such that $\alpha(q) = m$ Since $R_{m-1}(M) \subseteq \Omega$ and Ω is a model for R it follows that $q \in \Omega$ as well Contradiction

Suppose there exists another minimal model, Φ w r t M Again, $R(M) \not\subseteq \Phi$ since this contradicts Φ's minimality So $R(M) - \Phi \neq \emptyset$ Thus, in the same manner as above, a contradiction can be derived []

We now define $r(M)$ where r's body contains negative literals but its head has no occurrence of the form <X> The

definition is identical to that of $r(M)$ for simple rules except that the condition

(2) $B_i\theta$ is in M, $1 \leq i \leq m$

is replaced by

(2') If B_i is positive then $B_i\theta \in M$ and otherwise $B_i\theta \notin M$, $1 \leq i \leq m$

With the above extension to $r(M)$, extend the definition of $R(M)$ to cover sets R which may contain rules having negated body predicates (but no occurrence of the form <X>)

Lemma 3 2 2 Let R be a set of LDL1 rules with a layering consisting of a single layer, in which no rule head has an occurrence of the form <X>, and let M be a set of $U-facts$ Then $R(M)$ is a minimal model of R w r t M

Proof (sketch) The proof proceeds in five steps We first show a construction of R' and M' from R and M, and then of Ω from $R'(M')$ We show that $\Omega = R(M)$, that Ω is a model for R' containing M, that Ω is a model for R containing M, and that Ω is a minimal model for R w r t M

Construction For a predicate symbol $p \in R$ such that $\neg p(\ )$ occurs in the body of some rule, introduce a new predicate symbol $\overline{p}$ Let

$\delta = \{\overline{p}(t_1, \ldots, t_n) \mid$ p is a n-ary predicate symbol in R, $t_i \in U, 1 \leq i \leq n$, and $p(t_1, \ldots, t_n) \notin M\}$

$M' = M \cup \delta$

Let R' be R in which all occurrences of the form $\neg p(\ )$ have been replaced by $\overline{p}(\ )$ R' is a simple set of rules and Lemma 3 2 1 applies, i e , $R'(M')$ is the unique minimal model of R' w r t M' Let $\Omega = R'(M') - \delta$

The following sequence of claims, whose proofs are omitted, derives the lemma.

Claim 1 $\Omega = R(M)$

Claim 2 Ω is a model for R' containing M

Claim 3 Ω is a model for R containing M

Claim 4 Ω is a minimal model for R w r t M []

We now extend the definition of $r(M)$ to the case when the rule r may have an occurrence of <X> in its head $\overline{T}$ denotes a sequence of terms W l o g let r be of the form

$p(\overline{T}, <X>) \leftarrow B_1, \ldots, B_n$

Define r^- to be

$p(\overline{T}) \leftarrow B_1, \ldots, B_n$

Let Σ be the set of bindings applicable to r^- w r t M Define an equivalence relation $\equiv$ on Σ viz $\theta_1 \equiv \theta_2$ if for all $t_i \in \overline{T}$, $t_i\theta_1$ and $t_i\theta_2$ are interpreted as the same element in U Let SE be the set of equivalence classes under $\equiv$ Let Σ_j be an equivalence class under $\equiv$, i e , $\Sigma_j \in SE$

Define

$p\Sigma_j = p(\overline{T}\theta_1, \{X\theta \mid \theta \in \Sigma_j\})$

where θ_1 is some arbitrary member of Σ_j Finally,

$r(M) = \{p\Sigma_j \mid \Sigma_j \in SE$ and the second component in $p\Sigma_j$ is non-empty and finite $\}$

Extend $R(M)$ further as above for R containing rules with a <X> occurrence

Lemma 3 2 3 For a set of rules R, possibly with occurrences of the form <X> in some rule heads, and a set of U-facts M, $R(M)$ is a minimal model of R w r t M

Proof Split R into two disjoint parts R1 and R2 such that R1 contains rules having an occurrence of a term of the form <X> in the head R2 gets the remaining rules Since R is a single layer, no predicate in the body of a rule in R1 is in R2, hence rules in $R1$ need only be satisfied by grouping over U-facts in M Hence, $R1(M)$ is a model for $R1$ containing M By Lemma 3 2 2, since R2 is a collection of rules without an occurrence of <X>, R2(R1(M)) is a minimal model for $R2$ w r t $R1(M)$ We claim that R2(R1(M))=R(M) The argument is as follows

(i) M is contained in both $R1(M)$ and $R2(R1(M))$

(ii) The grouping is based solely on the U-facts in M because R is a single layer and thus the same U-facts will be derived for grouping by rules in $R(M)$ and $R2(R1(M))$

$R(M)$ is a model for R because $R1(M)$ is a model for $R1$ containing M and $R2(R1(M))$ is a model for R as no U-fact

derived by $R2$ can influence the satisfaction of a rule in $R1$

Finally, we argue that $R(M)$ is a minimal model for R w r t M Suppose, for the sake of deriving a contradiction, that there exists N, different from $R(M)$, such that $M \subseteq N$ and $(N - R(M)) \leq (R(M) - N)$ Consider $\alpha \in R(M) - N$ Since $R(M) = R2(R1(M))$ it follows that either (i) $\alpha \in R1(M)$, or (ii) $\alpha \in R2(R1(M)) - R1(M)$

Assume (i), $\alpha \in R1(M)$ implies for some rule $r \in R1$ of the form

$$head <-- B_1, \quad , B_n$$

α is needed for satisfaction But the body of this rule r depends only on $U-facts$ in M and $M \subseteq N$ By layering, for each predicate symbol p of a $U-fact$ in M, $R(M)/p = M/p$, hence, to satisfy $(N - R(M)) \leq (R(M) - N)$ it must be that $N/p = M/p$ That is, N and $R(M)$ have the same $U-facts$ for all predicate symbols p appearing in M Thus, it must be that $\alpha \in N$ and thus $\alpha \notin R(M) - N$

Assume (ii), by (i) $R1(M) \subseteq N$ By Lemma 3 2 2, $R2(R1(M))$ is a minimal model for R w r t $R1(M)$ N is a model for R w r t $R1(M)$ By definition, $(N - R2(R1(M)) \leq (R2(R1(M)) - N)$ is only possible if $N = R2(R1(M)) = R(M)$ Contradiction []

Corollary If R is positive then R(M) is the unique minimal model of R w r t M

Proof $R1(M)$ must be a subset of *any* model of R w r t M If R is positive then $R2(R1(M))$ is the unique minimal model of $R2$ w r t $R1(M)$ []

Theorem 1 Given an admissible program P partitioned into layers $L_1, \quad , L_n$ and a set of U-facts M_0 let $M_1 = L_1(M_0), \quad , M_i = L_i(M_{i-1}) \quad , \quad , M_n = L_n(M_{n-1})$ Then M_n is a minimal model for P w r t M_0

Proof We show by induction that M_n is a minimal model for P w r t M_0 containing M_{i-1}

Basis By definition of layering, the rules in L_1 do not depend upon rules in any other layer Thus, by Lemma 3 2 3, M_1 is a minimal model for L_1 w r t M_0

Induction Assume that, for $l \leq i$, M_l is a minimal model for $L_1 \cup \quad \cup L_l$ w r t M_0 containing M_{l-1} By Lemma 3 2 3 M_{i+1} is a minimal model for L_{i+1} w r t M_i

We need to show that M_{i+1} is a minimal model for $L_1 \cup \quad \cup L_{i+1}$ w r t M_0 Assume that M_{i+1} is not a model From the induction hypothesis, we know that M_i is a minimal model for $L_1 \cup \quad \cup L_i$ containing M_{i-1}, and M_{i+1} is a (minimal) model for L_{i+1} containing M_i So M_{i+1} must not be a model because there exists a rule (w l o g not containing negated predicates and no occurrences of the form <X>)

$$A <-- B_1, \quad , B_n$$

in $L_k, k < i+1$ and a binding θ such that $A\theta \notin M_{i+1}$ whereas $B_j\theta \in M_{i+1}, 1 \leq j \leq n$ This can only be if some $B_j\theta \notin M_k$ but $B_j\theta \in M_{i+1}$ (That is, $B_j\theta$ has been derived in layer L_{i+1} whereas it was absent in some lower layer) But because of layering that can not happen because all derivations of the type $B_j\theta$ occur in, or below, layer L_k So M_{i+1} is a model for $L_1 \cup \ , \quad , \cup L_{i+1}$ containing M_i

We now show that M_{i+1} is a *minimal* model for P w r t M_0 Assume that M_{i+1} is not a minimal model for $L_1 \cup \ , \quad , \cup L_{i+1}$ w r t M_0 Then there exists a model N of $L_1, \quad , L_{i+1}$ w r t M_0 such that $(N - M_{i+1}) \leq (M_{i+1} - N)$, N different than M_{i+1} It must be that N does not contain M_i because N is clearly a model for L_{i+1} and, by Lemma 3 2 3, M_{i+1} is a minimal model for the layer L_{i+1} w r t M_i, and therefore there can not be a model N for layer L_{i+1} containing M_i such that N is different from M_{i+1} and $(N - M_{i+1}) \leq (M_{i+1} - N)$ Let j be the smallest natural number such that N does not contain M_j Clearly, $j > 0$ because N and M_{i+1} both contain M_0, $j < i$ because N does not contain M_i as shown above Delete from N all U-facts associated with predicates appearing as head predicate symbols in layers $L_{j+1}, \quad , L_{i+1}$ Call the result N'

Claim. N' is a model for $L_1 \cup \quad \cup L_j$ containing M_{j-1} such that $(N' - M_j) \leq (M_j - N')$

Proof

(i) N' contains M_{j-1} by definition of j, layering and construction of N'

(ii) N' is a model for $L_1 \cup \quad \cup L_j$ because the deleted U-

facts correspond to predicate symbols that do not appear in bodies of rules in $L_1 \cup \quad \cup L_j$

(iii) $N' - M_j \leq M_j - N'$ because $N - M_{i+1} \leq M_{i+1} - N$, $M_j \subseteq M_{i+1}$ and the U-facts deleted from N' correspond to predicate symbols that appear in heads only for rules in layers $L_{j+1}, \quad , L_{i+1}$

This is a contradiction, because by Lemma 3 2 3, M_j is a minimal model for L_j w r t M_{j-1} Therefore, it must be that $N' = M_j$

This contradicts the definition of j because $N' \subseteq N$ and N does not contain M_j Hence M_{i+1} is a minimal model for P w r t M_0 containing M_i

This concludes the induction and the proof []

Corollary If $M = \phi$ then M_n is a minimal model for P []

Corollary If $M = \phi$ and P does *not* contain any negated predicates then M_n is the unique minimal model for P []

Theorem 2 Let $L_1, \quad , L_a$ and $E_1, \quad , E_b$ be two layerings for P Let $A_0 = B_0$ be a set of U-facts Then, $A_k = B_k$ where A_i and B_i are defined, for L and E, similarly to M_i in the previous theorem, i e , $A_i = L_i(A_{i-1})$, $B_i = E_i(B_{i-1})$

Proof Omitted []

3 3 The Power of Grouping

Using grouping, a negative predicate may be converted into a positive one as follows ($\perp$ is a constant in the *LDL 1 universe* whose usage is prohibited in programs)

$$p(\overline{T})$$

is converted into $g(\overline{T}, \perp)$

and the following rules are added

$$g(\overline{T}, <S>) <-- ok(\overline{T}, S)$$

$$ok(\overline{T}, \perp)$$

$$ok(\overline{T}, S) <-- S = \{\overline{T}\}, p(\overline{T})$$

Observe that (1) an admissible program remains so after this transformation, and (2) we state without proof that the standard model for the transformed program, restricted to predicates in the original program, is the standard model for the original program

4. Extensions to LDL1

In this section we propose extensions to LDL1 All extensions are designed to increase the ease of writing programs Denote LDL1 augmented with these new features as LDL1 5 All new features are such that LDL1 5 programs can be translated into LDL1 programs Thus, LDL1 5 has the same expressive power as LDL1

4.1 Complex Terms in Rule-Bodies and their Meaning

Consider a body predicate involving '<' and '>' Its meaning is different than a similar occurrence in the head While in the head it specifies what is to be generated based on elements supplied from the body, in the body it specifies how these values are obtained and which patterns are allowed Simply put, in the body it specifies a uniform pattern of terms involving angular brackets (and parentheses) that can unify with some term involving only the curly brackets For example, a term p(<X>) in the body of a rule specifies that it only matches with p tuples containing one entry which is a set, and that X ranges over elements of this set, i e , the elements of the set can be matched with X As another example, consider p(<<X>>) in the body of a rule This means that a match is allowed only with tuples of p containing a single entry which is a set all of whose elements are sets So, the above term does not match with p({1,2}, 3, {4,5}}) because 3 is not a set, however, it does match with p({{1,2}, {3}, {4,5}})

Thus, a term containing angular brackets specifies that a term to be matched with has a uniform structure in all its subterms This structure is specified by the parentheses and angular brackets We allow arbitrary terms involving the < > construct at any level of nesting in the body of a rule The interpretation of such terms is by translating the predicate containing the term into a new LDL1 term, adding a literal to the rule body and adding some new rules as follows

Suppose a term $<t>$ appears in literal A with predicate symbol p in a body of a rule r and t does not contain any occurrence of the form < > Rewrite r as follows

(1) Replace $<t>$, textually, by a new variable S

(2) Append the following literals to r member(t,S), collect(S,S)

(3) Add the following rule (where t is taken textually from the original rule)

collect(X,<Y>) <-- member(t,X), Y = t

The above transformation is performed repeatedly until no rule contains an occurrence of the form $<t>$ in its body The transformation terminates as each step reduces the number of levels of nesting of terms in the program The transformation ensures that (1) $<t>$ is a set, (2) t is a member of that set, and (3) the set is of a uniform structure

4 2 Complex Head Terms and their Meaning

We propose another possible extension to LDL which allows for more complex groupings to be expressible in head terms We shall provide semantics for this extension As we already stated, the extension improves the usability of the language but it does not add any power to it Furthermore, the semantics that we associate here with the extension is *one* of a number of reasonable *alternatives* and the syntax used here can be used with a different semantics

Consider a rule of the form *head* <-- *body* The body is assumed to supply a set of elements, this set may be viewed as a (non first normal form) relation over $X_1, \ldots, X_n$ where $X_1, \ldots, X_n$ are the variables appearing in the body The head is interpreted against this body relation The interpretation is a function of the structure of the head predicate The general form of a rule head is then some well balanced expression containing variables, commas (','), parentheses ('(' and ')') and angular brackets ('<' and '>') The formal definition is given below

4 2 1 Syntax

Definition A *head term* is defined inductively as (1) a constant, (2) a variable, (3) a variable or a constant enclosed in angular brackets, (4) a tuple head term $f(t_1, \ldots, t_n)$ where f is a functor and for $i = 1, \ldots, n$ t_i is a head term (the functor may be omitted in which case it is understood to be the functor *tuple*), and, (5) a tuple head term enclosed in angular brackets

Example. Some valid head terms

X, <X>, (X), (X,Y), <g(X,Y)>, (X,<X>,<Y>), (X,<h(Y,<Z>)>,(Y,<W>)), (X,Y,Z,<W>) []

First, we give some examples to demonstrate the intended use of head terms

Example:

Consider a relation r(Teacher,Student,Classes,Day) which means Teacher teaches Student in class Class on day Day

(T,<S>,<D>) each tuple has a teacher, the set of students taking some class with this teacher, and the set of days on which this teacher teaches some class

(T,<h(S,<D>)>) each tuple has a teacher in the first component and a set of tuples t in the second Such a tuple "t" has a student who takes some class from teacher and a set of days in which the student takes some class as the second component (not necessarily with this teacher)

(T,S,<h(C,<D>)>) For each teacher student combination a set of tuples The first component in a tuple is a class taken by student and taught by teacher The second is a set of days this class is taught by someone (not necessarily teacher T)

4 2 2 Rules for Converting Extended LDL1 Programs into LDL1 Programs

We now show how rules containing complex head terms can be rewritten This transformation can be viewed as the preprocessing of LDL1 5 rules and their expansion into LDL1 rules The transformation of a rule containing a complex head term proceeds as follows

(1) Match the head with the appropriate transformation rule

(2) Replace the rule with the target rules that pertain to the

transformation

(3) Repeat steps (1) and (2) until none of the transformations apply any more

The transformations are defined as follows

Notation **X,Y** denote sequences of distinct variables **Z** denotes the sequence of all head variables which, in some occurrence within the head, do not appear within $< >$ $term$, $term_i$, represent terms which are not simple variables

Basic rules

(i) (Distribution) p(**X**, $term_1$, , $term_n$) <-- body

Translation

p_i(**Z**,$term_i$) <-- body $i=1$, ,n

p(**X**,Y_1, , Y_n) <-- p_1(**Z**,Y_1), ,p_n(**Z**,Y_n), body

(ii) (Grouping) p(**X**,<g(**Y**, $term_1$, , $term_n$)>) <-- body

Translation

q(**Y**,$term_1$, , $term_n$) <-- body

q1(**Y**,g(**Y**,Y_1, , Y_n)) <-- q(**Y**,Y_1, , Y_n)

p(**X**, <S>) <-- q1(**Y**,S), body

(iii) (Nesting) p(**X**,g(**Y**, $term_1$, , $term_n$)) <-- body

Translation

q1(**Z**,$term_1$, , $term_n$) <-- body

q2(**Z**,g(**Y**,Y_1, , Y_n)) <-- q1(**Z**,Y_1, , Y_n)

p(**X**, S) <-- q2(**Z**,S), body

Rules for Degenerate Cases

(a) No **X** Erase it together with its comma, from all translation rules

(b) No g Erase g and its parentheses from all translation rules

(c) No $term_i$ Simply follow the translation scheme with n=0

(d) No **Y** handle as in (a) above

We suggested the possibility of associating alternative semantics with the rewriting rules of this section As an example, consider the following alternative to rules (i) - (iii) above (we only write (ii)', (i)' and (iii)' are respectively similar to (i) and (iii)) Here **X** affects the grouping together with **Y**

(ii)' (Grouping)

p(**X**,<g(**Y**, $term_1$, , $term_n$)>) <-- $body$

Translation

q(**X**,**Y**,$term_1$, , $term_n$) <-- body

q1(**X**,**Y**,g(**X**,**Y**,Y_1, , Y_n)) <-- q(**X**,**Y**,Y_1, , Y_n)

p(**X**,<S>) <-- q1(**X**,**Y**,g(**X**,**Y**,Y_1, , Y_n)),
S = g(**Y**,Y_1, , Y_n), body

5 Comparison of LDL1 with LPS

Kuper [KUPE86] has recently proposed a useful extension to logic programming, denoted LPS, by allowing rules of the form (*)

$$head \text{ <-- } (\forall x_1 \in X_1) \quad (\forall x_n \in X_n)\ [B_1, \quad , B_m],$$

where head and the B_i's are literals, the x_i's are of type element and X_i's of type set. All sets in LPS are finite See [KUPE86] for precise definitions

Example: Predicate disj tests whether X and Y are disjoint sets Predicate subset tests whether X is a subset of Y

disj(X, Y) <-- ($\forall x \in X$)($\forall y \in Y$) $x \neq y$

subset(X,Y) <-- ($\forall x \in X$) member(x, Y)

Theorem 3 For each LPS program P there is an LDL1 program P' whose unique minimal model, restricted to the predicates mentioned in P, is a model for P

Proof (sketch) A rule of the form (*) above can be translated into (here all lower-case letters also denote variables)

a(X_1, ,X_n,g(x_1, ,x_n)) <--
B_1, ,B_n, member(x_1,X_1), ,
member(x_n,X_n)

b(X_1, ,X_n ,g(x_1, ,x_n)) <--
member(x_1,X_1), ,member(x_n,X_n)

c(X_1, ,X_n,<S>) <-- a(X_1, ,X_n,S)

d(X_1, ,X_n,<S>) <-- b(X_1, ,X_n,S)

$head$ <-- d(X_1, ,X_n, S), e(X_1, ,X_n,S), B_1, ,B_n,

member(x_1, Y_1), ..., member(x_n, Y_n)

The a-rule, for particular sets $X_1, \ldots, X_n$, collects into a g-tuple elements for which the body is true concurrently. The b-rule, for particular sets $X_1, \ldots, X_n$, creates g-tuples with all possible combinations from $X_1, \ldots, X_n$. The c-rule (respectively d-rule), for particular sets $X_1, \ldots, X_n$, collects all g-tuples derived by the a-rule (respectively b-rule). Finally, the last rule derives *head* provided the sets produced by d and e are equal, this equality is tantamount to satisfying the $\forall$condition. []

Remark The above is only a sketch of the proof as we have not handled the case where some X_i's may be empty, we claim this to be a straight-forward task.

The following proposition indicates that LDL1 has richer models as compared to LPS.

Proposition There is an LDL1 program P having a unique minimal model M such that there is no corresponding LPS program P' and a model M' for P' such that M' restricted to the predicates of P equals M.

Proof Any model M' of an LPS program P' is based on $D \cup P(D)$ as its domain ([KUPE86], Def 4). Hence, for the following LDL1 program P [(p(<X>) <-- q(X)), (w(<X>) <-- p(X)), (q(1))] has the model M = {q(1), p({1}), w({{1}})} there is no LPS program with the (restricted) equivalent model. []

6 Magic Implementation of Admissible Programs

In this section, we consider efficient ways of executing admissible programs. The Magic Sets method ([BMSU86], [BR87]) computes relevant sets of constants and uses them to restrict computation. It does so, informally, by pushing these sets of constants through rules, including recursive rules. In a layered program containing set grouping and negation, this could lead to pushing constants through predicates in different layers, and care must be exercised in order to preserve the intended semantics. We only present the intuition here, full details shall be presented in a later report. We only consider programs in which all variables in the head appear in some positive predicate in the body.

A *database* D is a collection of facts. The result of *applying* a program to D with query q is the set of tuples in the interpretation of q in the minimal model for P $\cup$ D as defined earlier. We call this the *set of answers*. Our approach to computing the answer set consists of three steps: first, a *sideways information passing strategy* (sip) is chosen for each rule, for each combination of bound head arguments, second, an *adorned set of rules* is generated from the given set of rules, query, and sips, and third, the adorned set of rules is rewritten using a rewriting strategy such as Generalized Magic Sets ([BR87]).[4]

The rewritten rule set computes the same answers, but its specialized version of bottom-up evaluation is in general more efficient than bottom-up evaluation of the original program.

The first step in our approach is the choice of a sip for each rule for a given set of bound head arguments. Intuitively, a sip describes how bindings passed to a rule's head by a goal are used to evaluate the predicates in the rule's body.

Let r be a rule, with head predicate $p(\theta)$, and let p_h be a special predicate, denoting the head predicate restricted to its bound arguments. If a predicate appears in r's body more then once, we number its occurrences. Let $P(r)$ denote the set of predicate occurrences in the body. A sip for r is a labeled graph that satisfies the following conditions:

1. Each node is either a subset or a member of $P(r) \cup \{p_h\}$.
2. Each arc is of the form $N \rightarrow q$, with label χ, where N is a subset of $P(r) \cup \{p_h\}$, q is a member of $P(r)$, and χ is a set of variables, such that:
   (i) Each variable of χ appears in q and in an argument, not a head argument of the form <X>, of a positive member of N.
   (ii) Each member of N is *connected*[5] to a variable in χ.
   (iii) For some argument of q, all its variables appear in χ. Further, each variable of χ appears in an argument of q that satisfies this condition.

[4] We note that the other methods presented there can also be extended to cover set grouping and negation.

[5] The relation *connect* is defined in the obvious way. Each variable is connected to the predicate it appears in. Further, *connect* is an equivalence relation.

These two conditions define the nature of nodes and arcs of a sip There is a third condition which provides a consistency restriction on a sip

3 There exists a total ordering in which p_h precedes all members of $P(r)$, and for each arc $N \rightarrow q$, $q' \in N$, q' precedes q

We interpret the graph as follows The special node p_h may be thought of as a base relation whose attributes are the variables appearing in bound arguments of the head predicate When all predecessors of the predicates in a set N in the sip have been evaluated, we have a set of bindings for some of the variables of the predicates in N These predicates are evaluated with these bindings If N contains a negated predicate, say $\neg p$, we first compute p completely Let us denote the join of the positive predicates in N as *Pos* Every variable on the outgoing arc must be in *Pos*, and for each variable, we pass all values for it in (some tuple of) *Pos* However, we may often use the tuples in p to discard some tuples in *Pos*

Suppose there is an arc $N \rightarrow q$, labeled χ, in the sip and N contains p_h If one of the bound arguments in χ appears only as <X> in N, and X is in χ, it appears at first sight that we can restrict X to the values that appear in the set denoted by <X> This is incorrect because of the definition of set generation using <> [6] <X> denotes the set of all values for X for which the body is satisfied By passing (for X) only the values contained in the bound argument corresponding to <X>, we can test whether the body is satisfied for each of these values However, since we thereby restrict the computation of the body to those values of X, we have no way of determining whether the body is satisfied for any *other* values of X

Example. The following serves as a running example for this section We define a relation *young*, where *young*(X,S) holds if X has no descendants (i e , is not anyone's ancestor) and S is the set of all Y in the same generation as X The query is ?*young*(john,S) Thus, the query fails if john has descendants, and if john has no descendant, the answer is a tuple (john, S) where S is the set of all people in the same generation as john Note that by the semantics of <>, the query is defined to fail if S is empty (even if john has no descendants) The query is as follows (The relations p and *siblings* are base relations)

1 $a(X,Y) <-- p(X,Y)$
2 $a(X,Y) <-- a(X,Z), a(Z,Y)$
3 $sg(X,Y) <-- siblings(X,Y)$
4 $sg(X,Y) <-- p(Z1,X), sg(Z1,Z2), p(Z2,Y)$
5 $young(X, <Y>) <-- \neg a(X,Z), sg(X,Y)$
Query ? $young^{bf}$(john,S)

The following are the sips associated with rules 2, 4 and 5

2 $\{a_h\} \rightarrow_X a\ 1,\ \{a_h, a\ 1\} \rightarrow_Z a\ 2$
4 $\{sg_h, p\} \rightarrow_{Z1} sg$
5 $\{young_h\} \rightarrow_X \neg a,\ \{young_h, \neg a\} \rightarrow_X sg$

We now consider the second step in our approach, which is the generation of the adorned set of rules An *adornment* for an n-ary predicate p is a string a of length n on the alphabet $\{b, f\}$, where b stands for *bound* and f stands for *free* Let a program P and a query Q be given, where Q is an occurrence of a predicate q appearing in the program, with some arguments bound to constants q is called the *query predicate* Further, a sip is given for each rule in P We construct a new version of the program, denoted by P^{ad}, according to the sips We do not describe the details of this transformation here since it is exactly as described in [BR87] The differences due to sets and negated literals are covered by the definition of sips presented in this paper

For programs P_1 and P_2, and a query form p^a, we say that (P_1, p^a) and (P_2, p^a) are equivalent, if for all databases,for any assignment of constants to the arguments of p that are bound in a, the two programs produce the same application result for the resulting query on p We have the following counterpart of a theorem in [BR87] The proof of this theorem, as indeed the notion of a derivation, changes significantly due to the introduction of set grouping and negation

Theorem 3 For all p^a appearing in P^{ad}, (P, p^a) and (P^{ad}, p^a) are equivalent []

[6] This is why the second condition in the definition of a sip states that the label of a sip arc can only contain variables that appear in some argument *not of the form* <X>

Example In our running example, we obtain the following adorned set of rules

1 a^{bf}(X,Y) <-- p(X,Y)
2 a^{bf}(X,Y) <-- a^{bf}(X,Z), a^{bf}(Z,Y)
3 sg^{bf}(X,Y) <-- siblings(X,Y)
4 sg^{bf}(X,Y) <-- p(Z1,X), sg^{bf}(Z1,Z2), p(Z2,Y)
5 $young^{bf}$(X, <Y>) <-- $\neg a^{bf}$(X,Z), sg^{bf}(X,Y)
Query ? $young^{bf}$(john,S)

The next step in our approach is to rewrite P^{ad} using a rule rewriting algorithm In this paper, we consider the Generalized Magic Sets algorithm, described in [BR87] As with the transformation for producing P^{ad} from P, there is no change to the algorithm description, and we will not describe it here The basic idea is to rewrite P^{ad} into a new set of rules P^{mg} which computes the same set of answers, but is expected to have a more efficient bottom-up evaluation This is done by defining a number of auxiliary *magic* predicates and using them to restrict the number of successful applications of a rule

Although the rewriting algorithm does not change, the constraints due to negation and set grouping must be obeyed in evaluating P^{mg} bottom-up Consider a rule whose head uses grouping For each (vector of) value(s) of the bound head arguments, the body must be fully evaluated before grouping can be done Since the values for bound arguments are stored in the magic predicate corresponding to the head, this implies that for each tuple in this magic predicate, the other body predicates in this rule must be fully evaluated before the rule is applied Similarly, if a rule contains a negated literal $\neg p$, we must evaluate p fully for each (vector of) value(s) of the bound arguments In both cases (sets and negation), because the original rules are layered, the predicate occurrences that must be fully evaluated do not depend on the head predicate of the rule in which they occur However, these body predicates that must be fully evaluated could depend on the magic head predicate, and this predicate could in turn depend on them This cyclicity does not pose a problem since we only need to evaluate these body predicates fully for a *given* tuple in the magic predicate (Note that the rewritten program is not layered because of such cyclicity, although it computes the same answer set as the original program)

The proof of the following theorem again differs significantly from that of the corresponding theorem in [BR87] due to the presence of set grouping and negation

Theorem 4. Let P^{mg} denote a program obtained from P^{ad} by the Generalized Magic Sets rewriting algorithm, and let p^a be a predicate that appears in P^{ad} Then (P^{ad}, p^a) is equivalent to $(P^{mg} \cup \{\text{seed for } p^a\}, p^a)$ []

Example. The Magic Sets algorithm rewrites the adorned rule set as follows

1' $magic\_a^{bf}$(X) <-- $magic\_a^{bf}$(X) [rule 2, may be deleted]
2' $magic\_a^{bf}$(Z) <-- $magic\_a^{bf}$(X), a^{bf}(X,Z) [rule 2]
3' $magic\_a^{bf}$(X) <-- $magic\_young^{bf}$(X) [rule 5]
4' $magic\_sg^{bf}$(Z1) <-- $magic\_sg^{bf}$(X), p(Z1,X) [rule 4]
5' $magic\_sg^{bf}$(X) <-- $magic\_young^{bf}$(X),$\neg a^{bf}$(X,Z) [rule 5]
6' a^{bf}(X,Y) <-- $magic\_a^{bf}$(X), p(X,Y) [Modified rule 1]
7' a^{bf}(X,Y) <-- $magic\_a^{bf}$(X), a^{bf}(X,Z), a^{bf}(Z,Y) [Modified rule 2]
8' sg^{bf}(X,Y) <-- $magic\_sg^{bf}$(X), siblings(X,Y) [Modified rule 3]
9' sg^{bf}(X,Y) <-- $magic\_sg^{bf}$(X), p(Z1,X), sg^{bf}(Z1,Z2), p(Z2,Y) [Modified rule 4]
10' $young^{bf}$(X, <Y>) <-- $magic\_young^{bf}$(X), $\neg anc^{bf}$(X,Z), sg^{bf}(X,Y) [Modified rule 5]
11' $magic\_young^{bf}$(john) [From the query rule]
[]

7 Conclusion

A precise semantics has been provided for LDL1, a language extending logic programming languages with negation, set constructs and set grouping We have defined the notion of a model on a non-Herbrand universe, and a non-standard notion of minimality of models for an LDL1 program Also, the notion of a standard model for a subclass of LDL1 programs, the admissible (i e , layered or stratified) programs has been introduced It is shown that the standard model for an admissible program is minimal, and if additionally the program is positive then the standard model is the unique minimal model for the program Furth-

ermore, any admissible program P may be transformed into an admissible positive program P', whose models when restricted to the predicates mentioned in P, are models for P

We have proposed extensions to LDL1 to enhance its usability These do not extend the expressiveness of the language as they can all be translated into LDL1 LDL1 was compared favorably, in terms of expressive power, with the LPS set extensions to logic programming [KUPE86] We also sketched an extension of the technique of magic sets for admissible programs More work on this subject is expected

Other important issues remain One is the question whether admissibility is too restrictive a concept Some results on this issue are in [SN86] Another important question has to do with the finiteness of grouped sets It is possible to extend the approach shown in this paper to accommodate infinite sets While at first glance this issue may appear to be academic, for we can not realistically group infinite sets, a closer perusal reveals certain interesting aspects First, the finiteness condition was introduced to solve the problem that layered programs may not have a model Secondly, finiteness is a semantic condition and it may be desirable to have a syntactic restriction that could be easily checked Finally, in some cases the user may try to group an infinite set and look for some condition that may be determined to be satisfied by examining a prefix of the infinite set Had infinite grouping been allowed, an implementation could have looked at such prefixes It may be unreasonable for the language to consider such a program as ill-defined

One way out of this is to impose syntactic restrictions on the rules One such restriction could be that every head variable, or one appearing in a negative literal must appear in a positive literal in the same rule Thus, facts may not have variables as arguments It is then necessary to show that this restriction prevents grouping sets which are "out of" the LDL1 universe, and that the approach of this paper, in particular Theorem 1, could be modified accordingly These extensions, along with some other topics, are the subject of current research and will be discussed in an upcoming report

Acknowledgement

Many people have provided us with good advice and constructive feedback during the course of this work We would like to thank Francois Bancilhon, Danette Chimenti, Ravi Krishnamurthy, Marc Smith and Carlo Zaniolo In particular, we acknowledge Carlo's original intuitions about sets in LDL

REFERENCES

[ABW86] K Apt, H Blair, A Walker Towards a Theory of Declarative Knowledge, unpublished manuscript, 1986

[AE82] K Apt, M Van Emden Contributions to the Theory of Logic Programming", **J of the ACM 29**, No 3, 1982

[BMSU86] F Bancilhon, D Maier, Y Sagiv, J Ullman Magic Sets and Other Strange Ways to Implement Logic Programs, Proc of the 5th ACM Conf on PODS, Cambridge, 1986

[BR87] C Beeri, R Ramakrishnan On the Power of Magic, to appear in ACM PODS 1987

[CH85] A Chandra, D Harel Horn Clause Queries and Generalizations, **Journal of Logic Programming**, 1-15, 1985

[CLAR78] K Clark Negation as Failure, in **Logic and Databases**, (J Gallaire, J Minker, eds), Plenum Press, 1978

[FITT85] M Fitting A Kripke-Kleene Semantics for Logic Programs, **Journal of Logic Programming 4**, 295-312, 1985

[GELD86] A Van Gelder Negation as Failure Using Tight Derivations for General Logic Programs, unpublished manuscript

[KE76] R Kowalski, M Van Emden The Semantics of Predicate Logic as a Programming Language, **J of the ACM 23**, No 4, Oct 1976

[KUPE86] G M Kuper Logic Programming With Sets, XP/7 52 Workshop on Database Theory, University of Texas, Austin, 1986

[LLOY84] J Lloyd Foundations of Logic Programming, Springer-Verlag, 1984

[NAQV86] S Naqvi A Logic for Negation in Database System", MCC Technical Report Also in Proc of the Workshop on Logic and Databases, Washington, D C, 1986

[TARS55] A Tarski A Lattice-Theoretical Fixpoint Theorem and its Applications", **Pacific J Math 5**, 1955

[TZ85] S Tsur, and C Zaniolo "LDL Rev 0", MCC Technical Report, DB-150-85, 1986

[TZ86] S Tsur, and C Zaniolo "LDL A Logic-Based Data-Language", Proc 12th Int Conf on very Large Databases, Kyoto, Japan, 1986

[ZANI85] C Zaniolo The Representation and Deductive Retrieval of Complex Objects", Proc 11th Int Conf Very Large Databases, Stockholm, 1985

[KE76] R Kowalski, M Van Emden The Semantics of Predicate Logic as a Programming Language, **J of the ACM 23**, No 4, Oct 1976

[KUPE86] G M Kuper Logic Programming With Sets, XP/7 52 Workshop on Database Theory, University of Texas, Austin, 1986

[LLOY84] J Lloyd Foundations of Logic Programming, Springer-Verlag 1984

[NAQV86] S Naqvi A Logic for Negation in Database System", MCC Technical Report Also in Proc of the Workshop on Logic and Databases, Washington, D C, 1986

[SN86] O Shmueli, S Naqvi Set Grouping and Layering in Horn Clause Programs, unpublished manuscript

[TARS55] A Tarski A Lattice-Theoretical Fixpoint Theorem and its Applications", **Pacific J Math 5**, 1955

[TZ85] S Tsur, and C Zaniolo "LDL Rev 0", MCC Technical Report, DB-150-85, 1986

[TZ86] S Tsur, and C Zaniolo "LDL A Logic-Based Data-Language", Proc 12th Int Conf on very Large Databases, Kyoto, Japan, 1986

[ZANI85] C Zaniolo The Representation and Deductive Retrieval of Complex Objects", Proc 11th Int Conf Very Large Databases, Stockholm, 1985

[ULLM86] J D Ullman Implementation of Logical Query Languages for Databases, **TODS, Vol 10, No. 3, pp. 289-321,** 1985