On the Semantics and Expressive Power of Datalog-like Languages for NP Search and Optimization Problems

E. Zumpano, S. Greco, I. Trubitsyna
DEIS - Università della Calabria
via P. Bucci, Rende 87036, Italy
{greco,irina,zumpano}@deis.unical.it

P. Veltri
Università “Magna Grecia”
88100 Catanzaro, Italy
veltri@unicz.it

ABSTRACT
It has been shown that \textit{NP} (decision, search and optimization) problems can be expressed by means of DATALOG\textsuperscript{\dash} (Datalog with unstratified negation) queries under stable model semantics. Anyhow, the use of un restricted negation is often neither simple nor intuitive and, besides, DATALOG\textsuperscript{\dash} does not allow to optimize queries and to discipline the expressive power. This paper analyzes the power of Datalog-like languages in expressing \textit{NP} search and optimization problems. In more detail, in this paper we study the expressive power of several languages obtained by extending positive Datalog with intuitive and efficient constructs, i.e. stratified negation, constraints and (exclusive) disjunction. Finally, we investigate a further restricted language, called \textit{NP} Datalog, which uses disjunction only to define (nondeterministically) partitions of relations and which, in addition, captures the power of DATALOG\textsuperscript{\dash} in expressing search and optimization problems.

Categories and Subject Descriptors
D.1.6 [Programming Techniques]: Logic Programming; D.3.1 [Programming Language]: Formal Definitions and Theory—Syntax, Semantics; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages

General Terms
Algorithms, Languages, Theory.

Keywords
Deductive and Logic databases, Datalog, Search and Optimization Queries, Expressive Power of Query Languages.

1. INTRODUCTION
\textit{NP} optimization problems \cite{5,14,18} have recently received a renewed attention, also with the aim of characterizing classes of problems that are approximable \cite{3,4,12,13,17,19}. It’s well known that \textit{NP} problems can be formulated by means of unstratified DATALOG\textsuperscript{\dash} queries under nondeterministic stable model semantics so that each stable model corresponds to a possible solution \cite{7,11,16,21}. \textit{NP} optimization problems are then formulated by adding a max (or min) construct to select the stable model (thus, the solution) which maximizes (resp., minimizes) the result of a polynomial function applied to the answer relation. As an example take the \textit{Vertex Cover} problem:

**Example 1.** Given a graph \( G = (N, E) \), a subset of the vertexes \( V \subseteq N \) is a vertex cover of \( G \) if for each edge \((x, y) \in E \) either \( x \) or \( y \) is in \( V \). In particular, the problem can be formulated in terms of the query \( \langle P_1, \forall(X) \rangle \) where \( P_1 \) is the following set of rules:

\[
\forall(X) \leftarrow \text{node}(X), \neg \exists(Y), \\
\exists(Y) \leftarrow \text{node}(X), \neg \forall(X), \\
c \leftarrow \text{edge}(X, Y), \neg \forall(X), \neg \forall(Y), \neg c.
\]

and the predicates \textit{node} and \textit{edge} define, by means of a suitable number of facts, the vertices and the edges of the graph. The last rule enforces that every total stable model corresponds to some vertex cover as it is satisfied only if the conjunction \( \text{edge}(X, Y), \neg \forall(X), \neg \forall(Y) \) is false (otherwise \( c \) is undefined), whereas the first two rules define a partition of the relation \textit{node}.

The \textit{min} vertex cover problem can be expressed by selecting a stable model which minimizes the number of elements in \( \forall; \) this is expressed by means of the query \textit{goal min} \( \forall(X) \). □

The problem in using DATALOG\textsuperscript{\dash} to express search and optimization problems is that the unrestricted negation is often neither simple nor intuitive and, besides, it does not allow to discipline the expressive power and to optimize queries. Thus, in this paper we consider Datalog-like languages which extend positive Datalog with intuitive and efficient constructs. In such languages the use of stable model semantics is disciplined to refrain from abstruse forms of unstratified negation which make programs difficult to both understand and optimize \cite{8}. We show that the same expressive power is achieved by a language, denoted by DATALOG\textsuperscript{\dash}\textit{\dash}, which extends the simple intuitive structure of DATALOG\textsuperscript{\dash} (Datalog with stratified negation \cite{23}) with other two types of ‘controlled’ negation. In more detail, the control is made by embedding the non-stratified negation into: (i) rules with (exclusive) disjunctive heads, and (ii) constraint rules. Consider again the \textit{Vertex Cover} problem:
Example 2. The search query of the above example can be expressed as \((P_2, v(X)) \) with \(P_2\) defined as follows:
\[
 v(X) \oplus n v(X) \leftarrow \text{node}(X),
\]
\[
 \Leftrightarrow \text{edge}(X,Y), \neg v(X), \neg v(Y),
\]
where \(\oplus\) denotes exclusive disjunction, whereas the rule with empty head defines a constraint, i.e. a rule which is satisfied only if the body is false. The first rule guesses a partition of node and the second rule is a constraint stating that two connected nodes cannot be outside of the cover (defined by the nodes belonging to \(v\)).

The main contribution of this paper is a precise analysis of the expressive power of non monotonic constructs in Datalog languages and the proposal of a simple and intuitive language where the use of stable model semantics is disciplined to refrain from abstruse forms of unstratified negation and to avoid both undefinedness and unnecessary computational complexity. In more detail, we analyze the semantics and the expressive power of Datalog-like languages and show that \(NP\) search and optimization problems can be expressed by considering Datalog enriched with exclusive disjunction plus stratified negation or constraints.

2. Preliminaries
We assume the reader is familiar with the basic terminology and notation of relational databases and of database queries [1, 23].

2.1 Relational Databases
We consider an alphabet of relation symbols and attribute symbols. Relation symbols have a fixed arity, whereas attribute symbols have associated finite domains. The domain of an attribute \(A\) is denoted by \(\text{dom}(A)\) and all attribute domains are contained into the database domain \(\text{Dom}\). Every relation \(R\) has associated a schema \(Rs = p(A_1, \ldots, A_n)\), where \(p\) is a relation symbol with arity \(n\) and \(A_1, \ldots, A_n\) are attribute symbols, and an instance of \(R\) is any subset of \(\text{dom}(A_1) \times \cdots \times \text{dom}(A_n)\). A database schema \(DS\) is a finite set of relation schemas. A database (instance) over \(DS\) is any set of finite relations \(D = \{R_1, \ldots, R_k\}\) where each relation \(R_i\) is an instance over the relation schema \(Rs_i\). Given a set of relations \(D\) and a relation symbol \(p\), \(\text{Dom}[p]\) denotes the set of \(p\)-tuples in \(D\).

A (relational) query over a database defines a function from the database to a relation. Queries can be expressed by means of alternative equivalent languages such as relational algebra (RA), ‘safe’ first order formulas (FO) or ‘safe’ non recursive Datalog [23]. In the following we shall use FO formulas. A (relational) query is a tuple of the form \(Q = \langle DS, \phi, f \rangle\) where \(DS\) is database schema, \(f\) is a relation symbol (output relation) with schema \(F_o = f(B_1, \ldots, B_m)\) and \(\phi\) is a quantified first order formula over \(DS \cup \{F_o\}\), i.e. a formula of the form \(\forall X \langle (\exists Y) \phi(DS, F_o, X, Y) \rangle\). The application of a query \(Q = \langle DS, \phi, f \rangle\) to a database \(D\) over \(DS\), denoted as \(Q(D)\), gives a relation \(f\) over \(F_o\) satisfying \(\phi\), i.e. \(F \subseteq \text{Dom}(D)^{|F|} \land D \cup \{F\} \models \phi(D,F)\).

2.2 Search and Optimization Queries
Given a finite alphabet \(\Sigma\) with at least two elements, a partial multivalued (MV) function \(f : \Sigma^* \rightarrow \Sigma^*\) associates zero, one or several outputs to each input string. Let \(f(x)\) stands for the set of possible results of \(f\) on an input string \(x\); thus, we write \(y \in f(x)\) if \(y\) is a value of \(f\) on the input string \(x\) and \(\text{dom}(f) = \{x \mid \exists y \in f(x)\}\). If \(x \notin \text{dom}(f)\), we will say that \(f\) is undefined at \(x\). A computable (i.e., partial recursive) MV function \(f\) is computed by some Turing transducer, i.e. a (deterministic or not) Turing machine \(T\) which, in addition to accept any string \(x \in \text{dom}(f)\), writes a string \(y \in f(x)\) on an output tape before entering the accepting state. So, if \(x \notin \text{dom}(f)\), the set of all strings that are written in all accepting computations is \(f(x)\); on the other hand, if \(x \notin \text{dom}(f)\), \(T\) never enters the accepting state.

Given a database scheme \(DS\) and an additional relation symbol \(f\) (the query goal), a search query \(Q\) is a (possibly partial) multivalued recursive function which maps every database \(D\) on \(DS\) to a finite, none-empty set of finite (possibly empty) relations \(F \subseteq \text{Dom}(D)^{|F|}\) and is invariant under an isomorphism on \(U \rightarrow W\), where \(W\) is any finite subset of \(U\) (i.e. the function is \(W\)-generic). \(Q(D)\) yields a set of relations on the goal, that are the answers of the query; the query has no answer if this set is empty or the function is not defined on \(D\).

Thus, a search query is a computable MV function where the input \(x\) is a suitable encoding of a database \(D\) and each output string \(y\) encodes an answer of the query. Search queries correspond to functions rather than to languages as it instead happens for boolean queries (for a more comprehensive description of this topic see [22, 6]).

In classifying query classes we shall refer to the following complexity classes of languages: the class \(PTIME\) (languages that are recognized by deterministic Turing machines in polynomial time), the class \(NP\) (languages that are recognized by nondeterministic Turing machines in polynomial time), and the class \(coNP\) (the complement of \(NP\)) — the reader can refer to [10, 20] for excellent sources of information on this subject. The class \(NP\) search problems is also denoted by \(NPVM\) [22]. By analogy, \(PMV\) and \(coPMV\) denote the classes of search problems corresponding, respectively, to \(PTIME\) and in \(coNP\). \(NPMV\) (resp., \(QPMV\) and \(coQPMV\)) is the class of all search queries which are in \(NPVM\) (resp., \(PMV\) and \(coPMV\)).

Definition 1. A search query \(Q\) is a triple \(\langle DS, \phi, f \rangle\) where \(DS\) is the database schema, \(f\) is the output relation and \(\phi\) is a quantified first order relation expressing the search problem on \(DS\).

It has been shown that a query \(Q = \langle DS, \phi, f \rangle\) is in \(NQPMV\) (resp., \(QPMV\) and \(coQPMV\)) if and only if for each database \(D\) on \(DS\) and for each relation \(F\) on \(f\), deciding whether \(F\) is in \(Q(D)\) is in \(NP\) (resp., in \(PTIME\) and in \(coNP\)).

Fact 1. Let \(Q = \langle DS, \psi, f \rangle\) be a search query in \(NQPMV\), then there exists a sequence \(s\) of relation symbols \(s_1, \ldots, s_k\) distinct from those in \(DS \cup \{f\}\) and a closed first order formula \(\phi\) over \(DS\) and \(s\) such that for each database \(D\) on \(DS\), \(Q(D) = \{F : F \subseteq \text{Dom}(D)^{|F|}, S_i \subseteq \text{Dom}(D)^{|F|} (1 \leq i \leq k), \phi(D,F,S)\text{ is true}\}\).

From now on, we shall formulate a search query in \(NQPMV\) as \(Q = \{DS, \psi, f\}\) where \(\psi\) is a quantified formula over \(DS\),
and $s$. For instance, the search query of Example 1 can be defined as follows:

$$\{ v : (DS_G, v) \models (\forall x, y)[\text{edge}(x, y) \supset (v(x) \lor v(y))] \}.$$  

The result of the application of $Q$ to a database $D$ over $DS$ is $Q(D) = \{ F : (D \cup \{ F \} \cup S) \models \psi(D, F, S) \}$. With respect to the definition given in Subsection 2.1, we are here considering more powerful queries as the formula $\psi$ is defined over $DS$, $f$ and an additional set of relation symbols $s$.

In this paper we also deal with the issue of formulating optimization problems. We recall that an optimization (min or max) problem, associated to a search problem $f$, is a function $g$ such that $\text{dom}(g) = \text{dom}(f)$ and for each $x \in \text{dom}(g)$, $g(x) = \{ y \mid y \in f(x) \text{ and for each other } y' \in f(x), |y| \leq |y'| \} \text{ (or } |y| \geq |y'| \text{ if is a maximization problem)}$. The optimization problems associated to $NP$ search problems are called $NP$ optimization problems.

More formally, given a search query $Q = (DS, \phi, f)$, an optimization query $Q_O = \text{opt}(Q) = (DS, \phi, \text{opt}(f))$, where $\text{opt}$ is either $\text{max}$ or $\text{min}$, is a search query refining $Q$ such that for each database $D$ on $DS$ for which $Q$ is defined, $Q(D)$ consists of the answers in $Q(D)$ with the maximum or minimum (resp., if $\text{opt} = \text{max}$ or $\text{min}$) cardinality.

The Min Set Cover problem of Example 1 can be defined as follows:

$$\text{min}\{ x : (DS_G, v) \models [(v(x, y) \supset (v(x) \lor v(y)))] \}.$$  

Observe that, for the sake of simplicity, we are considering queries computing the maximum or minimum cardinality of the output relation, but we could consider any polynomial function. The query $Q$ is called the search query associated to $Q_O$ and the relations in $Q(D)$ are the feasible solutions of $Q_O$. The class of all optimization queries is denoted by $OPT^*$.

The class of all queries whose search queries are in $QC$ is denoted by $OPT^* QC$. The queries in the class $OPT^* NQP$ are called $NP$ optimization queries.

### 2.3 Datalog

A (logic) program $P$ is a finite set of rules $r$ of the form $H(r) \leftarrow B(r)$, where $H(r)$ is an atom (head of the rule) and $B(r)$ is a conjunction of literals (body of the rule). A DATALOG program is a logic program without functions symbols. We assume that programs are safe [23], i.e variables appearing in the head or in negated body literals are range restricted as they appear in some positive body literal. A ground rule with empty body is called a fact. The ground instantiation of $P$ is denoted by $\text{ground}(P)$; the Herbrand universe and the Herbrand base of $P$ are denoted by $U_P$ and $B_P$, respectively. A program $P$ is said to be safe if it is negation free. The semantics of a positive program $P$ is given by the unique minimal model $MM(P)$, which can be computed by applying the immediate consequence operator $T_P$ until the fixpoint is reached, i.e. $MM(P) = T_P^\infty(\emptyset)$. The semantics of programs with negation $P$ is given by the set of its stable model $SM(P)$. An interpretation $I$ is a stable model of $P$ if $I$ is the unique minimal model of the positive program $P^*$; $P^*$ denotes the positive logic program obtained from $\text{ground}(P)$ by (i) removing all rules $r$ such that there exists a negative literal $\neg A$ in $B(r)$ and $A$ is in $I$, and (ii) by removing all negative literals from the remaining rules [7]. It is well-known that a program may have $n$ stable models with $n \geq 0$.

Given a program $P$ and two predicate symbols $p$ and $q$, we write $p \rightarrow q$ if there exists a rule where $q$ occurs in the head and $p$ in the body or there exists a predicate $s$ such that $p \rightarrow s$ and $s \rightarrow q$. A program is stratified if there exists no rule where a predicate $p$ occurs in a negative literal in the body, $q$ occurs in the head and $q \rightarrow p$, i.e. there is no recursion through negation [2]. Stratified programs have a unique stable model which coincides with the stratified model, obtained by partitioning the program into an ordered number of suitable subprograms (called ‘strata’) and computing the fixpoints of every stratum in their order [2].

Predicate symbols can be either extensional (i.e. defined by the ground facts of a database — $EDB$ predicate symbols) or intensional (i.e. defined by the rules of the program $IDB$ predicate symbols). The class of all DATALOG programs is simply called DATALOG; the subclass of all positive (resp., stratified) programs is called $\text{DATALOG}^+$ (resp. $\text{DATALOG}^{\rightarrow\leftarrow}$) [1].

A DATALOG program $P$ has associated a relational database scheme $DS$, which consists of all $EDB$ predicate symbols of $P$. We assume that possible constants in $P$ are taken from the database domain. Given a set of ground atoms $M$ and an atom $g(t), M[g(t)]$ (resp. $M(g(t))$) denoted the set of $t$-tuples (resp. tuples matching $g(t)$ in $M$).

A DATALOG program $Q$ is a pair $(P, g(t))$ where $P$ is a DATALOG program and $g(t)$ an atom. The application of a query $Q$ to a database $D$ is denoted by $Q(D)$ and the union of the program $P$ and the facts in $D$ is denoted by $P_D$. Clearly, all models for $P_D$ contain the database $D$. The result of a query $Q = (P, g(t))$ applied to an input database $D$ is defined in terms of the stable models of $P_D$, by taking either the union (possible inference) or the intersection (certain inference) of all models. Thus, given a program $P$ and a database $D$, a ground atom $g(t)$ is true, under possible (brave) semantics, if there exists a minimal model $M$ for $P_D$ such that $g(t) \in M$. Analogously, $g(t)$ is true, under certain (cautious) semantics, if $g(t)$ is true in every minimal model for $P_D$.

### 3. DATALOG SEARCH AND OPTIMIZATION QUERIES

Search and optimization problems can be expressed using different logic formalisms such as Datalog with unstratified negation.

**Definition.** A DATALOG search query is a pair $Q = (P, g(t))$, where $P$ is a DATALOG program and $g(t)$ is an $IDB$ atom of $P$. The (non-deterministic) answer to a search query $Q$ applied to a database $D$ is $Q(D) = \{ M[g(t)] : M \text{ is a stable model of } P_D \}$.

The answer to a DATALOG optimization query $\text{opt}(Q) = (P, \text{opt}(g(t)))$ over a database $D$ is equal to $\text{opt}(\{S|S \text{ is a possible answer of } Q(D)\})$. 

\[ \square \]
It is worth noting that, as for search queries, also for optimization queries we output the relation with the optimal cardinality rather than just the cardinality. Thus, given a search query \( Q = (P, g(t)) \) over a database \( D \), the output relation \( Q(D) \) consists of all tuples \( g(u) \) matching \( g(t) \) and belonging to a stable model \( M \) of \( P_D \), selected nondeterministically. For a given optimization query \( OQ = (P, \min[g(t)]) \) (resp. \( (P, \max[g(t)]) \)), the output relation \( OQ(D) \) consists of the set of tuples \( g(u) \) matching \( g(t) \) and belonging to a stable model \( M \) of \( P_D \), selected nondeterministically among those which minimize (resp. maximize) the output relation. An example of \( \text{DATALOG} \) search query is shown in Example 1; the optimization problem is expressed by rewriting the query goal as \( \langle P_1, \min[v(X)] \rangle \) whose meaning is to further restrict the set of suitable stable models to those for which the subset of selected nodes is minimum. The set of all \( \text{DATALOG} \), \( \text{DATALOG}^+ \) or \( \text{DATALOG}^- \) queries are denoted respectively by \( Q^+, Q^-, Q^0 \). A given class of queries \( Q \), the corresponding sets of search queries are denoted, respectively by \( \text{search}(Q) \) and \( \text{opt}(Q) \). Thus, \( \text{search}(Q^+) \) (resp. \( \text{search}(Q^-) \), \( \text{search}(Q^0) \)) denotes the set of all \( \text{DATALOG}^+ \) (resp. \( \text{DATALOG}^- \), \( \text{DATALOG}^0 \)) search queries, whereas \( \text{opt}(Q^+) \) (resp. \( \text{opt}(Q^-) \), \( \text{opt}(Q^0) \)) denotes the set of all \( \text{DATALOG}^+ \) (resp. \( \text{DATALOG}^- \), \( \text{DATALOG}^0 \)) optimization queries. Observe that, given a database \( D \), if the program \( P_D \) has no stable models, then both search and optimization queries are not defined on \( D \).

**FACT 2.** 1. \( \text{search}(\text{DATALOG}^-) = \mathcal{NPQM}^\forall \),

2. \( \text{opt}(\text{DATALOG}^-) = \text{OPT}\mathcal{NPQM}^\forall \).

The previous fact, firstly proved in [9], states that \( \text{DATALOG}^- \) captures the classes of \( \mathcal{NP} \) search and optimization queries.

### 4. DATALOG\(^-\),\(^\oplus\),\(^\ominus\)

The problem in using \( \text{DATALOG}^- \) to express search and optimization problems is that the unrestricted negation in programs is often neither simple nor intuitive and, besides, it does not allow to discipline the expressive power. This situation might lead to write queries having no (total) stable models or whose computation is hard even though the problem is not. In order to avoid this problem, we consider restricted forms of negation forcing the user to write programs in a more disciplined way without loss of expressive power.

#### 4.1 Syntax

In this section we consider the language \( \text{DATALOG}^-\oplus,\ominus \) which extends \( \text{DATALOG}^-\) with two simple forms of unstratified negation embedded into built-in constructs: head disjunction and constraints (here denoted by \( \oplus \) and \( \ominus \), respectively).

An *(exclusive)* disjunctive rule is of the form:

\[
\begin{align*}
p_1(X_1) \oplus \cdots \oplus p_k(X_k) & \leftarrow Body(X) \\
\end{align*}
\]

where \( (i) \) \( Body(X) \) is a conjunction of literals, \( (ii) \) \( X \) is a vector of range restricted variables, and \( (iii) \) \( X_i \subseteq X \) for all \( i \in [1..k] \). The intuitive meaning of such a rule is that if \( Body(X) \) is true, exactly one head atom \( p_i(X_i) \) must be true.

We also allow a special form of disjunctive rule, called *generalized disjunctive rule* of the form:

\[
\begin{align*}
\oplus L \ p(X, L) & \leftarrow Body(X, Y, L) \\
\end{align*}
\]

in which the number of disjunctions is not defined at compile-time, but depends on the database instance and on the current computation (stable model). The intuitive meaning of such a rule is that the relation defined by \( \tau X . Body(X, Y, L) \) is partitioned into a number of subsets equal to the cardinality of the relation \( \tau X . Body(X, Y, L) \). We shall see some examples of generalized disjunctive rules in the next section.

A constraint (rule) is of the form:

\[
\begin{align*}
\ominus X \leftarrow Body(X) \\
\end{align*}
\]

where \( (i) \) \( Body(X) \) is a conjunction of literals, and \( (ii) \) \( X \) is a vector of range restricted variables. A ground constraint rule is satisfied w.r.t. an interpretation \( I \) if the body of the rule is false in \( I \). We shall often write constraint rules using rules of the form: \( A_1 \lor \ldots \lor A_k \leftarrow A_1 \lor \ldots \lor A_k \) to denote a constraint of the form \( \leftarrow A_1, \ldots, A_k \) (i.e. literals are moved from the body to the head)\(^1\). For instance, the constraint \( \ominus x \in Y \lor x \in L \) of Example 2 can be rewritten as \( \ominus x \in L \lor x \in Y \). We shall see some examples of generalized constraint rules in the next sections.

#### 4.2 Semantics

The declarative semantics of a \( \text{DATALOG}^-\oplus,\ominus \) query \( \langle P, g(t) \rangle \) is given in terms of an ‘equivalent’ \( \text{DATALOG}^\forall \) query and stable model semantics. In particular, given a \( \text{DATALOG}^-\oplus,\ominus \) program \( P \), \( st(P) \) denotes the standard \( \text{DATALOG}^\forall \) program derived from \( P \) as follows:

- Every standard rule in \( P \) belongs to \( st(P) \).
- Every disjunctive rule \( \tau X . P \in \mathcal{P} \) of the form (1) is translated into \( k \) rules of the form:

\[
\begin{align*}
p_1(X_1) & \leftarrow Body(X), \neg p_1(X_1), \ldots, \neg p_k(X_k), \\
\end{align*}
\]

with \( i \in [1..k] \).
- Every generalized disjunctive rule of the form (2) is translated into the two rules:

\[
\begin{align*}
p(X, L) & \leftarrow Body(X, Y, L), \text{ diff}_P(X, L), \\
\end{align*}
\]

\[
\begin{align*}
\text{diff}_P(X, L) & \leftarrow Body(X, Y, L), p(X, L'), L' \neq L \\
\end{align*}
\]

where \( \text{diff}_P \) is a new predicate symbol and \( L' \) is a new variable. Here \( \text{diff}_P \) is used to avoid to infer two ground atoms \( p(x, l_1) \) and \( p(x, l_2) \) with \( l_1 \neq l_2 \).

\(^1\)The symbol \( \ominus \) denotes inclusive disjunction and is different from \( \oplus \) as the latter denotes exclusive disjunction.
• Every constraint rule \( c \) of the form \( 3 \), is translated into a rule of the form:

\[
c \leftarrow Body(X), \neg c
\]

where \( c \) is a new predicate symbol not appearing elsewhere.

**Definition 4.** Given a \( \text{DATALOG}^\ominus \text{-} \oplus \ominus \) query \( Q = \langle P, g(t) \rangle \) and a database \( D \), the answer to the query \( Q \) over \( D \) is obtained by applying the \( \text{DATALOG}^\ominus \) query \( st(Q) \) to \( D \), i.e. \( Q(D) = st(Q(D)) \).

For any \( \text{DATALOG}^\ominus \text{-} \oplus \ominus \) search query \( Q = \langle P, g(t) \rangle \) (resp. optimization query \( \text{OQ} = \langle P, opt(g(t)) \rangle \)), \( st(Q) = \langle st(P), g(t) \rangle \) (resp. \( st(\text{OQ}) = \langle st(P), opt(g(t)) \rangle \) denotes the corresponding \( \text{DATALOG}^\ominus \) (optimization) query.

### 4.3 Expressive power

We first analyze the expressive power of \( \text{DATALOG}^\ominus \text{-} \ominus \ominus \) and next sub-languages obtained by considering restricted subsets of built-in constructs.

**Theorem 1.**

1. \( \text{search(\text{DATALOG}^\ominus \ominus \ominus )} = \text{NQPMV} \)

2. \( \text{opt(\text{DATALOG}^\ominus \ominus \ominus )} = \text{OPT NQPMV} \)

Thus, the language \( \text{DATALOG}^\ominus \ominus \ominus \) has the same expressive power of \( \text{DATALOG}^\ominus \). We now analyze the power of languages derived from \( \text{DATALOG}^\ominus \ominus \ominus \) by restricting the number of built-in constructs.

Considering restricted languages derived from \( \text{DATALOG}^\ominus \ominus \ominus \), we recall that \( \text{DATALOG} \) and \( \text{DATALOG}^\ominus \) express a subset of polynomial queries.

Therefore, we have that:

\[
\text{search(\text{DATALOG})} \subseteq \text{search(\text{DATALOG}^\ominus \ominus \ominus )} \subseteq \text{QP.MV} \quad \text{and} \quad \text{opt(\text{DATALOG})} \subseteq \text{opt(\text{DATALOG}^\ominus \ominus \ominus )} \subseteq \text{OPT QP.MV}.
\]

Since, both \( \text{DATALOG} \) and \( \text{DATALOG}^\ominus \) are deterministic, we derive the following corollary:

**Corollary 1.**

1. \( \text{search(\text{DATALOG}^\ominus \ominus \ominus )} = \text{search(\text{DATALOG})} \),

\( \text{opt(\text{DATALOG}^\ominus \ominus \ominus )} = \text{opt(\text{DATALOG})} \),

2. \( \text{search(\text{DATALOG}^\ominus \ominus \ominus \ominus \ominus \ominus \ominus )} = \text{search(\text{DATALOG}^\ominus \ominus \ominus )} \),

\( \text{opt(\text{DATALOG}^\ominus \ominus \ominus \ominus \ominus \ominus \ominus )} = \text{opt(\text{DATALOG}^\ominus \ominus \ominus )} \).

Regarding sub-languages of \( \text{DATALOG}^\ominus \ominus \ominus \) obtained by considering a subset of the constructs extending \( \text{DATALOG} \), we have the following results where \( \text{MIN}(L) \) denotes the set of minimization queries expressible in \( L \) whereas \( \text{MIN NQPMV} \) denotes the subset of minimization problems in \( \text{OPT NQPMV} \):

**Theorem 2.**

1. \( \text{search(\text{DATALOG}^\ominus \ominus \ominus \ominus \ominus \ominus \ominus )} = \text{NQPMV}, \quad \text{opt(\text{DATALOG}^\ominus \ominus \ominus \ominus \ominus \ominus \ominus )} = \text{OPT NQPMV} \).

2. \( \text{search(\text{DATALOG}^\ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \ominus \om
atom denoting the output relation. The optimization query \( \langle P, \text{opt}(g(t)) \rangle \) defines the optimization query \( \text{opt}(Q) \), where \( \text{opt} \in \{ \text{min}, \text{max} \} \).

Also in this case, we can use any polynomial function, but, for the sake of simplicity, we restrict our attention to optimization queries computing the maximum or minimum cardinality of the output relation.

**Theorem 3.**

1. \( \text{search}(\text{NPD} \text{ Datalog}) = \text{NQPMV} \), and
2. \( \text{opt}(\text{NPD} \text{ Datalog}) = \text{OPT}N\text{QPMV} \).

Thus, \( \text{NPD Datalog} \) has the same expressive power of both \( \text{DATALOG}^{\neg, \oplus, \leq} \) and \( \text{DATALOG}^{-} \). The advantage of using a simple language such as \( \text{NPD Datalog} \) is that queries can be easily translated into a different formalism and efficiently executed by means of specialized tools.

In the following we also assume the existence of subset rules, i.e. rules of the form

\[
s(X) \subseteq \text{Body}(X, Y) \tag{6}
\]

where \( s \) is an EDB predicate symbol not defined elsewhere in the program (subset predicate symbol) and \( \text{Body}(X, Y) \) is a safe conjunction of EDB literals. Observe that a subset rule of the above form can be encoded into a partition rule of the form \( s(X) \oplus s(X) \subseteq \text{Body}(X, Y) \). Often we rewrite constraint rules by moving predicates from the body to the head to make them more intuitive.

For instance, the vertex cover program of Example 1 can be rewritten as:

\[
v(X) \subseteq \text{node}(X), \text{edge}(X, Y) \Rightarrow v(X) \lor v(Y).\]

**6. CONCLUSION**

In this paper we have studied the power of Datalog like languages in expressing search and optimization problems. We have analyzed several languages expressing \( \text{NP} \) (decision, search and optimization) problems. We have also proposed a language, called \( \text{NPD Datalog} \), which extends \( \text{DATALOG}^{-} \) allowing head disjunction, only used to define (nondeterministically) partitions of relations, and constraints. This language allows us to express, in a simple and intuitive way, both \( \text{NP} \) search and optimization problems. With respect to other logic languages previously proposed, the novelty of the paper is that we considered languages able to express the complete set of \( \text{NP} \) decision, search and optimization problems, by using restricted form of unstratified negation. We are currently implementing a system prototype which executes \( \text{NPD Datalog} \) queries by translating them into ILOG programs.

**7. REFERENCES**


