The Expressive Powers of Stable Models for Bound and Unbound DATALOG Queries*

Domenico Saccà†

DEIS Department, Università della Calabria, 87030 Rende, Italy

Received August 15, 1994; revised October 26, 1995

Various types of stable models are known in the literature: T-stable (total stable), P-stable (partial stable, also called three-valued stable), M-stable (maximal stable, also known under various different names), and L-stable (least undefined stable).

1. INTRODUCTION

The problem of providing a formal semantics to logic program where rules contain negative literals in their bodies represents an important research issue in areas such as logic programming, non-monotonic reasoning, and deductive databases [2, 7]. An interesting solution has been given by the notion of stable model [17] and its various refinements [6, 13, 36, 38, 40, 46, 48, 49].

The fact that multiple stable models may exist for the same program has caused some conceptual difficulties for accepting stable models as the canonical meaning (i.e., the "intended" models) of a logic program. On the other side, it has been argued that the existence of several alternative stable models whose existence is not guaranteed for every program, certain semantics is taken into account as well. The expressive power of each type of stable model under the above versions of semantics are investigated for both bound (i.e., ground) and unbound queries on DATALOG programs with negation. As deterministic semantics is argued to be inappropriate for unbound queries, a non-deterministic semantics is also proposed for them and its expressive power is fully characterized as well.

but a powerful opportunity that can be exploited in two different directions:

- to express non-determinism in a purely declarative framework—for instance, as shown in [18, 38]—stable models provide a formal declarative semantics to non-deterministic pruning constructs of deductive databases, such as the choice construct of the Datalog language [32];
- to retain a deterministic semantics and use multiplicity only for increasing the expressive power—for instance, it has been illustrated that multiple stable models enable a declarative expression of \( N \neq P \) problems and co-\( N \neq P \) problems [41, 42].

In this paper we show that deterministic semantics for stable models supplies logic programs on finite universes with an expressive power which goes beyond the class \( N \neq P \) or co-\( N \neq P \) of problems. In general, high expressive power of a database query language is not considered an advantage since polynomial-time resolution is not longer guaranteed. But, in our opinion, what is really dangerous is an unexpected exponential time. As shown in [19], the usage of a logic language with stable model semantics can be disciplined so that we are guaranteed a polynomial-time computation, and when needed (for instance, to solve a small instance of a hard problem) we can enable a higher expressive power rather than switch to a general-purpose programming language.

The logic language on finite universes we shall consider is DATALOG\( ^+ \): a DATALOG\( ^+ \) program \( D \) is a logic program that (i) is function-free, (ii) may have negative literals in the rule bodies and (iii) contains a number of predicates (called EDB predicates) that are defined by a finite number of facts corresponding to the tuples in a database.

As "intended" models of a DATALOG\( ^+ \) program, we shall consider four types of stable models: the partial stable (P-stable) models (corresponding to the three-valued stable models of [36] and the strongly founded models of [38]), the total stable (T-stable) models of [17], the maximal stable (M-stable) models of [40] (corresponding to the partial stable models of [38], the preferred extensions of [13],

* Work partially supported by the ECUS033 project “DEUS EX MACHINA: Non-determinism in deductive databases” and by a MURST grant (40% share) under the project “Sistemi formali e strumenti per basi di dati evolute”. An extended abstract of the preliminary results about bound queries appears in the informal proceedings of the Workshop on “Structural Complexity and Recursion-Theoretic Methods in Logic Programming” (Vancouver, October 1993) and an extended abstract of the preliminary results about unbound queries appears in the proceedings of the conference ICDT’95 (Prague, January 1995).

† E-mail: sacca@unical.it.
the regular models of [48, 49], and the maximal stable classes of [6, 49]), and the least undefined stable (L-stable) models of [40]). Moreover, for each type of stable model we analyze two versions of the deterministic semantics [3, 42]: the possible semantics, which takes the union of all stable models of the given type, and the definite semantics, which instead takes their intersection if at least one stable model of the given type exists or takes the empty set otherwise. Note that definite semantics is a variation of classical certain semantics from which it differs only when no stable model of a given type exists. This may happen only for the case of the T-stable models: in this case, certain semantics will make the whole Herbrand base true whereas none is true for definite semantics. Thus, certain semantics declares certain what is not possible!

We shall discuss the expressive power of the four types of stable models under the two versions of deterministic semantics for bound DATALOG \(^\neg\) queries, consisting of a DATALOG \(^\neg\) program and of a ground literal (query goal). Every bound query has associated the set of databases for which the query goal is true according to a given deterministic stable model semantics; therefore, the set of all possible bound queries under a given semantics defines a family of database sets. The expressive power of each stable model semantics is measured in terms of the complexity of recognizing the associated family of database sets. We, therefore, say that the expressive power of a given semantics is DB-C, where C is the Turing-machine complexity class, if the associated family consists of all database sets D that are C-recognizable (i.e., deciding whether a database belongs to D is a problem in C)—equivalently, we shall also say that the given semantics captures (or expresses all queries in) DB-C. Observe that the intrinsic “impedance mismatch” between descriptive and computational complexity [5] does not pose any difficulty in this case as multiplicity of stable models can be used to introduce a desired order on the universe.

In [37] we have shown that, under definite semantics, total stable models express all queries in the class DB-\(\Sigma_2\) \(^p\), corresponding to decision problems that can be formulated as the conjunction of a problem in \(\forall \forall \forall\) and a problem in co.\(\forall \forall \forall\) [35]. Thus definite semantics has an expressive power higher than certain semantics which only captures DB-co.\(\forall \forall \forall\) [41, 42].

A surprising result of the paper is that, under definite semantics, M-stable models capture the class DB-\(\Pi_2\) \(^p\) at the second level of the polynomial hierarchy. Observe that, although M-stable models have been studied by several authors under different names, to the best of our knowledge no characterization of their expressive power has been provided before.

But the greatest expressive power belongs to L-stable models which, besides capturing the class DB-\(\Pi_2\) \(^p\) under definite semantics, gets to the class DB-\(\Sigma_2\) \(^p\) under the possible version. Thus, as many problems in non-monotonic reasoning on finite universes are \(\Sigma_2\) \(^p\) -complete or \(\Pi_2\) \(^p\) -complete [8], DATALOG \(^\neg\) with L-stable model semantics turns out to be a powerful language for expressing non-monotonic reasoning. It is interesting to observe that, in order to let total stable models achieve the same expressive power [14], it is necessary to switch to disjunctive DATALOG \(^\neg\), where a rule head is extended to be a disjunction of atoms. The relevance of L-stable models is confirmed by the fact that L-stable models differ from T-stable models only when a program has no T-stable models at all and thus L-stability is the most appropriate extension of the notion of T-stability to the domain of partial interpretations. This is in a sense surprising as most authors have instead recognized M-stable models as the natural extension of T-stable models.

Furthermore, we shall discuss the expressive power of DATALOG \(^\neg\) queries with a non-ground goal (unbound queries). A nice result is that the expressive power of unbound queries under a given semantics, measured in terms of the complexity of recognizing whether a tuple belongs to the answer of a query, is strongly related to the expressive power of bound queries under the same semantics. We shall also give a characterization in terms of the complexity of recognizing whether a relation consists of all the answer tuples. For completeness we shall also investigate the expressive power of certain semantics for the case of T-stable models and we shall show that certain semantics is less expressive than definite semantics also in the case of unbound queries.

Finally, we shall elaborate our opinion that deterministic semantics for unbound queries is not very effective since it requires one to specify which solution is to be selected for any problem admitting multiple solutions. This in general requires introducing contrived low-level details to single out a unique solution so that a finding problem is eventually transformed into an optimization problem thus increasing the complexity. We therefore propose integrating deterministic and non-deterministic semantics using queries with two goals: a ground goal selects the stable models which have certain properties, and a non-ground one non-deterministically returns the solution computed by any of the selected models. The expressive power of various semantics for such queries (called non-deterministic unbound queries) will be fully characterized as well.

The paper is organized as follows. In Section 2, we introduce the basic definitions and notation on logic programming and we discuss the four types of stable models and the two versions of deterministic semantics for logic programs in general. In Section 3, we concentrate on DATALOG \(^\neg\) and formally define bound DATALOG \(^\neg\) queries as well as their expressive power. In Section 4, we present some sample DATALOG \(^\neg\) queries whose schemes will be used in the proofs of main results. In Section 5 we
2. STABLE MODELS

We assume that the reader is familiar with the basic terminology and notation of logic programming [28]. Non-standard or specific terminology and notation are presented next.

A logic program (or, simply, a program) \( \mathcal{LP} \) is a finite set of rules. Each rule \( r \) of the \( \mathcal{LP} \) has the form

\[
A \leftarrow A_1, \ldots, A_m,
\]

where \( A \) is an atom (the head of the rule) and \( A_1, \ldots, A_m \) are literals (the body of the rule). Let \( H(r) \) and \( B(r) \) represent, respectively, the head of \( r \) and the set of all literals in the body of \( r \). A rule with an empty body is called a fact.

The ground instantiation of \( \mathcal{LP} \) is denoted by ground (\( \mathcal{LP} \)); the Herbrand universe and the Herbrand base of \( \mathcal{LP} \) are denoted by \( U_\mathcal{LP} \) and \( B_\mathcal{LP} \), respectively.

A ground atom \( A \in B_\mathcal{LP} \) and its negation, i.e., the literal \( \neg A \), are said to be the complement of each other. Moreover, if \( B \) is a ground literal, then \( \neg B \) denotes the complement of \( B \).

Let \( X \) be a set of ground literals \( A \) such that either \( \neg A \) is in \( B_\mathcal{LP} \) or \( \neg A \) is in \( B_\mathcal{LP} \). Then \( \neg X \) denotes the set \( \{ \neg A | A \in X \} \), \( X^+ \) (resp., \( X^- \)) denotes the set of all positive (resp., negative) literals in \( X \); moreover, \( \hat{X} \) denotes all elements of \( B_\mathcal{LP} \) which do not occur in \( X \) (i.e., \( \hat{X} = \{ A | A \in B_\mathcal{LP} \) and neither \( A \) nor \( \neg A \) is in \( X \})

Given \( X \subseteq (B_\mathcal{LP} \cup \neg B_\mathcal{LP}) \), \( X \) is a (partial) interpretation of \( \mathcal{LP} \) if it is consistent, i.e., \( X \hat{\hat{X}} \cap \neg X = \emptyset \). Moreover, if \( X^+ \cup \neg X^- = B_\mathcal{LP} \), the interpretation \( X \) is called total.

Given an interpretation \( I \) and a conjunction of \( n (n \geq 0) \) ground literals \( C, C \) is true in \( I \) if every literal in \( C \) is in \( I \), false in \( I \) if there exists some literal \( A \) in \( C \) for which \( \neg A \in I \), and undefined in \( I \) otherwise.

Given an interpretation \( I \) and \( X \subseteq B_\mathcal{LP} \), \( X \) is an unfounded set w.r.t. \( I \) if, for each rule \( r \in \text{ground}(\mathcal{LP}) \) with \( H(r) \in X \), some literal in \( B(r) \) is false in \( I \) or \( B(r) \cap X \neq \emptyset \). Thus, if \( I \cup \neg X \) is an interpretation, for each ground rule \( r \) with \( H(r) \in X \), the body of \( r \) is false in \( I \cup \neg X \) so that no atom in \( X \) can be derived.

The union of all unfounded sets w.r.t. \( I \), which is also an unfounded set w.r.t. \( I \), is called the greatest unfounded set and is denoted by \( \mathcal{U}_{\text{U}(\mathcal{LP})} \).

Given an interpretation \( I \) and \( I \) is founded if \( I^+ = T_{\mathcal{U}_{\text{U}(\mathcal{LP})}}(\emptyset) \), where \( T \) is the classical immediate consequence transformation and \( \mathcal{LP}(I) \) denotes the logic program that is obtained from ground (\( \mathcal{LP} \)) by (i) removing all rules \( r \) such that there exists a negative literal \( \neg A \in B(r) \) and \( \neg A \notin I^- \), and (ii) by removing all negative literals from the remaining rules. Foundness basically prescribes that every positive literal in an interpretation be derived from the rules, possibly using negative literals as additional axioms.

**Definition 2.1.** Let \( \mathcal{LP} \) be a logic program and \( M \) an interpretation of it. Then \( M \) is a P-stable (partial stable) model of \( \mathcal{LP} \) if the following conditions hold:

(a) \( M \) is founded, and

(b) \( \neg M^- = \mathcal{U}_{\text{U}(\mathcal{LP})}(M) \).

Thus an interpretation \( M \) is a P-stable model iff \( M^- \) consists of all derivable ground literals (see condition a) and any ground literal that is granted not to be derivable (i.e., it is in some unfounded set) is included in \( M^- \) (see condition b). As shown in [39], P-stable models correspond to the 3-valued stable models of [36] and the strongly founded models of [38].

We next present subclasses of stable models that are characterized by various criteria of maximality or minimality.

**Definition 2.2.** Given a logic program \( \mathcal{LP} \), a P-stable model \( M \) of \( \mathcal{LP} \) is:

(a) well founded if \( M \) is contained in every P-stable model of \( \mathcal{LP} \);

(b) T-stable (total stable) if \( M \) is a total interpretation;

(c) M-stable (maximal stable) if there exists no P-stable model of \( \mathcal{LP} \) which is a proper superset of \( M \);

(d) L-stable (least-undefined stable) if the set of its undefined atoms is minimal, i.e., no P-stable model \( N \) of \( \mathcal{LP} \) exists such that \( N \) is a proper subset of \( M \).

The well-founded model was first defined in [46] as the least fixpoint of \( W_{\mathcal{LP}}(I) = T_{\mathcal{U}_{\text{U}(\mathcal{LP})}}(I) \cup \neg \mathcal{U}_{\text{U}(\mathcal{LP})}(I) \); the equivalence with definition 2.2, part (a) has been proved in [36, 38, 40]. T-stable models correspond to the stable models of [17], and M-stable models [40] correspond to the (partial) stable models of [38], the preferred extensions of [13] (as proven in [23]), the regular models of [48, 49], and the maximal stable classes of [6, 49]. Finally, L-stable models have recently been proposed in [40].

Let \( \mathcal{F}_{\mathcal{LP}}, \mathcal{W}_{\mathcal{LP}}, \mathcal{M}_{\mathcal{LP}}, \mathcal{T}_{\mathcal{LP}}, \) and \( \mathcal{F}_{\mathcal{LP}} \) be the sets of models of a logic program \( \mathcal{LP} \) that are P-stable, well-founded, M-stable, L-stable, and T-stable, respectively. We shall omit the subscript \( \mathcal{LP} \) whenever it is understood from the context.

**Fact 2.1.** Let \( \mathcal{LP} \) be a program. Then

(a) \( |\mathcal{F}_{\mathcal{LP}}| = 1 \);

(b) \( \emptyset \neq \mathcal{F}_{\mathcal{LP}} \subseteq \mathcal{W}_{\mathcal{LP}} \subseteq \mathcal{M}_{\mathcal{LP}} \subseteq \mathcal{T}_{\mathcal{LP}} \);

(c) \( \mathcal{F}_{\mathcal{LP}} \neq \mathcal{T}_{\mathcal{LP}} \).
Proof. Existence and uniqueness of the well-founded model are well known [46]. The other parts have been proved in [40].

Thus the existence of P-stable, M-stable, and L-stable models but not of a T-stable model, is guaranteed for every program. Moreover, a T-stable model is also L-stable and an L-stable model is also M-stable, but the converse implications do not in general hold. Finally, whenever there exists a T-stable model, the definitions of T-stability and L-stability coincide. Note that there is no finite bound on the number of stable models; the number is obviously finite for programs with finite universes but it can be exponential in the size of the program.

As shown in [40], the set of P-stable models of \( \mathcal{LP} \) forms a non-empty Noetherian lower semilattice w.r.t. the containment relationship. The bottom element is the well-founded model [46] which is the intersection of all P-stable models of \( \mathcal{LP} \) and is the most undefined stable model. The top elements of the semilattice are all the M-stable models, thus they are the P-stable models with a minimal degree of undefinedness w.r.t. set containment. L-stable models are the M-stable models which leave undefined a minimal number of elements of the Herbrand base. The definition of T-stable models is the final step toward a criterion of minimum undefinedness; unfortunately, existence is no longer guaranteed.

Example 2.1. Consider the following program:

\[
\begin{align*}
    b & \leftarrow a(1), \neg c. \\
    c & \leftarrow a(1), \neg b. \\
    p & \leftarrow b, \neg p. \\
    d & \leftarrow a(2), \neg p, \neg e. \\
    e & \leftarrow a(2), \neg p, \neg d. \\
    q & \leftarrow \neg d, \neg q.
\end{align*}
\]

where the predicate symbol \( a \) is defined by the facts: “\( a(1) \)” and “\( a(2) \)”.

The P-stable models are: \( M_1 = \{ a(1), a(2) \} \), \( M_2 = \{ a(1), a(2), b, \neg c \} \), \( M_3 = \{ a(1), a(2), \neg b, c, \neg p \} \), \( M_4 = \{ a(1), a(2), \neg b, c, \neg p, \neg d, e \} \), and \( M_5 = \{ a(1), a(2), \neg b, c, \neg p, d, \neg e, \neg q \} \). \( M_1 \) is the well-founded model; \( M_2, M_3, \) and \( M_5 \) are the M-stable models; \( M_4 \) is also both L-stable and T-stable. The semilattice of P-stable models is shown in Fig. 1a.

Example 2.2. Suppose now that the predicate symbol \( a \) is defined only by the fact “\( a(1) \)”. Then the P-stable models are: \( M_1 = \{ a(1), \neg a(2), \neg d, \neg c \} \), \( M_2 = \{ a(1), \neg a(2), \neg d, \neg c, b, \neg c \} \), and \( M_3 = \{ a(1), \neg a(2), \neg d, \neg c, \neg b, \neg p \} \). \( M_1 \) is the well-founded model; \( M_2 \) and \( M_3 \) are the M-stable models; \( M_4 \) is also L-stable but not T-stable. The semilattice of P-stable models is shown in Fig. 1b.

Each set of stable model, (i.e., \( \mathcal{LP}, \mathcal{WP}, \mathcal{MF}, \mathcal{LP}, \mathcal{WP} \), or \( \mathcal{LF} \)) can be considered as the intended models of a logic program \( \mathcal{LP} \). Let \( \mathcal{A} \) denote a generic set of stable models, i.e., \( \mathcal{A} \) will stand for \( \mathcal{LP}, \mathcal{WP}, \mathcal{MF}, \mathcal{LP}, \mathcal{WP} \), or \( \mathcal{LF} \). For each \( \mathcal{A} \), we next present three versions of deterministic semantics: the possible (or credulous or brave) semantics [3, 42, 14], the certain (or skeptical or cautious) semantics [17, 3, 42, 14], and the definite semantics [37].

**Definition 2.3.** Let \( \mathcal{LP} \) be a logic program and \( A \) be a ground literal. Then

\( A \) is a \( \exists_{\mathcal{LP}} \) (possible) inference of \( \mathcal{LP} \) if \( A \) is true in some model in \( \mathcal{LP} \);

\( A \) is a \( \forall_{\mathcal{LP}} \) (certain) inference of \( \mathcal{LP} \) if \( A \) is true in each of the models in \( \mathcal{LP} \);

\( A \) is a \( \forall!_{\mathcal{LP}} \) (definite) inference of \( \mathcal{LP} \) if both \( \mathcal{LP} \neq \emptyset \) and \( A \) is true in each of the models in \( \mathcal{LP} \);

The difference between certain and definite semantics arises only when \( \mathcal{LP} \) is empty; in this case, any \( A \) is inferred in certain semantics whereas it is not in definite semantics. It turns out that the two semantics differ only for total stable models as the sets of all other stable models are never empty for any program. Therefore, certain semantics will be taken into account only for T-stable models. As shown in [37], the definite semantics for T-stable models has a greater expressive power than certain semantics. In addition, as stated in part (a) of Proposition 2.1 below, definite semantics is consistent with the intuition that it should never be more credulous than possible semantics.

**Proposition 2.1.** Let \( \mathcal{LP} \) be a logic program and \( A \) be a ground literal. Then

\( A \) is a \( \forall!_{\mathcal{LP}} \) inference \( \Rightarrow \) (A is a \( \exists_{\mathcal{LP}} \) inference)

\( A \) is a \( \forall!_{\mathcal{LP}} \) inference \( \Leftrightarrow \) (A is a \( \forall!_{\mathcal{LP}} \) inference)

\( A \) is a \( \forall!_{\mathcal{LP}} \) inference \( \Rightarrow \) (A is a \( \forall!_{\mathcal{LP}} \) inference)
THE EXPRESSIVE POWERS OF STABLE MODELS

445

(d) \((A \in \exists_{\forall^*} \text{ inference}) \Rightarrow (A \in \exists_{\forall^*} \text{ inference}) \Rightarrow (A \in \exists_{\forall^*} \text{ inference})\)

Proof. (a) If \(A \in \exists_{\forall^*} \text{ inference} \) then \(A \) is in every model in \(\mathcal{M} \) and \(\mathcal{M} \) is not empty; so \(A \) is in some model in \(\mathcal{M} \), thus \(A \) is also a \(\exists_{\forall^*} \text{ inference} \).

(b) By part (a) of Fact 2.1, \(\forall^* \) is a singleton and then the two versions of semantics coincide. Moreover, as the well-founded model is \(P\)-stable and is contained in every \(P\)-stable model by definition, \(A \) is a \(\exists_{\forall^*} \text{ inference} \) if and only if \(A \) is a \(\forall^* \text{ inference} \).

(c) The implications immediately follow from part (b) of Fact 2.1.

Example 2.3. Consider the program of Example 2.1. We have that both c and \(-^c \) are \(\exists_{\forall^*} \), \(\exists_{\forall^*} \) inferences but only c is a \(\exists_{\forall^*} \), \(\exists_{\forall^*} \) inference. Actually, c is also a \(\forall^* \text{ inference} \). In the program of Example 2.2, c is a \(\exists_{\forall^*} \), \(\exists_{\forall^*} \) inference but it is not \(\forall^* \text{ inference} \).

3. BOUND DATALOG QUERIES WITH NEGATION

DATALOG\(^\neg\) programs are logic programs with negative literals in the rule bodies but without functions symbols [2, 9, 24, 43]. Some of the predicate symbols (EDB predicates) correspond to database relations on a countable domain \(U \) and do not occur in the rule heads. The other predicate symbols are called IDB predicates. Possible constants in a DATALOG\(^\neg\) program are taken from the domain \(U \).

The DATALOG\(^\neg\) program \(\mathcal{P} \) has an associated relational database scheme \(\mathcal{DS}_{\mathcal{P}} = \{r\}_{r \in \mathcal{P}} \) is an EDB predicate scheme of \(\mathcal{P} \), thus EDB predicate symbols are seen as relation symbols. A database \(D \) on \(\mathcal{DS}_{\mathcal{P}} \) is a set of finite relations, one for each \(r \) in \(\mathcal{DS}_{\mathcal{P}} \), denoted by \(D(r) \). The set of all databases on \(\mathcal{DS}_{\mathcal{P}} \) is denoted by \(\mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \).

Given a database \(D \in \mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \), \(\mathcal{DS}_{\mathcal{P}} \) denotes the following logic program:

\[ \mathcal{LP}_{\mathcal{P}} = \mathcal{LP} \cup \{r(t), \; r \in \mathcal{DS}_{\mathcal{P}} \land t \in D(r) \} \]

The Herbrand universe \(\mathcal{U}_{\mathcal{DS}_{\mathcal{P}}} \) is a finite subset of \(\mathcal{U} \) and consists of all constants occurring in \(\mathcal{LP} \) or in \(D \) (the active domain). If \(D \) is empty and no constant occurs in \(\mathcal{LP} \), then \(\mathcal{U}_{\mathcal{DS}_{\mathcal{P}}} \) is assumed to be equal to \(\{a\} \), where \(a \) is any constant in \(U \).

Definition 3.1. A (bound DATALOG\(^\neg\)) query \(Q \) is a pair \((\mathcal{LP}, G)\), where \(\mathcal{LP} \) is DATALOG\(^\neg\) program and \(G \) is a ground literal (the query goal). Given a database \(D \) in \(\mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \) and a class of stable models \(\mathcal{MF} \), the \(\exists_{\forall^*} \) (resp., \(\forall^* \text{ or } \forall_{\forall^*} \) answer of \(Q \) on \(D \) is true if \(G \) is a \(\exists_{\forall^*} \) (resp., \(\forall^* \text{ or } \forall_{\forall^*} \) inference of \(\mathcal{LP} \) and is false otherwise.

The set of all queries is denoted by \(Q \).

Observe that, in general, two queries \((\mathcal{LP} \cup \mathcal{P}, G)\) and \((\mathcal{LP} \cup \mathcal{P}, \neg G)\) on the same database do not give symmetric answers. Thus, if \((\mathcal{LP} \cup \mathcal{P}, G)\) defines a problem, \((\mathcal{LP} \cup \mathcal{P}, \neg G)\) does not necessarily define the complementary problem.

Definition 3.2. Let \(Q = (\mathcal{LP}, G)\) be a query. Then the database collection of \(Q \) w.r.t. the set of stable models \(\mathcal{MF} \) is:

(a) under the possible version of semantics (\(\exists_{\forall^*} \text{ semantics} \)) the set of all databases \(D \in \mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \) for which the \(\exists_{\forall^*} \text{ answer of } Q \) is true and is denoted by \(\mathcal{DS}_{\mathcal{P}}(Q) \);

(b) under the definite version of semantics (\(\forall^* \text{ semantics} \)), the set of all databases \(D \in \mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \) for which the \(\forall^* \text{ answer of } Q \) is true and is denoted by \(\mathcal{DS}_{\mathcal{P}}(Q) \);

(c) under the certain version of semantics (\(\forall_{\forall^*} \text{ semantics} \)), the set of all databases \(D \in \mathcal{D}_{\mathcal{DS}_{\mathcal{P}}} \) for which the \(\forall_{\forall^*} \text{ answer of } Q \) is true and is denoted by \(\mathcal{DS}_{\mathcal{P}}(Q) \).

The expressive power of a type of semantics (i.e., the kind of stable model and the possible, certain, or definite version) is given by the family of the database collections of all possible queries, i.e., \(\mathcal{DS}_{\mathcal{P}}(Q) \subseteq \mathcal{DS}_{\mathcal{P}}(Q) \subseteq \mathcal{DS}_{\mathcal{P}}(Q) \).

Obviously, \(\mathcal{DS}_{\mathcal{P}}(Q) \subseteq \mathcal{DS}_{\mathcal{P}}(Q) \subseteq \mathcal{DS}_{\mathcal{P}}(Q) \) for all \(\mathcal{MF} \) but \(T\)-stable models; so it will be explicitly computed only in the latter case.

It is well known that the database collection of every query is indeed a generic set of databases [2]. Recall that a set \(\mathcal{D} \) of databases on a database scheme \(\mathcal{DS} \) with domain \(U \) is \((K)\)-generic [10, 2] if there exists a finite subset \(K \) of \(U \) such that for any \(D \in \mathcal{D} \) and for any isomorphism \(\theta \) on relations extending a permutation on \(U = K \), \(\theta(D) \) is in \(\mathcal{D} \) as well—informally, all constants not in \(K \) are not interpreted and relationships among them are only those explicitly provided by the databases. Note that for a query \(Q = (\mathcal{LP}, G) \), a suitable \(K \) consists of all constants occurring in \(\mathcal{DS} \) and in \(G \). From now on, any generic set of databases will be called a database collection.

After the data complexity approach of [10, 47] for which the query is assumed to be a constant while the database is the input variable, the expressive power coincides with the complexity class of the problem of recognizing each query database collection. The expressive power of each semantics will be compared with database complexity classes, denoted as follows. Given a Turing machine complexity class \(C \) (for instance \(P \) or \(NP \)), a relational database scheme \(\mathcal{DS} \), and a database collection \(\mathcal{D} \) on \(\mathcal{DS} \), \(\mathcal{D} \) is \(C\)-recognizable if the
problem of deciding whether \( D \) is in \( D \) is in \( C \). The database complexity class DB-C is the family of all \( C \)-recognizable database collections (for instance, DB-\( \mathcal{P} \) is the family of all database collections that are recognizable in polynomial time). If the expressive power of a given semantics coincides with some complexity class DB-C, we say that the given semantics captures (or expresses all queries in) DB-C.

**Example 3.1.** Consider the program \( \mathcal{D}_P \) of Example 2.1. We have that \( \mathcal{D}_P \subseteq \{ [a] \} \). Suppose that the universe is the set of naturals \( \mathcal{N} \). Then for each finite subset \( K \) of \( \mathcal{N} \), \( \mathcal{D}_P \subseteq \mathcal{D}_K \) contains a database \( D \) such that \( D(a) = K \). Let \( D_m \) and \( D_n \) be the set of all databases in \( \mathcal{D}_P \) for which \( 1 \in D(a) \) and \( 2 \in D(a) \), respectively.

Consider the query \( n = [c, \mathcal{L}, \mathcal{G}] \). We have that \( \mathcal{E}(\mathcal{P}, a) = D_0 \) and \( \mathcal{E}(\mathcal{P}, a) = D \) for each \( \mathcal{X} = \mathcal{P}, \mathcal{G}, \mathcal{P} \wedge \mathcal{G} \). Let \( D_m \) and \( D_n \) be the set of all databases in \( \mathcal{D}_P \) for which \( 1 \in D(a) \) and \( 2 \in D(a) \), respectively.

Let the query \( n = [c, \mathcal{L}, \mathcal{G}] \) be now given. Then \( \mathcal{E}(\mathcal{P}, a) = D_0 \) and \( \mathcal{E}(\mathcal{P}, a) = D \) for each \( \mathcal{X} = \mathcal{P}, \mathcal{G}, \mathcal{P} \wedge \mathcal{G} \). Let \( D_m \) and \( D_n \) be the set of all databases in \( \mathcal{D}_P \) for which \( 1 \in D(a) \) and \( 2 \in D(a) \), respectively.

In Section 5 we shall evaluate the expressive power of P-stable, T-stable, M-stable, and L-stable models for bound queries under the various versions of semantics. To familiarize the reader with the main techniques used in the proofs of the results, in the next section we show some simple examples of DATALOG \( ^\neg \) bound queries whose schemes will often occur in such proofs. We assume that the reader is familiar with the basic notions of complexity classes \([16, 22, 34]\) and of query language complexity evaluation (see, for instance, \([2-4, 10, 14, 21, 24-26, 42, 47]\)).

### 4. Basic Schemes of Bound Queries

In this section we present some examples of DATALOG \( ^\neg \) bound queries which refer to the graph kernel problem defined as: given a directed graph \( G = (\mathcal{V}, \mathcal{E}) \), does there exist a kernel for \( G \), i.e., is there a set \( S \subseteq \mathcal{V} \) of vertices such that both (i) for each \( i \in \mathcal{V} - S \), there does exist a \( j \) in \( S \) for which the edge \( (j, i) \) is in \( \mathcal{E} \), and (ii) for each \( i, j \in S \), \( (i, j) \) is not in \( \mathcal{E} \)? Note that the kernel problem is \( \mathcal{NP} \)-complete \([16]\) and it has been used in \([29]\) to analyze the complexity of deciding the existence of a \( T \)-stable model.

**Example 4.1.** The directed graph in Fig. 2 has two kernels: \([1, 2]\) and \([3, 4]\).

We denote the set of all (finite) directed graphs by \( \mathcal{G} \), the set of all graphs in \( \mathcal{G} \) for which a kernel exists by \( \mathcal{E}_K \), and \( \mathcal{G}_K = \mathcal{G} - \mathcal{E}_K \). Any graph is represented by a database on the database scheme \( \mathcal{D}_G = \{ v, e \} \) where \( v \) and \( e \) store its vertices and edges, respectively.

Consider the following second-order formula over \( \mathcal{D}_G \):

\[
\exists v \forall x [ (-s(x) \land \exists y(s(y) \land e(y, x))] \\
\lor [s(x) \land \forall y(s(y) \rightarrow \neg e(y, x))].
\]

Note that \( \nu \) supplies the interpretation domain of the formula. It is easy to see that a graph \( D \) is in \( \mathcal{G}_K \) iff the formula is satisfied by \( D \). The above formula can be rewritten in the following equivalent Skolem normal form for existential second order formulas:

\[
\exists v \forall x, y \exists v [ (-s(x) \land s(y) \land e(y, x))] \\
\lor [s(x) \land \exists y(s(y) \land \neg e(y, x))].
\]

This formula is next used to construct the following DATALOG \( ^\neg \) program:

**Program \( \mathcal{D}_P \):**

\[
\begin{align*}
r_1: \ & v(w) \leftarrow \nu(w), \neg \delta(w). \\
r_2: \ & \delta(w) \leftarrow \nu(w), s(w). \\
r_3: \ & q(x_1, x_2) \leftarrow \delta(x_1), s(y), e(y, x_1). \\
r_4: \ & q(x_1, x_2) \leftarrow s(x_1), \delta(x_1). \\
r_5: \ & q(x_1, x_2) \leftarrow s(x_1), s(x_2), \neg e(x_2, x_1). \\
r_6: \ & g \leftarrow \neg q(x_1, x_2).
\end{align*}
\]

whose EDB predicate symbols are \( v \) and \( e \), and

\[
\mathcal{D}_P \subseteq \mathcal{D}_G. 
\]

Note that the rules (3)–(5) implement the three conjunctions in the above Skolem normal form formula.

Let any directed graph \( D = (\mathcal{V}, \mathcal{E}) \) be given, say, with \( n \) vertices. A \( T \)-stable model is constructed as follows. The first two rules non-deterministically select two disjoint subsets of \( \mathcal{V} \), say, \( S \) and \( \tilde{S} \) respectively. In fact, a vertex \( v \) is included in \( S \) (i.e., \( s(v) \) is derived to be true) if it is excluded from \( \tilde{S} \) (i.e., \( \delta(v) \) is declared to be false), and conversely. Note that if \( S \) and \( \tilde{S} \) do not cover \( \mathcal{V} \) then, for each \( w \) in \( \mathcal{V} - (S \cup \tilde{S}) \), both \( s(w) \) and \( \delta(w) \) are undefined. For each \( x_1 \) in \( S \), if there exists a vertex \( y \) in \( S \) for which \( (y, x_1) \) is in \( G \) (i.e., \( x_1 \) is
connected to some vertex in $S$) then the third rule makes true $q(x_1, x_2)$ for every $x_2$ in $V$. The fourth rule makes true $q(x_1, x_3)$ for each $x_1$ in $S$ and for each $x_2$ in $N$, and the fifth rule makes true $q(x_1, x_2)$ if both $x_1$ and $x_2$ are in $S$ and the edge from $x_3$ to $x_4$ is not in $G$. Note that $q(x_1, x_3)$ is derived to be true for every $x_1$, $x_2$ in $V$ iff $S$ and $\tilde{S}$ cover $V$ and $S$ is a kernel. But, because of the definition of the unfounded set, $g$ is false iff for every $x_1$, $x_2$ in $V$, $q(x_1, x_2)$ is true; so $g$ is false iff $S$ and $\tilde{S}$ cover $V$ and $S$ is a kernel.

The number of $P$-stable models is equal to $3^n$, that is the number of distinct selections for $S$ and $\tilde{S}$; the number of $T$-stable models is equal to $2^n$, that is the number of distinct selections for $S$ and $\tilde{S}$ such that the two sets cover $V$. Note that an $M$-stable model is also both $T$-stable and $L$-stable, thus the three notions of stability coincide in this case. The well-founded model will make the trivial selection, thus $S = \tilde{S} = \varnothing$. Therefore, for each $x_1, x_2$ in $V$, $s(x_1), s(x_2), q(x_1, x_2)$ are undefined; so $g$ is undefined as well.

For a graph for which a kernel exists, $g$ may be true, false, or undefined in a generic $P$-stable model whereas $g$ may be either true or false in an $M$-stable, $L$-stable or $T$-stable model. Moreover, there exists at least one $P$-stable (and then $M$-stable, $L$-stable, $T$-stable) model which selects a kernel and, therefore, makes $g$ false.

For a graph without kernels, $g$ may be true or undefined in a generic $P$-stable model and $g$ is always true in an $L$-stable, $M$-stable, or $T$-stable model.

**Query $A_2$:** $<LP_n, \neg g>$.

We have that $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$ (i.e., the query defines the graph kernel problem), and $\exp_{LP_n}^P(b_0) = \varnothing$, for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$ (i.e., the query is meaningless). Finally, $\exp_{LP_n}^P(b_0) = \exp_{LP_n}^T(b_0)$, i.e., certain semantics coincides with definite semantics.

**Query $A_3$:** $<LP_n, g>$.

We have that $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$; so, under these semantics, the query defines the co.\!$P$-complete problem: does the graph have no kernels? On the other side, since $\exp_{LP_n}^P(b_0) = \varnothing$ and $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, the query is meaningless in these cases. Again certain semantics coincides with definite semantics, i.e., $\exp_{LP_n}^P(b_0) = \exp_{LP_n}^T(b_0)$.

**Program $LP_n$:** $LP_n \cup \{r_1\}$ where $r_1$ is:

$$r_1: \quad p \leftarrow g, \neg p.$$

A $P$-stable model is constructed in the same way as for $LP_n$ for the first six rules. Because of the seventh rule, $p$ is false iff $g$ is false and is undefined otherwise.

The number of $P$-stable models is again equal to $3^n$; actually, every $P$-stable model of $LP_n$ is also a $P$-stable model of $LP_n$ modulo adding $\neg p$ whenever $g$ is false.

Suppose first that a kernel exists for the graph. Then $L$-stable models are also $T$-stable and an $M$-stable model is $L$-stable iff $g$ is in it. The number of $L$-stable models (and then of $T$-stable models and of $M$-stable models containing $\neg g$) is equal to the number $n_k$ of distinct kernels in the graph, where $1 \leq n_k \leq 2^n$. The total number of $M$-stable models is equal to $2^n$; so $2^n - n_k M$-stable models contain $g$.

Suppose now that no kernel exists for the graph. Then there exists no $T$-table model and every $M$-stable model is also $L$-stable and contains $g$. The total number of $M$-stable models (and of $L$-stable models as well) is equal to $2^n$.

**Query $A_4$:** $<LP_n, g>$.

We have that $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$, and $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$; thus for these semantics the query defines the graph kernel problem. Observe that the problem is expressed by both versions of semantics for $L$-stable and $T$-stable models, definite and possible semantics coinciding.

On the other hand, as $\exp_{LP_n}^P(b_0) = D_n^b$, certain semantics differs from definite semantics and provides no actual meaning to the query. The query is meaningless as well for the definite semantics of $P$-stable and $M$-stable models as $\exp_{LP_n}^P(b_0) = \varnothing$ for $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$.

**Query $A_5$:** $<LP_n, \neg g>$.

We have that $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$, and $\exp_{LP_n}^P(b_0) = D_n^b$; so for these semantics the query defines the complement of the graph kernel problem. Observe that again possible semantics coincides with definite semantics for $L$-stable (but not $T$-stable) models; moreover, queries $A_4$ and $A_5$, give symmetric answers on the same database. The other semantics are meaningless since $\exp_{LP_n}^P(b_0) = D_n^b$ for each $\mathcal{I} = \mathcal{P}_n$, $\mathcal{M}_n$, $\mathcal{L}_n$, $\mathcal{T}_n$, $\mathcal{F}_n$ and $\exp_{LP_n}^P(b_0) = \exp_{LP_n}^T(b_0) = \exp_{LP_n}^F(b_0) = \varnothing$. As $\exp_{LP_n}^P(b_0) = D_n^b$, also in this case the certain and definite semantics are different although both are useless.

**Program $LP_n$:** $LP_n \cup \{r_7, r_8\}$ where $r_7$, and $r_8$ are:

$$r_7: \quad p \leftarrow g, \neg p.$$

$$r_8: \quad \neg g, \neg p \leftarrow g, \neg p.$$

A $P$-stable model is constructed in the same way as for $LP_n$ for the first six rules. The role of the rules $r_7$ and $r_8$ is to invalidate the selections of the first two rules (and, then, the $P$-stable model itself as it looses stability) whenever $g$ is not false (i.e., the selected $S$ and $\tilde{S}$ do not cover $V$ and/or $S$
is not a kernel). In fact, if \( g \) is not false both \( S(w) \) and \( \neg S(w) \) are not unfounded. Obviously, the well-founded model is not invalidated for it does not make any selection.

If a kernel exists for the graph, then the number of \( M \)-stable models is \( n_K \), every \( M \)-stable model is both total and \( L \)-stable, and the number of \( P \)-stable models is \( n_K + 1 \) (i.e., all \( M \)-stable models plus the well-founded model). Otherwise, there exists a unique \( P \)-stable model, i.e., the well-founded model and this model is also \( M \)-stable and \( L \)-stable but not \( T \)-stable.

**Query \( \mathcal{A}_s : \langle \mathcal{L}_P, \neg g \rangle \).**

We have that \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \text{D}_G^P \) for each \( \mathcal{L} = P, \neg P, H, L, F, F \); \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \emptyset \); and \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \text{D}_G^P \) for each \( \mathcal{L} = H, L, F, F \). Thus \( M \)-stable, \( T \)-stable, and \( L \)-stable models define the graph kernel problem under both versions of semantics. Again \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \text{D}_G \); so certain and definite semantics are different.

**Query \( \mathcal{A}_s : \langle \mathcal{L}_P, g \rangle \).**

We have that \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \emptyset \) for each \( \mathcal{L} = P, \neg P, H, L, F, F \), thus the query is meaningless. So it is also for the certain semantics of \( T \)-stable models since \( \text{EXP}_{\neg g}^P(\mathcal{A}_s) = \text{D}_G \).

We stress that the above programs are structured in a general format that corresponds to an immediate implementation of existential second order Skolem normal form formulas. The graph kernel problem as well as any specific problem can actually be formulated with a simpler structure by further exploiting its properties—the next example shows how to improve the programs \( \mathcal{L}_P \) and \( \mathcal{L}_G \).

**Example 4.2.** Consider the following program \( \mathcal{L}_P \):

\[
\begin{align*}
s(W) & \leftarrow v(W), \neg \text{conn_to}_S(W), \neg \text{\#s}(W). \\
\text{\#s}(W) & \leftarrow v(W), \neg s(W). \\
\text{conn_to}_S(X) & \leftarrow s(Y), s(Y, X). \\
\text{not_a_kernel} & \leftarrow \text{\#s}(Y), \neg \text{conn_to}_S(Y).
\end{align*}
\]

Because of the first rule, any two nodes that are included in \( S \) are not connected. Therefore, to check whether \( S \) is a kernel, it is sufficient to verify whether every node not in \( S \) is connected to some node in \( S \) (see the last rule). So there is a kernel iff there is a \( T \)-stable model for which \( \text{not_a_kernel} \) is false.

The program \( \mathcal{L}_P \) is obtained from \( \mathcal{L}_P \) by adding the following rule:

\[
\text{no_kernels} \leftarrow \text{not_a_kernel}, \neg \text{no_kernels}.
\]

Because of the above rule, there exists a \( T \)-stable model iff the graph has a kernel and in every \( T \)-stable model both \( \text{not_a_kernel} \) and \( \text{no_kernels} \) are false.

Finally, we point out that the above program schemes can be combined to define problems in complexity classes higher than \( \text{NP} \) or \( \text{co-NP} \).

**Example 4.3.** Let \( \mathcal{L}_P \) consist of two copies of the program \( \mathcal{L}_P \) of Example 4.2, with two copies of the IDB predicates (adorned with indices 1 and 2, respectively), plus the following three rules:

\[
\begin{align*}
r_1 &: \text{two_sets} \leftarrow s_1(X), \neg s_2(X). \\
r_2 &: \text{two_sets} \leftarrow s_2(X), \neg s_1(X). \\
r_3 &: \text{unique_kernel} \leftarrow \neg \text{two_sets}.
\end{align*}
\]

The query \( Q_1 = \langle \mathcal{L}_P, \text{unique_kernel} \rangle \) under the \( \forall \mathcal{F}_P \) semantics defines the decision problem of whether a graph has a unique kernel—this problem is in the complexity class \( \text{EXP} \) which is located between \( \text{NP} \) and \( \text{PSPACE} \) [22, 34]. In fact, note that (1) if there is no kernel, there are no \( T \)-stable models, so the \( \forall \mathcal{F}_P \) semantics infers nothing; (2) if there is a unique_kernel, then there is just one \( T \)-stable model, in which case \( S_1 = S_2 \) and the \( \forall \mathcal{F}_P \) semantics infers \( \text{unique_kernel} \); and (3) if there are two kernels, then there is a \( T \)-stable model with \( S_1 \neq S_2 \), so the \( \forall \mathcal{F}_P \) semantics does not infer \( \text{unique_kernel} \).

Observe that, under the \( \forall \mathcal{F}_P \) semantics, an \( L \)-stable model exists even though there is no kernel. Therefore, it may happen that \( S_1 \neq S_2 \) and the \( \forall \mathcal{F}_P \) semantics infers \( \text{unique_kernel} \) even though none of the two sets is a kernel. Thus under the \( \forall \mathcal{F}_P \) semantics \( Q_1 \) defines the decision problem of whether a graph has at most one kernel—this problem is in \( \text{co-NP} \). To capture the unique-kernel problem with \( L \)-stable models, we construct the program \( \mathcal{L}_P \) by modifying the rule \( r_1 \) of \( \mathcal{L}_P \) into

\[
\text{unique_kernel} \leftarrow \neg \text{not_a_kernel}, \neg \text{two_sets},
\]

so that the existence of a kernel is necessary to infer \( \text{unique_kernel} \) under the \( \forall \mathcal{F}_P \) semantics. Therefore, the query \( Q_2 = \langle \mathcal{L}_P, \text{unique_kernel} \rangle \) defines the unique-kernel problem under both the \( \forall \mathcal{F}_P \) and the \( \forall \mathcal{F}_P \) semantics.

Finally, note that the query \( Q_2 = \langle \mathcal{L}_P, \neg \text{unique_kernel} \rangle \) defines the complement of the unique-kernel problem under the \( \forall \mathcal{F}_P \) semantics because at least one \( L \)-stable model exists and \( \neg \text{unique_kernel} \) is in every \( L \)-stable model also when there is no kernel. Instead, under the \( \forall \mathcal{F}_P \) semantics, \( Q_2 \) (as well as the query \( \langle \mathcal{L}_P, \neg \text{unique_kernel} \rangle \)) defines the problem of whether the graph has more than one kernel—this problem is in \( \text{NP} \). As a matter of fact, since the complement of the unique-kernel problem is not in \( \text{PSPACE} \) and the expressive power of \( \forall \mathcal{F}_P \) semantics does not go beyond...
\(\mathcal{D}^P\) (see Section 5.2), this problem cannot be expressed by the \(\forall X \mathcal{P}\) semantics (unless \(\mathcal{N}\mathcal{P} = \mathcal{N}\mathcal{P}\)).

5. THE EXPRESSIVE POWERS FOR BOUND QUERIES

5.1. P-Stable Models

The recognition of \(P\)-stable models can be done in polynomial time.

**Fact 5.1.** Given a DATALOG\(^-\) program \(\mathcal{LP}\), a database \(D\) in \(\mathcal{D}_{\mathcal{LP}}\), and an interpretation \(M\) for \(\mathcal{LP}\), deciding whether \(M\) is a \(P\)-stable model for \(\mathcal{LP}\) is in \(\mathcal{P}\).

**Proof.** The size of \(\mathcal{LP}\) is assumed to be constant; so the size of \(M\) is polynomially bound on the size of \(D\). \(M\) is \(P\)-stable if and only if the conditions (a) and (b) of Definition 2.1 are satisfied. To test the condition (a) about foundness it is sufficient to non-deterministically guess an interpretation \(S\). As for the condition (b), we need to compute the greatest unfounded set and also this computation can be done in time polynomial in the size of \(D\) (see [46]). So deciding whether \(M\) is a \(P\)-stable model for \(\mathcal{LP}\) is in \(\mathcal{P}\).

Under the possible semantics, \(P\)-stable models capture \(DB_{\mathcal{N}\mathcal{P}}\).

**Proposition 5.1.** \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q}) = DB_{\mathcal{N}\mathcal{P}}\).

**Proof.** Let us first prove that, given any query \(\mathcal{Q} = (\mathcal{LP}, G)\), in \(\mathcal{Q}\), recognizing whether a database \(D\) is in \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q})\) is in \(\mathcal{N}\mathcal{P}\). \(G\) is a \(\exists \mathcal{LP}\) inference of \(\mathcal{LP}\) if there exists a \(P\)-stable model \(M\) of \(\mathcal{LP}\) such that \(G\) is in \(M\). Therefore, it is sufficient to non-deterministically guess an interpretation \(M\) of \(\mathcal{LP}\) and to check in polynomial time whether (1) \(M\) is \(P\)-stable and (2) \(G\) is in \(M\).

Let us now prove that every \(\mathcal{N}\mathcal{P}\) recognizable database collection \(\mathcal{D}\), say on the database scheme \(\mathcal{D}_{\mathcal{LP}}\), is in \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q})\). To this end, we use Fagin’s result [15] that \(\mathcal{D}\) is defined by an existential second order formula \(\exists \mathcal{R}\Phi(R)\), where \(\mathcal{R}\) is a list of predicate symbols distinct from the ones in \(\mathcal{D}_{\mathcal{LP}}\) and \(\Phi\) is a first-order formula involving predicate symbols in \(\mathcal{D}_{\mathcal{LP}}\) and in \(\mathcal{R}\). As shown in [26], this formula is equivalent to one of the form (second order Skolem normal form) \((\exists S)\Gamma(S)\), where

\[
\Gamma(S) = (\forall X)(\exists Y)(\Theta_1(S, X, Y) \lor \cdots \lor \Theta_\ell(S, X, Y)),
\]

where \(S\) is a superlist of \(\mathcal{R}\) consisting of the predicate symbols \(s_i, 1 \leq i \leq m\); and \(\Theta_1, \ldots, \Theta_\ell\) are conjunctions of literals involving variables in \(X\) and \(Y\) and predicate symbols in \(\mathcal{D}_{\mathcal{LP}}\) and \(S\).

Consider the following program \(\mathcal{LP}\):

\[
\begin{align*}
\sigma_j(W_j) & \leftarrow \neg \delta_j(W_j). \quad (1 \leq j \leq m) \\
\delta_j(W_j) & \leftarrow \neg \sigma_j(W_j). \quad (1 \leq j \leq m) \\
q(X) & \leftarrow \Theta_i(S, X, Y). \quad (1 \leq i \leq k) \\
g & \leftarrow \neg q(X).
\end{align*}
\]

(Note that this program follows the scheme of the program \(\mathcal{LP}\) in Section 4—we would have also used the scheme of the program \(\mathcal{LP}\) or of the program \(\mathcal{LP}\).)

We have that \(\mathcal{D}_{\mathcal{LP}, g} = \mathcal{D}\). We show that \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q}) = \mathcal{D}\) where \(\mathcal{Q} = (\mathcal{LP}, \neg g)\). Let \(D\) be a database on \(\mathcal{LP}\) and assume that there exists a \(P\)-stable model of \(\mathcal{LP}\), say \(M\), for which \(\neg g\) in \(M\). Let \(s = \langle s_1, \ldots, s_m \rangle\) be the relations selected by the first two groups of rules in the construction of \(M\). Since \(\neg g\) is in the greatest unfounded set of \(M\), for each \(x, q(x) \in M\); so, by the third group of rules, there exists constants \(y\) and a conjunction \(\Theta_i\) such that \(\Theta_i(s, x, y)\) is satisfied. Therefore, \(\Gamma(s)\) is satisfied; so \((\exists S)\Gamma(S)\) is satisfied as well and, then, \(D \subseteq D\). Hence, \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q})\).

Under the definite semantics, \(P\)-stable models have the same expressive power as well-founded models, i.e., they only capture a subset of \(DB_{\mathcal{P}}\).

**Fact 5.2.** \(\mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q}) = \mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q}) = \mathcal{E}\mathcal{X}\mathcal{P}_{\mathcal{N}\mathcal{P}}(\mathcal{Q}) \subseteq DB_{\mathcal{P}}\).

**Proof.** The definite semantics for \(P\)-stable models coincides with that of well-founded models by part (b) of Proposition 2.1. Inclusion in \(DB_{\mathcal{P}}\) derives from the fact that the well-founded model can be computed in polynomial time [46]. Finally, it is well known that well-founded semantics only captures a proper subset of \(DB_{\mathcal{P}}\) [2], that is, the same subset captured by the so-called fixpoint queries [10].

As proven in [33], queries on stratified programs [1, 11, 45, 31] with ordered universes capture the whole \(DB_{\mathcal{P}}\); therefore, as the stratified model is indeed well-founded, also the definite \(P\)-stable model semantics captures the whole \(DB_{\mathcal{P}}\) if an ordering is available.
5.2. T-Stable Models

It is known that the recognition of T-stable models can be done in polynomial time [17].

**Fact 5.3.** Given a DATALOG\(\Gamma\) program \(\mathcal{L}\), a database \(D\) on \(\mathcal{L}_{\mathcal{L},\Gamma}\), and an interpretation \(M\) for \(\mathcal{L}_{\mathcal{L},\Gamma}\), deciding whether \(M\) is a T-stable model for \(\mathcal{L}_{\mathcal{L},\Gamma}\) is in \(\mathcal{N}\).

**Proof.** It is sufficient to check whether \(M\) is both total and a P-stable model. By Fact 5.1, the latter check can be done in time polynomial in the size of \(D\). Obviously, also the recognition of whether an interpretation can be total is done in polynomial time. So the recognition is in \(\mathcal{P}\). ●

As pointed out before, the existence of T-stable models is not guaranteed. It is known in the literature [29] that recognizing the existence of T-stable model is an \(\mathcal{N}\)-complete problem. Next we present a proof of this result that uses the program \(\mathcal{L}\) of Section 4.

**Fact 5.4.** Given a query \(\mathcal{L}_{\mathcal{L},\Gamma}\) and a database \(D\) on \(\mathcal{L}_{\mathcal{L},\Gamma}\), deciding whether \(\mathcal{L}_{\mathcal{L},\Gamma}\) has a T-stable model is \(\mathcal{N}\)-complete.

**Proof.** Membership in \(\mathcal{N}\) is clear. To show that the problem is \(\mathcal{N}\)-complete, we take the database collection \(\mathcal{D}\) on \((\mathcal{L},\Gamma)\) of all graphs that have a kernel. Given a database \(D\) on \((\mathcal{L},\Gamma)\) (i.e., a graph), the recognition of whether \(D \in \mathcal{D}\) (i.e., the graph has a kernel) is an \(\mathcal{N}\)-complete problem. Consider the program \(\mathcal{L}_{\mathcal{L},\Gamma}\) of Section 4. As discussed in that section, there exists a T-stable model if and only if the graph has a kernel. So \(\mathcal{N}\)-completeness is proved. ●

Under the possible version of semantics, the expressive power of T-stable models coincides with that of P-stable models.

**Proposition 5.2.** \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) = DB - \mathcal{N}\).

**Proof.** Given any query \(\mathcal{L}_{\mathcal{L},\Gamma}\) and a database \(D\) on \(\mathcal{L}_{\mathcal{L},\Gamma}\), recognizing whether a database \(D\) in \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q)\) since we can non-deterministically guess an interpretation \(M\) of \(\mathcal{L}_{\mathcal{L},\Gamma}\), deciding whether \(M\) is T-stable and (2) \(G\) is in \(M\). To prove that every \(\mathcal{N}\) recognizable database collection \(D\) is in \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q)\), consider the program \(\mathcal{L}\) and the query \(Q\) in the proof of Proposition 5.1. We have shown that \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) \subseteq D\), so, \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) \subseteq D\) by Proposition 2.1. \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) \subseteq D\). Observe now that the P-stable model used in the above proof to show that \(D \in \mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q)\) is indeed total; so \(D \in \mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q)\) as well. Hence, \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) = \mathcal{D}\). ●

As proven in [37], under the definite semantics, the expressive power of T-stable models coincides with the class \(DB - \mathcal{P}\). We recall that a problem is in \(\mathcal{P}\) if it can be expressed as a conjunction of a problem in \(\mathcal{N}\) and a problem in \(\mathcal{co}\mathcal{N}\) [35].

**Fact 5.5.** \(\mathcal{E} \mathcal{P} \mathcal{T}_{\mathcal{L},\Gamma}(Q) = DB - \mathcal{P}\).

**Proof.** See [37].

In Example 4.3 we have shown how to express a problem in the class \(\mathcal{N}\) using definite semantics of T-stable models. We next show how to express a \(\mathcal{P}\)-complete problem.

**Example 5.1.** A well-known \(\mathcal{P}\)-complete problem is the exact clique [22, 34]: given a graph \(G\) and an integer \(k\), is the size of the maximum clique in \(G\) precisely \(k\)? This problem is formulated, under the definite semantics of T-stable models, as follows.

Consider first the following generic program \(\mathcal{L}_{h,\text{clique}}\) (there exists an instance for each value of the index \(i\)):

\[
\begin{align*}
(1) & : c_{i}(X, J) \leftarrow v(X), v(J), \text{size}_i(H), J \notin H, \neg \text{diff}_{i}(X, J) . \\
(2) & : \text{diff}_{i}(X, J) \leftarrow c_{i}(X, J'), J \neq J' . \\
(3) & : \text{diff}_{i}(X, J) \leftarrow c_{i}(J', X), X \neq X' . \\
(4) & : \text{not}_{a_{clique}} \leftarrow c_{i}(X, J), c_{i}(Y, J'), X \neq Y, \neg v(X, Y) .
\end{align*}
\]

The database relations \(v\) (vertices of the graph) and \(e\) (edges) describe the graph. We have assumed that the vertices are numbered from 1 to \(n\) and are, therefore, ordered; this is not a restriction for stable model semantics allows to easily construct an ordering in any case. Assume that size, is true exactly for one value, say \(h\) and \(1 \leq h \leq n\). Then, the rules (1)–(3) select the pairs \((x_1, 1), ..., (x_h, h)\) such that \(x_1, ..., x_h\) are distinct vertices—thus the rules non-deterministically return any subset of \(h\) vertices. Then the rule (4) checks whether the selected subset is an \(h\)-clique. Observe that, using the choice construct of [38], the first three rules can be replaced by the following rule:

\[
\begin{align*}
(5) & : \text{size}_i(K) \leftarrow \text{given}_k(K) . \\
(6) & : \text{no}_{\text{cliques}_K} \leftarrow \text{not}_{\text{a}_{\text{clique}}}, \neg \text{no}_{\text{cliques}_K} .
\end{align*}
\]
The rule (7) non-deterministically selects exactly one \( k' > k \) as the size for \( \mathcal{L} \mathcal{P}_{k' \text{-clique}} \)—this rule is equivalent to the following two rules:

\[
\begin{align*}
&\text{size}_{2}(K') \leftarrow \text{size}_{2}(k'), \text{given}_{k}(K), \text{K'} \text{-} k, \text{choice}((k'),(K')). \\
&\text{exact}_{2 \cdot \text{k-clique}} \leftarrow \text{not} \cdot \text{a-clique}_{2}.
\end{align*}
\]

By the rule (8), \( \text{exact}_{2 \cdot \text{k-clique}} \) is true in every \( T \)-stable model if \( \text{not} \cdot \text{a-clique}_{2} \) is true (i.e., no clique of size \( k' > k \) exists). On the other hand, if there exists no clique of size \( Q \) in every \( T \)-database \( D \) on \( \mathcal{DSLP} \), deciding whether \( M \) is an \( M \)-stable model for \( \mathcal{LP} \) is \( \text{co} \cdot \mathcal{NP} \)-complete. Under the possible semantics, the expressive power of \( M \)-stable models coincides with that of \( P \)-stable models and \( T \)-stable models.

Proposition 5.3. \( \mathcal{EX}_{\mathcal{G}}^{3} \cdot \mathcal{NP}[Q] = \text{DB-co}. \mathcal{NP} \).

Proof. By part (d) of Proposition 2.1, a ground literal \( A \) is a \( \exists_{\mathcal{G}} \) inference iff \( A \) is a \( \exists_{\mathcal{G}} \) inference. Hence, \( \mathcal{EX}_{\mathcal{G}}^{3} \cdot \mathcal{NP}[Q] = \mathcal{EX}_{\mathcal{G}}^{3} \cdot \mathcal{NP}[Q] \). By Proposition 5.1, \( \mathcal{EX}_{\mathcal{G}}^{3} \cdot \mathcal{NP}[Q] = \text{DB-co} \cdot \mathcal{NP} \).

Under the definite semantics, the expressive power of \( M \)-stable models tremendously increases and gets to the second level of the polynomial hierarchy \([30]\).

Theorem 5.2. \( \mathcal{EX}_{\mathcal{G}}^{3} \cdot \mathcal{NP}[Q] = \text{DB-co}. \mathcal{NP} \).
S¹ and S² being two lists of respectively m₁, m₂ predicate symbols, containing all symbols in R¹ and R², respectively. Consider the following program D P.

\[ \begin{align*}
    r₁ : & \quad s_j^1(W^j) \leftarrow \neg s_j^1(W^j) \quad (1 \leq j \leq m₁) \\
    r₂ : & \quad s_j^1(W^j) \leftarrow \neg s_j^2(W^j) \quad (1 \leq j \leq m₁) \\
    r₃ : & \quad s_j^2(W^j) \leftarrow \neg s_j^2(W^j) \quad (1 \leq j \leq m₂) \\
    r₄ : & \quad s_j^2(W^j) \leftarrow \neg s_j^2(W^j) \quad (1 \leq j \leq m₂) \\
    r₅ : & \quad q(X) \leftarrow \Theta_j(S¹, S², X, Y) \quad (1 \leq i \leq k) \\
    r₆ : & \quad g(X) \leftarrow \neg q(X). 
\end{align*} \]

We have that D P = D P. To complete the proof it is sufficient to show that D P = D, where D = (D P, = g). Let D be any database in D P; then each M-stable model of D P contains g. Let M be any M-stable model and S¹ = (s₁¹, ..., s_m¹), S² = (s₁², ..., s_m²) be the relations selected by the first four groups of rules in the construction of M. Since g is the greatest unfounded set of M, for each x, q(x) ∈ M; so, by the group r₅ of rules, there exist constants y and a conjunction \( \Theta_j \) such that \( \Theta_j(s^1, s^2, x, y) \) is satisfied. Therefore, I(s¹, s²) is satisfied; so (\( S^1 \)) I(s¹, S²) is satisfied as well. Hence, to prove that (\( S^1 \)) I(s¹, S²) is satisfied, it is sufficient to show that each possible list of relations s¹, there exists an M-stable model that selects this list through the first two groups of rules. To this end, observe that a P-stable model N cannot be constructed such that it selects s¹ and, besides, no ground atom with symbol in S¹ is left undefined in N. By the definition of the M-stable model, there exists an M-stable model, say N', for which N ⊆ N'. So, N' is an M-stable model that selects s¹; hence, we can replace M with N' in the above argument and we get that (\( S^2 \)) I(s¹, S²) is satisfied for each s¹. It follows that (\( S^1 \)) (\( S^2 \)) I(s¹, S²) holds. Hence D = D and, therefore, D P = D.

Consider now any database D ∈ D. Then for each list of relations s¹, there exists a list of relations s² for which I(s¹, s²) is satisfied. Take any list of relations s¹. Let M be the set of M-stable models that select s¹ through the first two groups of rules; because of the structure of these rules, M is not empty. Moreover, for each s² for which I(s¹, s²) is satisfied, there exists M ∈ M that selects S¹ and, therefore, for each x, q(x) ∈ M by the group r₅ of rules. So \( \neg g \) is in M. We next show by contradiction that every model in M contains \( \neg g \), i.e., D ∈ D P. Suppose not and assume that N ∈ M does not contain \( \neg g \). Then all the ground rules of the groups r₉ and r₁₀ do not have a false body, so the ground rules of the groups r₁ and r₆ cannot be used; therefore, all ground literals with symbol in S¹ are undefined in N. Hence, N ⊆ M—a contradiction with the assumption that N is M-stable. It follows that D ∈ D P. Hence, D P = D and this concludes the proof.

Next we give an example of how M-stable models can express a problem in a complexity class higher than D P. For presentation’s sake, we have chosen a problem in the class \( \mathcal{NP} \), i.e., a problem that can be solved by consulting an \( \forall \mathcal{P} \) oracle a polynomial number of times [22, 34].

**Example 5.2.** The unique maximum clique problem is defined as: given a graph G, is the maximum clique in G unique? This problem is in \( \mathcal{NP} \) but does not seem to be in D P. Note that the problem is not \( \mathcal{NP} \)-complete since it can be solved by consulting an \( \forall \mathcal{P} \) oracle a logarithmic number of times only (e.g., using binary search to find the size of the maximum clique). Also, the complementary problem is in \( \mathcal{NP} \) and is defined, under the \( \forall \mathcal{P} \) semantics, by the query \( \exists \mathcal{P} (\forall \mathcal{P}, \text{no unique max clique}) \) where D P is the program consisting of two instances of the program D P clique (with indices 1 and 2, respectively) of Example 5.1 plus the following rules:

\[ \begin{align*}
    (5) : & \quad \text{size}_1(K) \leftarrow v(K), \text{choice}((,), (K)). \\
    (6) : & \quad \text{size}_1(K') \leftarrow v(K'), \text{size}_1(K), K' \neq K, \\
    & \quad \text{choice}((,), (K')). \\
    (7) : & \quad \text{distinct_clique}_i \leftarrow \neg \text{not_a_clique}_i, \\
    & \quad c_i(X, J), \neg c_i(X, J'). \\
    (8) : & \quad \text{no_un_max_clique} \leftarrow \neg \text{not_a_clique}, \\
    (9) : & \quad \text{no_un_max_clique} \leftarrow \text{distinct_clique}_i, \\
    (10) : & \quad \text{diffc}_i(X, J) \leftarrow \neg \text{no_un_max_clique}, c_i(X, J). 
\end{align*} \]

The rule (5) non-deterministically selects any value \( k \leq n \) for size₁—for presentation’s sake, we have again used the choice construct as in Example 5.1. Therefore, D P clique selects any subset \( C \) of \( k \) vertices and not_a_clique₁ is false iff \( C \) forms a clique. The rule (6) non-deterministically selects any \( k' > k \) as the value for size₁ so that D P clique selects any subset \( C' \) of \( k' \) vertices and not_a_clique₂ is false iff \( C' \) forms a clique. The rule (7) checks whether \( C' \) both is a clique and is distinct from \( C \)—note that, as \( k' \geq k \), \( C' \) cannot be a proper subset of \( C \). If \( C \) is not a clique or \( C' \) is a clique different from \( C \) then we have computed an M-stable model for which no_un_max_clique is true. On the other hand, if \( C \) is a clique and \( C' \) is not another distinct clique, the bodies of the rules (8), (9) are false and, therefore, no_un_max_clique is false as well; hence, the rule (10)
invalidates the “stability” of $C$ so that another selection will be made in the subprogram $L^H_{\text{un}\_\text{max}\_\text{clique}}$. If $C$ happens to be the unique clique, any other selection for $C$ will fail; then, there will be exactly one $M$-stable model that selects $C$ and $\text{no}_{..\text{un}\_\text{max}\_\text{clique}}$ will be undefined in this model. Thus a graph has no unique maximum clique iff $\text{no}_{..\text{un}\_\text{max}\_\text{clique}}$ is true in every $M$-stable model, i.e., for each $C$, either $C$ is not a clique or there exists another clique with equal or greater size. Hence, under the $\forall!\_\exists\forall$ semantics, the query $\exists\langle L^P, \text{no}_{..\text{un}\_\text{max}\_\text{clique}}\rangle$ defines the complement of the unique maximum clique problem.

Observe that the query $\exists\langle L^P, \text{no}_{..\text{un}\_\text{max}\_\text{clique}}\rangle$ does not define the unique maximum clique problem under $\forall!\_\exists\forall$ semantics because of the lack of complementarity between $\forall!$ and $\exists\forall$. Obviously, the problem can be defined under $\forall!\_\forall\exists\forall$ semantics as $A^P \subseteq \Pi^P_0$; however, we have to write the program $L^P$ in a rather different (and more contrived) way.

5.4. L-Stable Models

The recognition of $L$-stable models cannot be done in polynomial time unless $P = \Sigma^P_2$.

**Theorem 5.3.** Given a DATALOG program $L^P$, a database $D$ on $L^P$, and an interpretation $M$ for $L^P$, deciding whether $M$ is an $L$-stable model for $L^P$ is co-$\Sigma^P_2$-complete.

**Proof.** Let $M$ be an interpretation and consider the complementary problem $\overline{M}$: is it true that $M$ is not an $L$-stable model? $\overline{M}$ is in $\Sigma^P_2$ since we can guess an interpretation $N$ and verify in polynomial time that (i) $N$ is $P$-stable and (ii) either $M$ is not $P$-stable or $N$ is a proper subset of $M$. Completeness can now be proven in the same way as for $M$-stable models since $M$-stable models are also $L$-stable models in the program $L^P$ used in the proof of Theorem 5.1.

It is not surprising that, under the definite version of semantics, $L$-stable models capture $DB$-HIT as they are $M$-stable models with an additional constraint on minimal undefinability.

**Theorem 5.4.** $\exists\Sigma^{P}_{2}[Q] = DB$-HIT.

**Proof.** Let us first prove that, given any query $\exists \subseteq \langle L^P, G \rangle$ in $Q$, recognizing whether a database $D$ is in $\exists\Sigma^{P}_{2}[D]$ is in $\Pi^P_2$. Consider the complementary problem $\overline{P}$: is there any $L$-stable model $M$ of $L^P$ that $G \not\subseteq \mathbb{D}^P_M$? To solve this problem, we guess an interpretation $M$ and use an $\exists\Sigma^{P}_{2}$ oracle to ask whether $M$ is not $L$-stable; if the answer is no (i.e., $M$ is indeed $L$-stable), we check in polynomial time whether $G \subseteq M$. Therefore, $\overline{P}$ is in $\Pi^P_2$ and, then, recognizing whether a database $D$ is in $\exists\Sigma^{P}_{2}[D]$ is in $\Pi^P_2$.

Let us now prove that every $\Pi^P_2$ recognizable database collection $D$ on a database scheme $\mathbb{D}^{P}_{2}$ is in $\exists\Sigma^{P}_{2}[Q]$. Then $D$ is definable by a second order formula of the form $\forall V \exists R$ ($\forall R$, $\mathbb{R}$). As shown in the proof of Theorem 5.2, this formula can be put into the Skolem format ($\forall S^1 \exists S^2$) $\mathcal{I}([S^1], [S^2])$, where

$$ \mathcal{I}([S^1], [S^2]) = (\forall X) (\exists Y) (\Theta_1([S^1], [S^2], X, Y) \lor \ldots \lor \Theta_n([S^1], [S^2], X, Y)) $$

Consider the following program $L^P$:

- $\forall r_1: s_j([W^1_j]) [\neg s_i([W^1_j]), (1 \leq j \leq m_1)]$
- $\forall r_2: s_j([W^2_j]) [\neg s_i([W^2_j]), (1 \leq j \leq m_2)]$
- $\forall r_3: q([X]) [\Theta_1([S^1], [S^2], X, Y)]$
- $\forall r_4: s_i([W^1_j]), \neg q([X]), (1 \leq j \leq m_1)$
- $\forall r_5: s_i([W^2_j]), \neg q([X]), (1 \leq j \leq m_1)$

We have that $\mathbb{D}^{P}_{2} = \mathbb{D}^{P}_{0}$. We show that $\exists\Sigma^{P}_{2}[\exists \subseteq \langle L^P, \neg g \rangle]$ is in $\Pi^P_2$. Then each $L$-stable model of $\mathbb{D}^{P}_{0}$ contains $\neg g$. Let $M$ be any $L$-stable model and $s^1, s^2$ be the relations selected by the first four groups of rules in the construction of $M$. By repeating the same argument as in the proof of Theorem 5.2, it is easy to see that ($\exists S^2 \exists \mathcal{I}([S^1], [S^2])$ is satisfied.

To prove that ($\forall S^1 \exists S^2 \mathcal{I}([S^1], [S^2])$ is satisfied, we have to show that, for each $s^1$, there exists an $L$-stable model that selects $s^1$ through the first two groups of rules. To this end, observe that the groups (8), (9) of rules make false all ground atoms $e_i([t])$ for which $s_j([t])$ is in $s^1$ and all ground atoms $e_i([t])$ for which $s_j([t])$ is not in $s^1$; so for each $P$-stable model $M$ selecting $s^1 \neq s^2$, $M \not\subseteq M$ and then there will exist at least one $L$-stable model which selects $s^1$. Therefore, ($\forall S^2 \exists \mathcal{I}([S^1], [S^2])$ is satisfied for each $s^1$. It follows that ($\forall S^1 \exists S^2 \mathcal{I}([S^1], [S^2])$ holds. Hence $D \subseteq \mathbb{D}^{P}_{2}$ and, then, $\exists\Sigma^{P}_{2}[\exists \subseteq \langle L^P, \neg g \rangle]$ is in $DB$-HIT. Consider now any database $D \subseteq \mathbb{D}^{P}_{2}$. Then for each list of relations $s^1$, there exists a list of relations $s^2$ for which $\mathcal{I}([s^1], [s^2])$ is satisfied. Take any list of relations $s^1$. Let $M$ be the set of $L$-stable models that select $s^1$; because of the groups (8), (9) of rules, $M$ is not empty. Moreover, for each $s^2$ for which $\mathcal{I}([s^1], [s^2])$ is satisfied, there exists $M \subseteq M$ that selects $s^2$ and, therefore, for each $x$, $q(x) \in M$ by the group $r_5$ of rules. So $\neg g$ is in $M$. We next show by contradiction that every model in $M$ contains $\neg g$, i.e., $D \subseteq \exists\Sigma^{P}_{2}[\exists \subseteq \langle L^P, \neg g \rangle]$. Suppose not and assume that $N \not\subseteq M$ does not contain $\neg g$. Then, because of the rule (7), $p$ is undefined in $N$. Hence,
$M \in \bar{N}$—a contradiction with the assumption that $N$ is $L$-stable. It follows that $D \in \mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$, i.e., $D \in \mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$. This concludes the proof.

Under the possible version of semantics, the expressive power of $M$-stable models under possible semantics is complementary to the expressive power under definite semantics. Thus, with $L$-stable models, possible semantics also gets to the second level in the polynomial hierarchy.

**Theorem 5.5.** $\mathcal{D}P_{L}^{3}(\mathcal{Q}) = \Sigma_{2}^{P}$.

**Proof.** Let us first prove that, given any query $\varphi \in \mathcal{Q}$, recognizing whether a database $D$ is in $\mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$ is in $\Sigma_{2}^{P}$. To solve this problem, we guess an interpretation $M$ and use an $L$-oracle to ask whether $M$ is not $L$-stable; if the answer is no (i.e., $M$ is indeed $L$-stable), we check in polynomial time whether $G \in M$. Therefore, $\mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$ is in $\Sigma_{2}^{P}$.

Let us now prove that every $\Sigma_{2}^{P}$-recognizable database collection $\mathcal{D}$ on a database scheme $\mathcal{D}$ is in $\mathcal{D}P_{L}^{3}(\mathcal{Q})$. We have that $\mathcal{D}$ is defined by a second order formula of the form $\exists R \forall R \phi(R^{1}, R^{2})$. By setting $\phi(R^{1}, R^{2}) = \neg \phi(R^{1}, R^{2})$, we have that the formula $\exists R \forall R \phi(R^{1}, R^{2})$ defines the database collection $\mathcal{D}$, where $D \in \mathcal{D}$ iff $D$ is the set of all databases on $\mathcal{D}$. Consider the program $\mathcal{L} \mathcal{P}$ and the query $\varphi = (\mathcal{L} \mathcal{P}, \neg g)$ in the proof of Theorem 5.4. We have therein shown that a database $D$ in $\mathcal{D}$ is in $\mathcal{D}P_{L}$ iff $D$ is in $\mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$; hence a database $D$ in $\mathcal{D}P_{L}$ is in $\mathcal{D}P_{L}(\mathcal{A}(\mathcal{D}))$ iff there exists some $L$-stable model $M$ for which $g$ is in $M$. It follows that $\mathcal{D}P_{L}^{3}(\mathcal{A}(\mathcal{D}))$ where $\varphi = (\mathcal{L} \mathcal{P}, g)$.

Next we show that $L$-stable models can express both the unique maximum clique problem and the complementary problem using the same program.

**Example 5.3.** Consider the program $\mathcal{L} \mathcal{P}$ and the query $\varphi$ of Example 5.2. The query $\varphi$ does not define the complement of the unique maximum clique problem under the definite semantics of $L$-stable models. In fact, if the graph has a unique maximum clique, say $C$, then the unique $L$-stable model selecting $C$ will not be $L$-stable because of the criterion of minimal undefinedness: so, no un max clique will be true in every $L$-stable model. To remove this inconvenience, we need to retain an $L$-stable model for each possible $C$; so, we modify $\mathcal{L} \mathcal{P}$ into $\mathcal{L} \mathcal{P}$ by adding the rule

\[\text{(11): store}_{c_{1}}(X, I, K) \leftarrow c_{1}(X, I), \text{ size}_{c_{1}}(K), \neg \text{store}_{c_{1}}(X, I, K),\]

which leaves undefined every store $c_{1}(x, i, k)$, thus allowing it to have at least one $L$-stable model for every $C$. Now the query $\varphi = (\mathcal{L} \mathcal{P}, \text{no}_\ast \text{un}_\ast \text{max}_\ast \text{clique})$ defines the complement of the unique maximum clique problem under

\[\text{definite semantics of both } L\text{-stable and } M\text{-stable models.}\]

Next we further refine $\mathcal{L} \mathcal{P}$ into the program $\mathcal{L} \mathcal{P}_{1}$ to define the unique maximum clique problem using the possible semantics. To this end, we replace the rule (10)

\[\text{(10): diff}_{c_{1}}(X, J) \leftarrow \neg \text{no}_\ast \text{un}_\ast \text{max}_\ast \text{clique}, c_{2}(X, J)\]

with the following rule:

\[\text{(10'): un}_\ast \text{max}_\ast \text{clique} \leftarrow \neg \text{no}_\ast \text{un}_\ast \text{max}_\ast \text{clique}, \text{ un}_\ast \text{max}_\ast \text{clique}.\]

Let $M$ be an $M$-stable model selecting the sets $C$ and $C'$. If $C$ is a clique and $C'$ is not a different clique with equal or greater size, then $\text{un}_\ast \text{max}_\ast \text{clique}$ is false and, because of the rule (10'), $\text{un}_\ast \text{max}_\ast \text{clique}$ is undefined. Therefore $M$ will be $L$-stable iff every other $M$-stable selecting the same set $C$ leaves $\text{un}_\ast \text{max}_\ast \text{clique}$ undefined, i.e., there exists no clique different from $C$ with equal or greater size so that $C$ is indeed the unique maximum clique. Observe that, because of the rule (11), the above $M$-stable model is not comparable for minimal undefinedness with any $M$-stable model selecting a different set $C$. Hence, $\text{un}_\ast \text{max}_\ast \text{clique}$ is undefined in some $L$-stable model iff $\text{no}_\ast \text{un}_\ast \text{max}_\ast \text{clique}$ is true; so the query $\varphi = (\mathcal{L} \mathcal{P}_{1}, \text{un}_\ast \text{max}_\ast \text{clique})$ defines the unique maximum clique problem under the $\exists_{\mathcal{D}}g$ semantics. Finally, observe that, under the definite semantics of $L$-stable models (but not of $M$-stable models), the query $(\mathcal{L} \mathcal{P}_{1}, \text{un}_\ast \text{max}_\ast \text{clique})$ defines the complementary problem. This is another strong advantage of $L$-stable models: the same program can be used to express complementary problems.

**6. THE EXPRESSIVE POWERS FOR NON-BOUND QUERIES**

**6.1. Deterministic Semantics**

In this section we shall study the expressive powers of stable models for non-bound Datalog queries, i.e., the query goals are not ground. For simplicity but without substantial loss of generality, we shall assume that the query goal is an atom and that no term in it is ground.

**Definition 6.1.** An unbound (Datalog) query is a pair $\mathcal{Q} = (\mathcal{L} \mathcal{P}, r(X))$ where $\mathcal{L} \mathcal{P}$ is a Datalog program, $r$ is an IDB predicate symbol occurring in $\mathcal{L} \mathcal{P}$, and $X$ is a list of variables.

Given a database $D$ on $\mathcal{D} \mathcal{P}_{\mathcal{A}(\mathcal{D})}$, the answer of $\mathcal{Q}$ on $D$ under the $3_{\mathcal{D}}g$ (resp., $1_{\mathcal{D}}g$ or $2_{\mathcal{D}}g$) semantics is the (possibly empty) relation $r = \{ t | t \in D \text{ is a } 3_{\mathcal{D}}g \text{ (resp., } 1_{\mathcal{D}}g \text{ or } 2_{\mathcal{D}}g) \text{ inference of } \mathcal{L} \mathcal{P}_{\mathcal{A}(\mathcal{D})} \}$ and is denoted by $\mathcal{Q}^{3_{\mathcal{D}}g}(D)$ (resp., $\mathcal{Q}^{1_{\mathcal{D}}g}(D)$ and $\mathcal{Q}^{2_{\mathcal{D}}g}(D)$).
DEFINITION 6.2. Given a database scheme $\mathcal{D}$ and a relation symbol $r \notin \mathcal{D}$, both on a countable domain $U$, a database mapping from $\mathcal{D}$ to $r$ is a total recursive function which maps every database $D$ on $\mathcal{D}$ to a finite (possibly empty) relation on $r$ and which is $\mathcal{W}$-generic, i.e., it is invariant under an isomorphism on $U - W$, where $W$ is any finite subset of $U$.

Thus, for every stable model semantics, an unbound query $\exists \mathcal{D} \mathcal{F}$ is indeed a database mapping from $\mathcal{D} \mathcal{F} \rightarrow r$ to the query predicate symbol $r$ and it is $\mathcal{W}$-generic, where $W$ is the finite set of constants occurring in $\mathcal{D} \mathcal{F}$. Therefore, the expressive power of stable model semantics coincides with the class of database mappings that are defined by all possible unbound queries. Classes of database mappings can be characterized in terms of the complexity of their recognition. A typical measure of complexity for a database mapping is [20, 10, 2]: given a database mapping $DM : \mathcal{D} \mathcal{F} \rightarrow r$ and a Turing machine complexity class $C$, $DM$ is $C$-recognizable if for each $D$ on $\mathcal{D} \mathcal{F}$, deciding whether a tuple $t$ is in $DM(D)$ can be done in $C$ time.

LEMMA 6.1. Let $DM$ be a database mapping from $\mathcal{D} \mathcal{F}$ to $r$. Then

(a) for each $D$ on $\mathcal{D} \mathcal{F}$, $DM(D)$ is polynomially bound on the size of $D$;

(b) for $C = \mathcal{P}$, $\mathcal{NP}$, co-$\mathcal{NP}$, $\mathcal{D} \mathcal{F}$, $\Sigma^p_2$, or $\Pi^p_2$, $DM$ is $C$-recognizable if and only if deciding whether a relation on $r$ is a (not necessarily proper) subset of $DM(D)$ is in $C$.

Proof. For each $D$, $DM(D)$ is polynomially bound on the arity of $r$ and the size of $D$ and $W$ because of the isomorphism and of the finiteness of $DM(D)$. But, the arity of $r$ and the size of $W$ are constant for each $D$; so the first part of the lemma is proven. As the number of tuples is polynomial bound and $\phi \subseteq DM(D)$ by the totality of $DM$, the recognition problems for every tuple can be combined into a subrelation recognition problem; therefore the second part holds as well.

Part (b) of the above lemma suggests another complexity measure: the complexity of deciding whether a relation is exactly $DM(D)$ and not only a subset of it. More formally, given a database mapping $DM : \mathcal{D} \mathcal{F} \rightarrow r$ and a Turing machine complexity class $C$, $DM$ is $C$-(relation)-recognizable if, for each $D$ on $\mathcal{D} \mathcal{F}$ and $r$ on $r$, deciding whether $r = DM(D)$ can be done in $C$ time. For relation recognition it is not sufficient to verify that all tuples in $r$ are in $DM(D)$; we must also check that any tuple outside $r$ is not in $DM(D)$.

LEMMA 6.2. Let $DM : \mathcal{D} \mathcal{F} \rightarrow r$ be a database mapping. Then

(a) if $DM$ is $\mathcal{P}$-recognizable then it is $\mathcal{P}$-(relation)-recognizable;

(b) if $DM$ is $\mathcal{NP}$ or co-$\mathcal{NP}$-recognizable then it is $\mathcal{D} \mathcal{F}$-(relation)-recognizable;

(c) if $DM$ is $\Sigma^p_2$ or $\Pi^p_2$-recognizable then it is $\mathcal{D} \mathcal{F}$-(relation)-recognizable, where $\Sigma^p_2$ is the class of all problems that can be defined as the conjunction of a problem in $\Sigma^p_1$ and a problem in $\Pi^p_1$;

(d) if $DM$ is $\mathcal{NP}$-recognizable then it is $\mathcal{NP}$-recognizable;

(e) if $DM$ is $\Sigma^p_2$-recognizable then it is $\Sigma^p_2$-recognizable.

Proof. (a–c). Consider the relation $Q$ on $\mathcal{D} \mathcal{F} \times r$ such that the pair $(D, r)$ is in $Q$ if $r \subseteq DM(D)$. We have that $DM$ is $C$-recognizable if the instance-solution problem for $Q$ is in $C$, i.e., deciding whether a pair $(D, r)$ is in $Q$ can be done in $C$ time. Observe that, by Part (2) of Lemma 6.1., $Q$ is polynomially balanced, i.e., for each $(D, r) \in Q$, the size of $r$ is polynomially bound on the size of $D$; moreover, $Q$ is hereditary, i.e., given $(D, r) \in Q$, for any $r^t \subseteq r$, $(D, r^t)$ is in $Q$ in $C$ as well. The maximal instance-solution for $Q$ in $DM(D)$; so the maximal instance-solution problem for $Q$ coincides with the problem of relation recognizability for $DM$. The following results have been proved in [12]:

- if $Q$ is polynomially balanced and hereditary and the instance-solution problem for $Q$ is in $\mathcal{P}$, then the maximal instance-solution problem for $Q$ is in $\mathcal{P}$.

Therefore parts (a) and (b) of the proposition hold. A simple extension of the above results provides the proof for part (c) of the lemma.

(d,e). Suppose that $DM$ is $\mathcal{NP}$-rel-recognizable. Then, by Fagin’s result [15], there exists an existential second order formula $3X \phi \phi \mathcal{D} \mathcal{F} \mathcal{F} \rightarrow r$ that is satisfied for a given $D$ on $\mathcal{D} \mathcal{F}$ and a given $r$ on $r$ if $DM(D) = r$. Consider any tuple $t$ on $r$. We rewrite the above formula as $3X \exists r(t) \wedge \phi$ so that the $r$ is now an IDB symbol. Then, given $D$ on $\mathcal{D} \mathcal{F}$, $t$ is in $DM(D)$ if the new formula is satisfied for $D$. Hence, as the new formula is still an existential second order formula, the recognition of $t$ is in $\mathcal{NP}$ thus $DM$ is $\mathcal{NP}$-recognizable. A similar argument can be used for part (e).

The reverse implications do not in general hold. For instance, deciding whether a relation consisting of a set of nodes represents a kernel for the directed graph (i.e., the input database) is in $\mathcal{P}$, whereas deciding whether a node
Let $\mathcal{DS}$ be a database mapping. Then

(a) $\mathcal{DS}$ is expressible by an unbound query under the $\exists_{\mathcal{DS}}$ semantics, where $\mathcal{DS} = \mathcal{DS}_r$, $\mathcal{IF}$, or $\mathcal{MF}$, if and only if $\mathcal{DS}$ is $NP$-recognizable;

(b) $\mathcal{DS}$ is expressible by an unbound query under the $\exists_{\mathcal{DS}}$ semantics if and only if $\mathcal{DS}$ is $\Sigma_2^p$-recognizable.

**Proof.** Let $\mathcal{D}$ be a database mapping from a database scheme $\mathcal{DF}$ to a relation symbol $r$. In the proof we refer to a generic type $\mathcal{IF}$ of stable models. Let $\mathcal{EXP}_{\mathcal{DF}}[\mathcal{Q}] = DB-C$ be the expressive power of all bound queries under $\exists_{\mathcal{DF}}$ semantics. Observe that $C = \mathcal{NP}$ for $\mathcal{IF} = \mathcal{PF}$, $\mathcal{IF}$, or $\mathcal{MF}$ because of Propositions 5.1, 5.2, and 5.3, respectively; furthermore, by Theorem 5.5, $C = \Sigma_2^p$ for $\mathcal{IF} = \mathcal{LP}$.

{\begin{itemize}
\item *(Only-if part).* Suppose that there exists an unbound query $\mathcal{U}_2 = \langle \mathcal{LP}, r(x) \rangle$ such that $\mathcal{DF}_{r, \mathcal{DF}} = \mathcal{DF}_r$, $r$ is an IDB predicate symbol of $\mathcal{LP}$, and for each $D$ on $\mathcal{DF}$, $\mathcal{DM}(D) = \mathcal{DF}_{\exists_{\mathcal{DF}}}(D)$. For any tuple $t$ on $r$, we construct the bound query $\mathcal{U}_2 = \langle \mathcal{LP}, r(t) \rangle$. Obviously, $r(t)$ is a $\mathcal{IF}_{\mathcal{DF}}$ inference of $\mathcal{LP}_r$ if $t$ is in $\mathcal{U}_2 = \mathcal{DF}_{\exists_{\mathcal{DF}}}(D) = \mathcal{DM}(D)$; so $t$ is in $\mathcal{DM}(D)$ if and only if $t$ is in $\mathcal{DF}_{\exists_{\mathcal{DF}}}(D)$. Hence, as $\mathcal{DF}_{\exists_{\mathcal{DF}}}(D) \in DB-C$, membership of $D$ in $\mathcal{DF}_{\exists_{\mathcal{DF}}}(D)$ can be tested in $C$-time and, then, membership of $t$ in $\mathcal{DM}(D)$ can be tested in $C$-time as well. Thus $\mathcal{DF}$ is $C$-recognizable.

{\begin{itemize}
\item *(If part).* Suppose now that $\mathcal{DF}$ is $C$-recognizable. Consider the database scheme $\mathcal{DF}' = \mathcal{DF} \cup \{ r \}$ and the database collection $\mathcal{D}'$ on $\mathcal{DF}'$ defined as: $\{ D \cup \{ r \} | D$ is any database on $\mathcal{DF}$ and $r \in \mathcal{DM}(D) \}$. By part (b) of Lemma 6.1, $\mathcal{D}'$ is a $C$-recognizable database collection. So there exists a bound query $\mathcal{U}_2' = \langle \mathcal{LP}', G \rangle$ such that $\mathcal{DF}_{r, \mathcal{DF}'} = \mathcal{DF}_r'$ and $\mathcal{EXP}_{\mathcal{DF}'}[\mathcal{U}_2'] = \mathcal{D}'$. Let $\mathcal{LP} = \mathcal{LP} \cup \{ r_1, r_2, r_3, r_4 \}$ where:

\begin{itemize}
\item $r_1$: $r(X) \leftarrow \neg \exists r(X)$.
\item $r_2$: $\exists r(X) \leftarrow \neg r(X)$.
\item $r_3$: $r(X) \leftarrow \neg G, \exists r(X)$.
\item $r_4$: $\exists r(X) \leftarrow \neg G, r(X)$.
\end{itemize}

We have that $\mathcal{DF}_{r, \mathcal{DF}'} = \mathcal{DF}_r$, $r$ is now an IDB predicate symbol and $G$ is a new IDB predicate symbol. Given a database $D$ on $\mathcal{DF}$, the rules $r_1$ and $r_2$ allow us to select a set of ground atoms with $r$ as the predicate symbol, thus they enable the selection of any relation on $r$. But any selection will not eventually make $G$ true will be invalid by the rules $r_3$ and $r_4$; so all and only all subsets of $\mathcal{DM}(D)$ will be eventually selected. Hence, because the union of all subsets of $\mathcal{DM}(D)$ yields $\mathcal{DM}(D)$, we have that, given the unbound query $\mathcal{U}_2 = \langle \mathcal{LP}, r(X) \rangle$, $\mathcal{DF}_{\exists_{\mathcal{DF}}}(D) = \mathcal{DM}(D)$.

The next result shows that the characterization in terms of relation recognizability is less precise.

**Corollary 6.1.** Let $\mathcal{DS}$ be a database mapping. Then

(a) sufficient and necessary conditions for the $\mathcal{DF}$ to be expressible by an unbound query under the $\exists_{\mathcal{DF}}$ semantics are, respectively, that $\mathcal{DF}$ is $NP$-recognizable and $\mathcal{DS}$ is $\mathcal{DF}$-rel-recognizable;

(b) sufficient and necessary conditions for $\mathcal{DF}$ to be expressible by an unbound query under the $\exists_{\mathcal{DF}}$ semantics are, respectively that, $\mathcal{DF}$ is $\Sigma_2^p$-rel-recognizable and $\mathcal{DS}$ is $\mathcal{DF}$-rel-recognizable.

**Proof.** The proofs follow from Lemma 6.2 and Theorem 6.1.

Thus, under relation recognizability and possible semantics, we have singled out a lower bound and an upper bound for the class of database mappings that are expressed by unbound queries for each type of stable model. Observe that each class contains database mappings that are complete for its upper bound but do not cover it; thus inclusions are proven (obviously, provided that the polynomial hierarchy does not collapse).

Next we analyze the expressive powers of unbound queries under definite semantics.

**Theorem 6.2.** Let $\mathcal{DS}$ be a database mapping. Then

(a) $\mathcal{DF}$ is expressible by an unbound query under the $\forall \exists_{\mathcal{DF}}$ semantics (i.e., the well-founded semantics) if and only if $\mathcal{DF}$ is $\mathcal{P}$-recognizable, where $\mathcal{P} = \mathcal{DF}_{\exists_{\mathcal{DF}}}(Q) \subseteq \mathcal{P}$;

(b) $\mathcal{DF}$ is expressible by an unbound query under the $\forall \exists_{\mathcal{DF}}$ semantics if and only if it is $\Pi_2^p$-recognizable.

**Proof.** Let $\mathcal{DS}$ be a database mapping from a database scheme $\mathcal{DF}$ to a relation symbol $r$. In the following we refer to a generic type $\mathcal{IF}$ of stable models, where $\mathcal{IF} = \mathcal{PF}$, $\mathcal{MF}$, or $\mathcal{IF}$. Let $\mathcal{EXP}_{\mathcal{DF}}[\mathcal{Q}] = DB-C$ be the expressive
power of all bound queries under $\forall X \phi$ semantics. Recall that $C = \mathcal{P}$ for $\mathcal{IF} = \mathcal{IF}$ by Fact 5.2, and $C = \mathcal{P}_0$ for $\mathcal{IF} = \mathcal{HF}$ and $\mathcal{IF}$ by Theorems 5.2 and 5.4, respectively.

(Only-if part). Suppose that there exists an unbound query $\mathcal{U} = \langle \mathcal{LP}, r(X) \rangle$ such that $\mathcal{LP}_0 = \mathcal{LP}$, $r$ is an IDB predicate symbol of $\mathcal{LP}$, and for each $D$ on $\mathcal{DP}$, $DM(D) = \mathcal{U}^{|\mathcal{DP}|}(D)$. Given a tuple $t$ on $r$, $r(t)$ is a $\forall X \phi$ inference of $\mathcal{LP}_0$ if $t$ is in $\mathcal{U}^{|\mathcal{DP}|}(D) = DM(D)$, i.e., if $t$ is in $\mathcal{D}$ in $\mathcal{U}^{|\mathcal{DP}|}(D)$ where $\mathcal{D} = \mathcal{U}(\mathcal{LP}, r(t))$. Hence, as $\mathcal{U}^{|\mathcal{DP}|}(D) \in DB-C$, DM is $C$-recognizable.

(If part). Suppose that DM is $C$-recognizable. Consider the database scheme $\mathcal{IF} = \mathcal{IF} \cup \{ r \}$ and the database collection $D'$ on $\mathcal{DP}$ defined as: $\{ D \cup \{ r \} | D \text{ is any database on } \mathcal{IF} \text{ and } r \in DM(D) \}$. By part (b) of Lemma 6.1, $D'$ is a $C$-recognizable database collection. Therefore, there exists a bound query $\mathcal{U} = \langle \mathcal{LP}', g(X) \rangle$ such that $\mathcal{LP}_0 = \mathcal{LP}'$ and $\mathcal{U}^{|\mathcal{DP}|}(D) = D'$. Without loss of generality, we assume that $G$ is equal to an IDB predicate symbol with arity 0, say $g$. Let $k$ be the arity of $g$; then, given a database $D$ on $\mathcal{IF}$ and a $k$-tuple $t$, $t$ is in $DM(D)$ if and only if $\{ t \}$ is in $\mathcal{U}^{|\mathcal{DP}|}(D)$, i.e., $g$ is a $\forall X \phi$ inference in $\mathcal{LP}'_{D-C}(t)$.

We now construct a program $\mathcal{LP}$ as follows:

- the predicate symbols of $\mathcal{LP}$ are the same as those of $\mathcal{LP}$ except for $r$, which becomes an IDB predicate symbol (so the EDB predicate symbols of $\mathcal{LP}$ are those in $\mathcal{DP}$);
- the arity of each IDB predicate symbols is increased by $k$ (in particular, the arity of $r$ and of $g$ become $2 \times k$ and $k$, respectively);
- we add the fact $r(Y, Y)$ so that $r(t, t)$ is true for each $k$-tuple $t$;
- every rule in $\mathcal{LP}$, say $p(X) \leftarrow q_1(X_1), \ldots, q_n(X_n)$, where $C$ is a (possibly empty) conjunction of literals with predicate symbols in $\mathcal{DP}$ and $q_i$ ($1 \leq i \leq n$ and $n \geq 0$) is a predicate symbol in $K$, is modified into the rule $p(X) \leftarrow r(Y, Y), C, q_1(X_1, Y), \ldots, q_n(X_n, Y)$, where $Y$ is a list of $k$ distinct variables not occurring in the rule—because of the new variables, a ground rule for each possible $k$-tuple $t$ is generated.

Given a database $D$ on $\mathcal{DP}$, the ground instance of $\mathcal{LP}$ consists of a subprogram $\mathcal{LP}'$ for each $k$-tuple $t$—the rules of $\mathcal{LP}'$ are those in which the new $k$ variables $Y$ are replaced by $t$. In addition, the fact $r(t, t)$ corresponds to the definition of a database relation $\{ t \}$; so $\mathcal{LP}'$ can be thought of as a labeled copy of the ground instance of $\mathcal{LP}'_0$, where $D'$ is the database $D \cup \{ t \}$ on $\mathcal{DP}$ and the label is the tuple $t$. Hence, $g(t)$ is in an $\mathcal{IF}$ stable model of $\mathcal{LP}_0$, if $g$ is in an $\mathcal{IF}$ stable model of $\mathcal{LP}_0$, and, therefore, $r(t)$ is in $DM(D)$. Since the various subprograms $\mathcal{LP}'$ are independent from each other because of the new $k$ arguments, an $\mathcal{IF}$ stable model for the overall program $\mathcal{LP}_D$ is equal to the union of a $\mathcal{IF}$ stable model of each $\mathcal{LP}'$. As an $\mathcal{IF}$ (for $\mathcal{IF} = \mathcal{HP}$, $\mathcal{HF}$, or $\mathcal{IF}$) stable model exists for any program by (6) of Fact 2.1, we have that, given the unbound query $\mathcal{U} = \langle \mathcal{LP}, g(X) \rangle$, $\mathcal{U}^{|\mathcal{DP}|}(D) = DM(D)$. (Note that for $\mathcal{IF} = \mathcal{IF}$, it may happen that for some $t \notin DM(D)$ there exists no $T$-stable model for the corresponding copy; then $\mathcal{LP}'$ would have no $T$-stable model even though $DM(D)$ is not empty—thus the proof is not directly applicable to $T$-stable models.)

We next characterize the expressive power of $T$-stable models under definite and certain semantics. To give a precise characterization for the case of definite semantics, we need the following notation: given two Turing machine complexity classes $C_1$ and $C_2$, a database mapping $\mathcal{DM} : \mathcal{DP} \rightarrow r$ is $C_1/C_2$-recognizable if there exists a database collection $D$ in $DB-C_1$ and a $C_2$-recognizable database mapping $\mathcal{DM}_r : \mathcal{DP} \rightarrow r$ such that (i) for each $D$ on $\mathcal{DP}$ that is not in $D$, $DM(D) = \emptyset$ and (ii) for each $D \in D$, $DM(D) = DM(D')$. Thus given a database $D$ and a $k$-tuple $t$, $t$ is recognized to be in $DM(D)$ if and only if the following two tests both succeed: (1) test in $C_1$ time whether $D$ is in $D$; (2) test in $C_2$ time whether $t$ is in $DM(D')$. Note that the first test is independent from the tuple being recognized. As an example of this notation, consider a database mapping $\mathcal{DM}$ which is $\mathcal{IF}/co.NP$-recognizable. As $co.NP \subseteq \mathcal{IF}$, $\mathcal{DM}$ is also $\mathcal{IF}$-recognizable but not $co.NP$-recognizable (unless $NP = co.NP$). On the other hand, there exist $\mathcal{IF}$-recognizable database mappings that are not $\mathcal{IF}/co.NP$-recognizable (again, unless $NP = co.NP$).

**Theorem 6.3.** Let $DM$ be a database mapping from a database scheme $\mathcal{DP}$ to a relation symbol $r$. Then

(a) $DM$ is expressible by an unbound query under the $\forall X \phi$ semantics if and only if $DM$ is $co.NP$-recognizable;

(b) $DM$ is expressible by an unbound query under the $\forall X \phi$ semantics if and only if $DM$ is $\mathcal{IF}/co.NP$-recognizable.

**Proof.** (a) It follows the lines of the proof of Theorem 6.2 by replacing definite semantics with certain semantics, and setting $\mathcal{IF} = \mathcal{IF}$ and $C = co.NP$ by Fact 5.6. To let the if-part of the proof hold, we have to select a query $\mathcal{IF}'$ for which a $T$-stable model exists for every database; by Fact 5.7, such a query always exists.
(b) Only-if part. Suppose that there exists an unbound query \( \mathcal{Q} = \langle \mathcal{L}, Q(X) \rangle \) such that \( \mathcal{D}_{\mathcal{L}, \mathcal{Q}} = \mathcal{L} \), \( r \) is an IDB predicate symbol of \( \mathcal{L} \), and for each \( D \) on \( \mathcal{L} \), \( \text{DM}(D) = \mathcal{U}_{\mathcal{Q}, r}(D) \). Let \( D \) be the collection of all databases \( D \) on \( \mathcal{L} \) such that \( \text{DM}(D) \) admits at least one T-stable model; then, for every \( D \) on \( \mathcal{L} \) that is not in \( D \), \( \text{DM}(D) = \emptyset \). Moreover, \( D \) is in \( DB-\mathcal{V} \), as deciding whether \( \text{DM}(D) \) has a T-stable model is in \( \mathcal{NP} \) by Fact 5.4. Let DM' be the database mapping defined by \( \mathcal{Q} \) under certain semantics. By part (a) of this theorem, the DM' is \( \mathcal{NP} \)-recognizable. Moreover, for each \( D \in D \), \( \text{DM}'(D) = \text{DM}(D) \) since definite and certain semantics coincide for a program for which a T-stable model exists. Hence, DM' is \( \mathcal{NP} \)-recognizable and, then, also \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizable for \( \mathcal{NP} \subseteq \mathcal{P} \).

(b) If part. Suppose that DM is \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizable. Then there exists a database collection \( D \) in \( DB-\mathcal{P} \) such that for each database \( D \notin D \), \( \text{DM}(D) = \emptyset \). By Fact 5.5, there exists a bound query \( \mathcal{Q}' = \langle \mathcal{L}, g^* \rangle \) such that \( \mathcal{V}_{\mathcal{Q}, r}(\mathcal{Q}') = D \). Moreover, by the definition of \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizability and by part (b) of Lemma 6.1, there exists a database collection \( D' \) on \( \mathcal{L} \) defined as \( |D \cup \{ r \}| \) is a database on \( \mathcal{L} \) and if \( D \in D \) then \( r \subseteq \text{DM}(D) \), which is in \( DB-\mathcal{P} \). Hence, by Fact 5.6, there exists a bound query \( \mathcal{Q}' = \langle \mathcal{L}, g^* \rangle \) such that \( \mathcal{V}_{\mathcal{Q}, r}(\mathcal{Q}') = D' \). Moreover, by Fact 5.7, we can choose \( D' \) in such a way that it admits a T-stable model for each \( D \in D' \). We modify the program \( \mathcal{L} \) into \( \mathcal{L} \) in the same way as in the proof of the if-part of Theorem 6.2 except for the additional fact \( r(X, X) \) that is replaced by the rule

\[
r(X, X) \leftarrow g^*,
\]

so that \( \mathcal{L} \) is enabled only if \( g^* \) is true. By considering \( \mathcal{L} \) and \( \mathcal{L}' \) we have that \( g^* \) is true if \( D \in D' \); hence, \( \mathcal{U}_{\mathcal{Q}, r}(D) = \text{DM}(D) \), where \( \mathcal{Q} = \langle \mathcal{L}, g^* \rangle \), \( r(X) \). From the proof of part (b) of Theorem 6.3 we can derive the following interesting result: \( \mathcal{NP} \)-recognizability and \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizability coincide.

We now give a partial characterization of definite stable model semantics in terms of relation recognizability.

**Corollary 6.2.** Let DM be a database mapping. Then

(a) the necessary condition for DM, to be expressible by an unbound query under the \( \mathcal{V}_{\mathcal{Q}, r} \) semantics is that DM is \( \mathcal{P} \)-recognizable;

(b) the necessary condition for DM to be expressible by an unbound query under the \( \mathcal{V}_{\mathcal{Q}, r} \) or \( \mathcal{V}_{\mathcal{Q}, s} \) semantics is that DM is \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizable;

(c) the necessary and sufficient conditions for DM to be expressible by an unbound query under the \( \mathcal{V}_{\mathcal{Q}, r} \) or \( \mathcal{V}_{\mathcal{Q}, s} \) semantics are, respectively: DM is \( \mathcal{D}^{\mathcal{P}/\mathcal{V}} \)-recognizable and DM is \( \mathcal{NP} \)-recognizable.

**Proof.** The proofs of necessary conditions immediately follow from Lemma 6.2, Theorem 6.2, and Proposition 6.3. To also see that the sufficient condition of part (c) holds, observe that if DM is \( \mathcal{NP} \)-recognizable then DM is \( \mathcal{NP} \)-recognizable by part (d) of Lemma 6.2; hence, as \( \mathcal{NP} \subseteq \mathcal{P} \), DM is expressible by an unbound query under the \( \mathcal{V}_{\mathcal{Q}, r} \) or \( \mathcal{V}_{\mathcal{Q}, s} \) semantics by part (b) of Theorem 6.2. Having characterized the expressive power of unbound queries under possible, definite and certain semantics of various types of stable models is not satisfactory as, in our belief, any deterministic semantics on unbound queries does not have practical validity. In fact, collecting the tuples of a query answer from a number of distinct models requires a rather awkward and obscure style of writing DATALOG programs; worse, it often happens that a program solving a decision problem cannot be used to solve the associated finding problem, particularly in the case in which it admits multiple solutions.

**Example 6.1.** Take the bound query \( \mathcal{Q} = \langle \mathcal{L}' \rangle \), \( \langle \text{mot}_{\text{a kernel}} \rangle \) of Example 4.2 which, under the possible and definite semantics of T-stable models, defines the graph kernel problem. In order to actually get a kernel, we can just fire the unbound query \( \mathcal{Q} = \langle \mathcal{L}' \rangle \) only when the kernel is not unique, otherwise the query would return the union of all kernels under the possible semantics and the intersection of all kernels under the definite semantics! For the graph of Fig. 2, the results would be \{1, 2, 3, 4\} under the possible semantics and the empty set under the definite semantics.

But our criticism of deterministic semantics for unbound queries is not only motivated by a matter of programming style: we claim that determinism (or even any restricted type of non-determinism such as semideterminism \{44\}) is not appropriate for unbound queries. In fact, as an unbound query often corresponds to a problem with multiple solutions, determinism must introduce into the query some properties to single out exactly one of the possible solutions; such properties typically involve some low-level details (e.g., fixing some order) which are difficult to encode, data dependent, and, besides, they could even increase the complexity of the query. For instance, for the graph kernel problem, we have to specify which kernel is to be returned when the kernel is not unique, e.g., by enforcing the selection of the kernel whose list of vertices is first in some lexicographic order—this corresponds to transforming a finding problem into an optimization problem. In particular, the graph kernel query is no longer \( \mathcal{NP} \)-recognizable.
In the next subsection we resume the potential non-determinism of stable models to provide a simple and efficient formulation of unbound queries as well as the precise characterization of their expressive power.

6.2. Non-deterministic Semantics

To deal with unbound queries we here propose to combine the expressive strength of possible or definite semantics in defining decision problems with the “cut” capability of the intrinsic non-determinism of stable models [18, 38]. Thus we define an unbound query as composed by two goals; the first one is ground and allows to select the stable models in which it is true and the second one is unbound and is to be unified with any of the selected stable models.

**Definition 6.3.** A non-deterministic unbound query \( \mathcal{N} \) is a triple \( \langle \mathcal{L}, G, r(X) \rangle \), where \( \mathcal{L} \) is a DATALOG program, \( G \) is a ground literal, \( r \) is an IDB predicate symbol, and \( X \) is a list of variables.

Given a database \( D \) on \( \mathcal{D}_G \) and an interpretation \( M \) for \( \mathcal{L}_G, r_M \) denotes the relation \( \{ t | r(t) \in M \} \); we say that \( r_M \) is fully defined in \( M \) if for each ground atom \( r(t) \), either \( r(t) \) or \( \neg r(t) \) is in \( M \).

The answer set of \( \mathcal{N} \) for a database \( D \) on \( \mathcal{D}_G \), under the \( \exists_x \mathcal{G}(D) \) (resp. \( \forall_x \mathcal{G}(D) \) or \( \neg \forall_x \mathcal{G}(D) \)) semantics, denoted by \( \mathcal{N}^{\mathcal{G}}(D) \) (resp. \( \mathcal{N}^{\forall_x \mathcal{G}}(D) \) or \( \mathcal{N}^{\neg \forall_x \mathcal{G}}(D) \)), is the empty set if \( G \) is not a \( \exists_x \mathcal{G} \) (resp. \( \forall_x \mathcal{G} \) or \( \neg \forall_x \mathcal{G} \)) inference or otherwise the (possibly empty) set of (possibly empty) relations \( \{ r_M | M \) is a \( \mathcal{G} \)-stable model of \( \mathcal{L}_G, G \in M \), and \( r_M \) is fully defined in \( M \) \}.

In practice it is sufficient to non-deterministically return any relation in the answer set.

Requiring that a relation in the answer set be fully defined is consistent with the fact that the condition for selecting the relation must depend only on the bound query goal, and the possible usage of undefinedness to increase the expressive power can be confined to this goal. Thus the restriction does not reduce the expressive power (except for possibly the \( M \)-stable models under the possible semantics as will be discussed later); moreover, the restriction corresponds to a natural writing of unbound queries.

**Example 6.2.** The mixed query \( \langle \mathcal{L}, g \text{~no~kernel}, s(X) \rangle \) on the program of Example 4.2 with \( \exists_x \mathcal{G} \) or \( \forall_x \mathcal{G} \) semantics filters the \( T \)-stable models corresponding to the selection of a kernel and returns the kernel recognized by any of these models.

**Definition 6.4.** Given a database scheme \( \mathcal{D} \) and a relation symbol \( r \), both with a countable domain \( U \), a database multivalued mapping \( \mathcal{D} \) to a relation \( r \) is a recursive function which maps every database on \( \mathcal{D} \) to a finite (possibly empty) set of finite (possibly empty) relations on \( r \) and is invariant under an isomorphism on \( U \)– \( W \), where \( U \) is the domain of \( \mathcal{D} \) and \( W \) is any finite subset of \( U \).

The database collection defined by \( \mathcal{D} \) is the database collection \( \mathcal{D} \cup \{ r \} \) where \( \mathcal{D} = \{ D \cup \{ r \} \mid D \) is a database on \( \mathcal{D} \) and \( r \) is a relation on \( r \) for which \( r \in \mathcal{D}(D) \).

Thus, a non-deterministic unbound query \( \mathcal{N} = \langle \mathcal{L}, G, r(X) \rangle \) defines a database multivalued mapping \( \mathcal{D} \rightarrow r \).

To characterize the expressive power of a non-deterministic unbound query, the notion of \( C \)-recognizability is not appropriate as two tuples may belong to different relations in the answer set. So we shall use \( C \) -rel-recognizability, thus our measure is based on the complexity of recognizing whether a relation is in the answer set. The recognition of database multivalued mappings is strongly related to the recognition of database collections.

**Lemma 6.3.** A database multivalued mapping \( \mathcal{D} \) is \( C \)-rel-recognizable if and only if the database collection defined by \( \mathcal{D} \) is in \( DB-C \).

**Proof.** Straightforward.

Let us now provide the characterization of non-deterministic queries under the possible semantics.

**Theorem 6.4.** Let \( \mathcal{D} \) be a database multivalued mapping. Then

(a) \( \mathcal{D} \) is expressible by a non-deterministic unbound query under the \( \exists_x \mathcal{G} \) semantics, where \( \mathcal{G} \subseteq \mathcal{L} \), or \( \mathcal{G} \) is, if and only if it is \( \mathcal{N} \)-rel-recognizable;

(b) \( \mathcal{D} \) is expressible by a non-deterministic unbound query under the \( \forall_x \mathcal{G} \) semantics if and only if it is \( \Sigma_2 \)-rel-recognizable.

**Proof.** Let \( \mathcal{D} \) be a database multivalued mapping from a database scheme \( \mathcal{D} \) to a relation symbol \( r \).

(Only-if part). Suppose that there exists a non-deterministic unbound query \( \mathcal{N} = \langle \mathcal{L}, G, r(X) \rangle \) such that \( \mathcal{D}(\mathcal{D}) = \mathcal{D} \), \( r \) is an IDB predicate symbol of \( \mathcal{L} \), and for each \( D \) on \( \mathcal{D} \), \( \mathcal{D}(D) = \mathcal{N}(\mathcal{D}(D)) \). Let \( r \) be a relation on \( r \); we want to verify whether \( r \) is in \( \mathcal{D}(D) \). We guess an interpretation \( M \) of \( \mathcal{L} \) and check in polynomial time whether \( G \) is \( M \) and whether for each tuple \( t \) on \( r \), \( r(t) \) is in \( M \) if \( t \in r \) and \( \neg r(t) \) is in \( M \) otherwise. Moreover, depending on \( \mathcal{D} \) we perform the following additional test:

- if \( \mathcal{D} = \mathcal{P} \) or \( \mathcal{H} \), we verify whether \( M \) is a \( P \)-stable model—this test is in \( \mathcal{P} \) by Fact 5.1.
• if $\mathcal{X} = \mathcal{F}$, we verify whether $M$ is a $T$-stable model—this test is in $\mathcal{P}$ by Fact 5.3;
• if $\mathcal{X} = \mathcal{L}$, we verify whether $M$ is an $L$-stable model—this test is in $\mathcal{C}$ by Theorem 5.3.

It is easy to see that $r$ is in DMM($D$) if all of the above tests succeed. Observe that for $\mathcal{X} = \mathcal{M}$ we can just check whether $M$ is $P$-stable rather than $M$-stable (that is, instead of $\mathcal{C}$-complete) because of the condition that $r$ is fully defined in $M$; in fact, if such a $P$-stable model $M$ exists, no $r(t)$ is undefined in the $\mathcal{M}$ stable model containing $M$ as well. It turns out that $r$ is in DMM($D$) if $M$ is $\mathcal{P}$-rel-recognizable for $\mathcal{X} = \mathcal{P}$, $\mathcal{M}$, or $\mathcal{L}$ and $\Sigma_r^\mathcal{M}$-rel-recognizable for $\mathcal{X} = \mathcal{L}$.

(If part). Let $\mathcal{E}^{3}_{\mathcal{X}, \mathcal{Y}, \mathcal{Z}}[\mathcal{Q}] = \text{DB-C}$ be the expressive power of all bound queries under $\exists_{\mathcal{X}, \mathcal{Y}}$ semantics, where $\mathcal{X} = \mathcal{P}$, $\mathcal{I}$, $\mathcal{M}$, or $\mathcal{L}$. Assume that $C = \mathcal{M}$ for $\mathcal{X} = \mathcal{P}$, $\mathcal{M}$, and $\mathcal{L}$ because of Propositions 5.1 and 5.3, and Fact 5.3, respectively; moreover, by Theorem 5.2, $C = \Sigma_r^\mathcal{M}$ for $\mathcal{X} = \mathcal{L}$. Assume that $C$ is $\Sigma_r^\mathcal{M}$-recognizable. Consider the database collection $\mathcal{D}$ defined by DMM, i.e., $\mathcal{D} = \{D | \{r\} | D$ is any database on $\mathcal{D}$ and $r \in \text{DMM}(D)\}$. By Lemma 6.3, $\mathcal{D}$ is in DB-C. So there exists a bound query $\mathcal{Z} = \langle \mathcal{L}, \mathcal{S}, G \rangle$ such that $\mathcal{D}^{\mathcal{X}, \mathcal{Y}} = \mathcal{D}^{\mathcal{X}, \mathcal{Y}} | \{r\}$ and $\mathcal{E}^{3}_{\mathcal{X}, \mathcal{Y}, \mathcal{Z}}[\mathcal{Q}] = \mathcal{D}$. Let $\mathcal{D} = \mathcal{D}^{\mathcal{X}, \mathcal{Y}} | \{r_1, r_2\}$ where:

\[
\begin{align*}
  r_1 : & \quad r(X) \rightarrow \neg r(X), \\
  r_2 : & \quad r(X) \rightarrow \neg r(X).
\end{align*}
\]

We have that $\mathcal{D}^{\mathcal{X}, \mathcal{Y}} | \{r\} = \mathcal{D}$, $r$ is now an IDB predicate symbol, and $\mathcal{S}$ is a new IDB predicate symbol. Given a database $D$ on $\mathcal{D}$, the rules $r_1$ and $r_2$ enable the selection of any fully defined relation $r$ on $r$. Moreover, $G$ will be true if $D \cup \{r\}$ is in $\mathcal{D}$, i.e., if $r$ is in DMM($D$). Therefore, the unbound query $\mathcal{N} = \langle \mathcal{L}, \mathcal{P}, G, r(X) \rangle$ under $\forall_{\mathcal{X}, \mathcal{Y}}$ semantics defines DMM.

Note that the restriction on the full definiteness of the relation is only used in the proof of the only-if part for $M$-stable models. The problem of whether removing the restriction increases the expressive power for $M$-stable models is open, but our conjecture is that it does not.

To characterize the expressive power of non-deterministic unbound queries we extend the notation of $C_1/C_2$-recognizability to the case of relation recognizability: given two Turing machine complexity classes $C_1$ and $C_2$, a database multivalued mapping DMM: $\mathcal{D} \rightarrow r$ is $C_1/C_2$-rel-recognizable if there exists a database collection $\mathcal{D}$ in DB-C and a $C_2$-rel-recognizable database mapping DMM: $\mathcal{D} \rightarrow r$ such that (i) for each $D$ on $\mathcal{D}$ that is not in D, DMM($D$) = $\emptyset$ and (ii) for each $D \in \mathcal{D}$, DMM($D$) = DMM($D$).

Observe that, as the definite semantics of $P$-stable models is deterministic, it does not make sense to consider non-determinism for this case.

Theorem 6.5. Let DMM be a database multivalued mapping. Then

(a) DMM is expressible by a non-deterministic unbound query under the $\forall_{\mathcal{X}, \mathcal{Y}}$ semantics if and only if it is $\mathcal{P}, \mathcal{N}, \mathcal{P}$-rel-recognizable;
(b) DMM is expressible by a non-deterministic unbound query under the $\forall_{\mathcal{X}, \mathcal{Y}}$ semantics if and only if it is $\Pi_2^\mathcal{M}, \mathcal{N}, \mathcal{P}$-rel-recognizable;
(c) DMM is expressible by a non-deterministic unbound query under the $\forall_{\mathcal{X}, \mathcal{Y}}$ semantics if and only if it is $\Pi_2^\mathcal{M}, \mathcal{N}, \mathcal{P}$-rel-recognizable.

Proof. Let DMM be a database multivalued mapping from a database scheme $\mathcal{D}$ to a relation symbol $r$. Let $\mathcal{E}^{3}_{\mathcal{X}, \mathcal{Y}, \mathcal{Z}}[\mathcal{Q}] = \text{DB-C}$ be the expressive power of all bound queries under $\exists_{\mathcal{X}, \mathcal{Y}}$ semantics, where $\mathcal{X} = \mathcal{F}, \mathcal{I}, \mathcal{M}$, or $\mathcal{L}$. Observe that $C_1 = \Pi_2^\mathcal{M}$ for $\mathcal{X} = \mathcal{M}$ by Theorem 5.2, $C_1 = \mathcal{P}$ for $\mathcal{X} = \mathcal{F}$ by Fact 5.5, and $C_1 = \Pi_2^\mathcal{M}$ for $\mathcal{X} = \mathcal{P}$ by Theorem 5.4. Moreover, let $\mathcal{E}^{3}_{\mathcal{X}, \mathcal{Y}, \mathcal{Z}}[\mathcal{Q}] = \text{DB-C}$ be the expressive power of all bound queries under $\exists_{\mathcal{X}, \mathcal{Y}}$ semantics, where $\mathcal{X} = \mathcal{F}, \mathcal{I}, \mathcal{M}$, or $\mathcal{L}$. Observe that $C_2 = \mathcal{N}$ for $\mathcal{X} = \mathcal{F}$ and $\mathcal{I}$ because of Propositions 5.2 and 5.3, respectively; by Theorem 5.5, $C_2 = \Pi_2^\mathcal{M}$ for $\mathcal{X} = \mathcal{P}$.

(Only-if part). Let $\mathcal{N} = \langle \mathcal{L}, \mathcal{P}, G, r(X) \rangle$ be a non-deterministic unbound query such that $\mathcal{D}^{\mathcal{X}, \mathcal{Y}} | \{r\} = \mathcal{D}$ and $r$ is an IDB predicate symbol of $\mathcal{L}$. Suppose for each $D$ on $\mathcal{D}$, DMM($D$) = $\mathcal{N}^{\mathcal{X}, \mathcal{Y}} | \{D\}$; we show that DMM is $C_1/C_2$-rel-recognizable. Let $\mathcal{D}$ be the database collection for which $\mathcal{N}$ admits a non-empty answer set; then $D \in \mathcal{D}$ if both (i) $G$ is an $\exists_{\mathcal{X}, \mathcal{Y}}$ inference of $\mathcal{L}^{\mathcal{X}, \mathcal{Y}}$ and (ii) there exists an $\mathcal{X}$ stable model $M$ of $\mathcal{L}^{\mathcal{X}, \mathcal{Y}}$ such that $r_M$ is fully defined in $M$. The test (i) is in $C_1$. We carry out the test (ii) for the various types of stable models as follows:

• for $\mathcal{X} = \mathcal{F}$ the test is superfluous since any $T$-stable model is total and the existence of a $T$-stable model is guaranteed by the success of the test (i);
• for $\mathcal{X} = \mathcal{M}$ or $\mathcal{L}$ we guess an interpretation $M$ and verify in polynomial time whether $r_M$ is fully defined in $M$ and $M$ is a $P$-stable model; the test is then in $\Pi_2^\mathcal{M}$—so, as the test (i) is in $\Pi_2^\mathcal{M}$, the conjunction of the two tests is in $\Pi_2^\mathcal{M}$ as well.

Note that for $\mathcal{X} = \mathcal{M}$ we can avoid verifying whether $M$ is $M$-stable (that is a co-$\mathcal{N}$-complete test) because if $r_M$ is fully defined in the $P$-stable model $M$ then $r_M$ is fully defined in the $\mathcal{M}$ stable model $M'$ containing $M$ and $r_M = r_M$. Moreover, for $\mathcal{X} = \mathcal{L}$ the co-$\mathcal{N}$-complete
test of $L$-stability is not necessary either because if such a $P$-stable model exists then there must exist also an $L$-stable model $M'$ in which $r_M = r_M$ (by the condition of minimal undefinedness for $L$-stable models). In sum, $D$ is in $DB-C_2$.

Consider now the database multivalued mapping $DMM'$ defined by the query $\forall x. \exists y. P(x, y)$ under the possible semantics. By Theorem 6.4 $DMM'$ is $C_1$-rel-recogizable. Moreover, for each $D$ for which $NQ$ admits a non-empty answer set (i.e., for each $D \in D$), $DMM(D) = NQ_{NQ}(D)$; so, as $NQ_{NQ}(D) = NQ_{NQ}(D)$ and $DMM'(D) = \exists x. \forall y. P(x, y)$, $DMM'(D) = DMM(D)$. Hence, $DMM$ is $C_1/C_2$-rel-recogizable.

(If part). Suppose now that $DMM$ is $C_1/C_2$-rel-recogizable. Then there exists a database collection $D$ on $DD$ such that $D \in DB-C_1$ and for each $D$ not in $D$, $DMM(D) = \emptyset$. So there exists a bound query $Q = \langle D, G \rangle$ such that $DD_{Q,D} = DD$ and $DMM_{Q,D}(D) = D$. Moreover, there exists a $C_2$-rel-recogizable database multivalued mapping $DMM' : DD \rightarrow r$ such that $DMM(D) = DMM'(D)$ for each $D \in D$. Consider the database collection $D'$ defined by $DMM'$, i.e., $D' = \{ D \cup \{ r \} | D$ is any database on $DD$ and $r \in DMM'(D) \}$. By Lemma 6.3, $D'$ is in $DB-C_2$. So there exists a bound query $Q = \langle D', G' \rangle$ such that $DD_{Q,D} = DD \cup \{ r \}$ and $DMM_{Q,D}(D) = D'$. Consider the database scheme $DD' = DD \cup \{ r \}$ and the database collection $D'$ on $DD'$ defined as: $\{ D \cup \{ r \} | D$ is any database on $DD$ and $r \in DMM'(D) \}$. We construct the program $L' = L \cup DD \cup \{ r_1, r_2, r_3, r_4, r_5 \}$ where

\[
\begin{align*}
r_1: & \quad r(X) \leftarrow \neg G(X). \\
r_2: & \quad G(X) \leftarrow \neg r(X). \\
r_3: & \quad G \leftarrow G, G'. \\
r_4: & \quad G \leftarrow \neg G, \neg r(X). \\
r_5: & \quad \hat{r}(X) \leftarrow \neg r(X). \\
r_6: & \quad \hat{r}(X) \leftarrow \neg G', \neg r(X).
\end{align*}
\]

We have that $DD_{Q,D} = DD$ and $r$ is an IDB predicate symbol. Observe that, in the construction of an $M$-stable model, the rules $r_3$ and $r_5$ select a fully defined relation on $r$. Given a database $D$ in $D$ (i.e., $G$ is true), the rules $r_4$ and $r_3$ accept only the relations on $r$ that are in $DMM'(D)$ (i.e., $G$ is true and, then, $G'$ is true as well), and, the other hand, if $D \notin D$, these rules discard every relation on $r$ as $G$ false and, then, $G'$ is false as well. Therefore, given the non-deterministic unbound query $\forall x. \exists y. P(x, y)$, for each $D$ on $DD$, $\exists x. \forall y. P(x, y) = DMM(D)$ and $DMM'(D) = DMM(D)$ if $D \in D$ or otherwise $\exists x. \forall y. P(x, y) = DMM(D) = \emptyset$; hence, $DMM$ is defined by $\forall x. \exists y. P(x, y)$.

Observe that the proof of the only-if part would have been carried out also without the restriction on the full definiteness of $r$; actually, without this restriction, the proof would have been much simpler.

We conclude this section by giving the characterization of $T$-stable models under certain semantics.

**Proposition 6.1.** Let $DMM$ be a database multivalued mapping. Then $DMM$ is expressible by a non-deterministic unbound query under the $\forall x. \exists y. P(x, y)$ semantics if and only if it is $coNP$. $\forall x. \exists y. P(x, y)$-rel-recogizable.

**Proof.** It follows the lines of the proof of Theorem 6.5—in this case we have that $C_1 = coNP$ by Fact 5.6 and $C_2 = \forall x. \exists y. P(x, y)$ by Proposition 5.2.

7. CONCLUSION

In this paper we have analyzed the expressive power of various types of stable models in DATALOG queries. A $P$-stable model [36, 38, 39] is characterized by the two following properties: (i) every positive literal in it is inferred from the rules, possibly using the negative literals as additional axioms, (ii) the set of negative literals is the greatest unfounded set. The traditional definition of a stable model [17] also requires that the model be total ($T$-stable model).

The $P$-stable models of logic program form a lower semi-lattice w.r.t. containment relationships; the bottom element is the well-founded model and the top elements are the $M$-stable (maximal stable) models of [40], which correspond to the (partial) stable models of [38], the preferred extensions of [13], the regular models of [48], and the maximal stable classes of [6]. $L$-stable (least undefined stable) models [40] are the $M$-stable models which leave undefined a minimal number of elements of the Herbrand base. Figure 36 reports the complexity of recognizing the various types of stable models and pinpoints that a stable model for every type exists for all programs except for $T$-stable models whose existence test is $NP$-complete.

As the stable models of any of the above types can be taken as the “intended” models of a logic program, we have considered two versions of deterministic semantics for each type: the possible semantics, which is based on the union of all stable models of the chosen type, and the definite semantics, which is based on their intersection. For $T$-stable models we have also considered the certain semantics that differs from the definite semantics only for the programs with no $T$-stable models; in this case certain semantics infers that everything is true whereas definite semantics infers that nothing is true. As shown in [37], definite semantics has a higher expressive power than does certain semantics.

The results on the expressive power for bound queries are summarized in Fig. 3b, and are given in terms of database complexity classes, thus BD-C is the family of all database sets whose recognition is in C, where C is a Turing machine.
### Complexity Results on Stable Models

**Complexity Class**

- The table shows that $L$-stable models have a higher expressive power than other stable models and their expressive power under possible semantics is complementary to the expressive power under definite semantics. Considering their expressive power and the fact that $L$-stability differs from $T$-stability only when a program has no $T$-stable models, it seems that $L$-stable models are the more appropriate extension of stable models to the domain of partial interpretations.

- For $T$-stable and $M$-stable models the definite semantics has an expressive power higher than the other possible semantics while for $P$-stable models the expressive power under definite semantics is lower than the one under possible semantics for it coincides with the expressive power of the well-founded model.

- We have also characterized the expressive power of DATALOG\(c\) unbound queries and we have shown that it is in strong correspondence with the one of DATALOG\(c\) bound queries. The results are shown in Fig. 3c and are represented in terms of $C$-recognizability (thus, recognizing a tuple in the answer is in $C$) as well as of $C$-rel-recognizability (thus, recognizing the whole set of tuples in the answer is in $C$). For the definite semantics of $T$-stable model we also use $C_1$/$C_2$-(rel)-recognizability (thus there exists a $C_1$-recognizable database set $D$ such that the query has a non-empty answer only for databases in $D$ and the restriction of the query to $D$ is $C_2$-(rel)-recognizable).

**Table:**

<table>
<thead>
<tr>
<th>Stable Model</th>
<th>Possible</th>
<th>Definite</th>
<th>Certain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$-stable</td>
<td>$NP$-rec.</td>
<td>$NP$-rec.</td>
<td>$NP$-rec.</td>
</tr>
<tr>
<td>$T$-stable</td>
<td>$NP$-rec.</td>
<td>$NP$-rec.</td>
<td>$NP$-rel-rec.</td>
</tr>
<tr>
<td>$M$-stable</td>
<td>$NP$-rel-rec.</td>
<td>$NP$-rel-rec.</td>
<td>$NP$-rel-rec.</td>
</tr>
<tr>
<td>$L$-stable</td>
<td>$NP$-rec.</td>
<td>$NP$-rel-rec.</td>
<td>$NP$-rel-rec.</td>
</tr>
</tbody>
</table>

- The definite semantics is more expressive than certain semantics also for the case of unbound queries.

We have finally substantiated our skepticism about the practical applicability of any deterministic semantics for unbound queries and we have then proposed to combine the expressive strength of determinism in defining decision problems with the “cut” capability of non-determinism for selecting one of the solutions of a finding problem. We have characterized the expressive power of stable models also for these types of queries, called non-deterministic unbound queries, that return a set of relations (answer set) and which are shown in Fig. 3d and are given in terms of $C$-rel-recognizability (thus recognizing a relation in the answer set is in $C$) as well as of $C_1$/$C_2$-(rel)-recognizability (thus, recognizing the whole set of relations in the answer set is in $C$). For the definite semantics of $T$-stable model we also use $C_1$/$C_2$-(rel)-recognizability.
to characterizing the expressive power of subclasses of queries (particularly, those for which definite and possible semantics coincide) and to finding a theoretical framework as well as optimization strategies for an effective combination of possible and definite semantics with non-determinism.

REFERENCES


19. S. Greco, D. Saccá, and C. Zaniolo, DATALOG queries with stratified negation and choice: From $\Pi^0_1$ to $\Sigma^0_1$, in “Proc. of the Fifth Int. Conf. on Database Theory (ICDT'95), LNCS 893, Springer-Verlag, 1995,” pp. 82–96.


27. N. Leone, M. Romeo, P. Rullo, and D. Saccá, Effective implementation of negation in database logic query languages, in “LOGC’91 + : Deductive Databases with Complex Objects” (P. Atzeni, Ed.), pp. 159–175, LNCS 701, Springer-Verlag, 1993.


