NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

The expressive power of second-order datalog under the well-founded semantics of negation

Romanos E. Aslanis Petrou

Supervisors: **Panos Rondogiannis,** Professor
Angelos Charalambidis, Assistant Professor

ATHENS

OCTOBER 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Η εκφραστική ισχύς της Datalog δεύτερης τάξης υπό την well-founded σημασιολογία της άρνησης

Ρωμανός Ε. Ασλάνης Πέτρου

Επιβλέποντες: **Πάνος Ροντογιάννης,** Καθηγητής
Άγγελος Χαραλαμπίδης, Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2022

BSc THESIS

The expressive power of second-order datalog under the well-founded semantics of negation

Romanos E. Aslanis Petrou

S.N.: 1115201600012

SUPERVISORS: **Panos Rondogiannis,** Professor
Angelos Charalambidis, Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Η εκφραστική ισχύς της Datalog δεύτερης τάξης υπό την well-founded σημασιολογία της άρνησης

Ρωμανός Ε. Ασλάνης Πέτρου
Α.Μ.: 1115201600012

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Πάνος Ροντογιάννης,** Καθηγητής
Άγγελος Χαραλαμπίδης, Επίκουρος Καθηγητής

ABSTRACT

It was recently shown that higher-order positive datalog captures the complexity class $(k-1)$-$EXPTIME$ on ordered databases [2], and in particular for $k=2$, second order datalog captures $EXPTIME$. In this thesis we investigate the expressive power of second-order datalog under the well-founded semantics [4]. More specifically, we present a recursive algorithm that takes as input an existential second-order logic formula ϕ and constructs a second-order WFS[1] datalog program P_ϕ. Furthermore there is an equivalence, in terms of satisfiability, between the finite models of ϕ and the well-founded model of P_ϕ, that implies a correct behaviour of the constructed program. In other words, the satisfaction of a given $SO(\exists)$ formula by a structure is reduced into the satisfaction of the well founded model of P_ϕ. As a result, by the well-known theorem of Fagin [10] and the above transformation we deduce that second-order WFS datalog captures $\mathcal{NP}$. Finally this result is not strict, meaning that it is possible that second-order WFS datalog is stronger, and does not depend upon any ordering assumption since $SO(\exists)$ is powerful enough to nondeterministically guess such an ordering.

SUBJECT AREA: Higher-order logic programming and complexity theory

KEYWORDS: Higher-order logic programming, second-order datalog, descriptive complexity theory

[1]Sometimes instead of writing well-founded semantics, we will use the abbreviation WFS

ΠΕΡΙΛΗΨΗ

Πρόσφατα δείχτηκε ότι η datalog k-τάξης χωρίς άρνηση εκφράζει την κλάση πολυπλοκότητας $(k-1)$-$EXPTIME$ σε διατεταγμένες βάσεις δεδομένων [2] και ειδικότερα η datalog δεύτερης τάξης εκφράζει την κλάση $EXPTIME$. Σε αυτή την πτυχιακή ερευνούμε την εκφραστική ισχύ της datalog 2ης-τάξης κάτω απο την well-founded σημασιολογία της άρνησης [4]. Πιο συγκεκριμένα, παρουσιάζουμε έναν αναδρομικό αλγόριθμο που δέχεται ως είσοδο έναν λογικό τύπο ϕ της υπαρξιακής λογικής δεύτερης τάξης $SO(\exists)$ και κατασκευάζει ένα πρόγραμμα datalog 2ης-τάξης P_ϕ. Επιπλέον υπάρχει μια ισοδυναμία, όσον αφορά την ικανοποιησιμότητα, μεταξύ των πεπερασμένων μοντέλων του ϕ και του well-founded μοντέλου του P_ϕ, που υπονοεί μια σωστή συμπεριφορά του προγράμματος P_ϕ. Με άλλα λόγια η ικανοποιησιμότητα ενός δοσμένου λογικού τύπου της $SO(\exists)$ από μια συγκεκριμένη δομή (ερμηνεία), μεταφέρεται στην ικανοποιησιμότητα ενός "τελικού" κανόνα του προγράμματος P_ϕ απο το well-founded μοντέλο. Ως αποτέλεσμα, με βάση το γνωστό θεώρημα του Fagin [10] και τον παραπάνω μετασχηματισμό συμπαιρένουμε ότι η datalog 2ης-τάξης με well-founded σημασιολογία εκφράζει την κλάση $\mathcal{NP}$. Τέλος το αποτέλεσμα αυτό δεν είναι αυστηρό, δηλαδή η datalog 2ης-τάξης μπορεί να είναι ακόμα πιο δυνατή, και δεν βασίζεται σε κάποια υπόθεση διάταξης της εισόδου, δεδομένου ότι η $SO(\exists)$ είναι αρκετά δυνατή για να επιλέξει μη ντετερμινιστικά μια τέτοια διάταξη.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Λογικός προγραμματισμός υψηλής τάξης και θεωρία πολυπλοκότητας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Λογικός προγραμματισμός υψηλής τάξης, datalog δεύτερης τάξης, περιγραφική θεωρία πολυπλοκότητας

CONTENTS

1. INTRODUCTION

Logic and functional programming are the two most commonly used, in both theoretical studies and real world applications, declarative paradigms. However, while functional programming is well known for the higher order functions that lead to a more clean and expressive way of programming, logic programming is in general first order. This is partly justified since there had been many attempts to extend logic programming into a higher order one, but came across one difficulty after the other. Of course they found their way into some applications, such as theorem proving and meta-programming [9], [19], but could not form a general-purpose higher order programming language. However extensional higher-order logic programming was introduced in [1] and opened new doors in the semantics of higher-order logic programming. The goal of a more general purpose and compact programming language that retains all the well-known properties of classical first order logic programming was achieved, namely, a higher-order logic programming language with extensional semantics in which every program has a unique minimum Herbrand model,which is the greatest lower bound of all Herbrand models and the least fixed-point of an immediate consequence operator, as well as a generalized SLD-resolution proof procedure that is sound and complete with respect to the above model theoretic semantics. The next step was to invent the semantics of negation of the above language, [3], [16], which ultimately led to the well-founded semantics of higher-order logic programming [4]. So whenever we mention second-order datalog under well-founded semantics, we are referring to the second-order function-free subset of the language introduced in [4], of which we study the expressive power. Lastly, we know from the case of first-order datalog, and it is intuitively obvious, that adding negation might change the expressive power of the language. Moreover for different semantics of negation we get different results. For example if we use stratified negation (or even well-founded negation) for first-order datalog, datalog still captures $PTIME$ (under an ordering assumption), while if we use stable model semantics datalog becomes stronger and captures $co\text{-}NP$ [5], [17].

Usually to prove that a query language $\mathcal{Q}$ captures a complexity class $\mathcal{C}$ one shows that every Turing machine that decides a language in $\mathcal{C}$ can be simulated by a program in $\mathcal{Q}$. If in addition the opposite direction is true as well, namely that every query in $\mathcal{Q}$ is computable in $\mathcal{C}$, then we can say that $\mathcal{Q}$ expresses exactly all those languages in $\mathcal{C}$ and no more. The approach that was used in the case of higher-order positive datalog [2] was exactly this. That is to say *M is a deterministic Turing machine that decides a language L in $(k-1)$-EXPTIME iff there exists a k-order positive datalog program P that decides L*. On this thesis we are going to use a different approach. One that relates datalog programs with logic formulas in a direct way and exploits the complexity theoretic results that descriptive complexity has given us.

More specifically we are going to use the well-known Fagin's theorem [10] which intuitively states that *NP is equal to the set of all existential second-order queries*, [13], [14]. In other words every language in NP can be expressed in existential second-order logic $SO(\exists)$[1] and vice versa. As for the content of this thesis, let $\phi \in SO(\exists)$ and $\mathcal{U}$ be a structure of the same vocabulary as ϕ. We construct a second-order WFS datalog program P_ϕ, using a recursive transformation that takes as input the logic formula ϕ and produces the datalog program P_ϕ. We can see $\mathcal{U}'s$ universe and the meaning that $\mathcal{U}$ gives in $\phi's$ predicate

[1] $\phi \in SO(\exists) \Leftrightarrow \phi \equiv \exists R_1, \ldots, \exists R_n \psi$ where $R_1, \ldots R_n$ are first order predicate variables and ψ is a first-order logic formula

symbols as input and translate it as the input database of P_ϕ. In other words, we set the universe $|\mathcal{U}|$ as the Herbrand universe U_{P_ϕ} and the input of the constructed program is a set of facts that encode the meaning that $\mathcal{U}$ has given in predicate symbols. This allows us to show an equivalence between the finite models of ϕ and the well founded model of P_ϕ. More specifically if the structure $\mathcal{U}$ satisfies ϕ, which in turn means that there exist the relations $R_1, \ldots R_n$ in the first part of ϕ, then those relations also exist in U_{P_ϕ} and the well founded model assigns the value true in a specific predicate of P_ϕ. If on the other hand $\mathcal{U}$ does not satisfy ϕ then the well founded model assigns the value false. From Fagin's theorem and the above transformation, we get that second-order datalog under the well-founded semantics can express every language in NP.

Another thing to mention is that of the importance of ordered input. Both in descriptive complexity and in the complexity theoretic database theory, there is a significant obstacle. In order to show an equivalence between logic and complexity, we have to somehow relate logic queries and Turing machines. On the one hand logic queries, datalog programs, work on databases and on the other hand Turing machines work with strings of characters. By definition, a string comes with a total order (first character, second character, $\ldots$, last character) while databases are unordered sets. At first this might seem to be little to no important, but in reality this is not true. Think of how a Turing machine works and how you can simulate one without an order on the input. What does next or previous tape cell mean if next and previous are not defined anywhere? To conclude, query languages, especially first-order ones, without an order in the input are incapable of expressing even the simpliest problems. Let us just mention that classical datalog (first-order datalog) can express $PTIME$ under an ordering assumption [15], [18], while without that assumption it fails to express even some regular languages.

The structure of this thesis is organized as follows: There are two main parts, Part 2 (the preliminaries) and Part 3 (the main content of this thesis). Both of them are divided into two sections. Section 2.1 presents the syntax of higher-order datalog and makes a reference on the well-founded semantics, Section 2.2 gives some preliminary knowledge for second-order logic and descriptive complexity, Section 3.1 presents the transformation of $SO(\exists)$-formulas into second-order WFS datalog programs and Section 3.2 refers to how we can bypass some obstacles, regarding function symbols and the order of input. Finally the last section, Section 4, discusses the conclusions we get from this thesis and proposes some ideas for extending the semantics of higher-order WFS datalog and possibly making it a more powerful language.

2. PRELIMINARIES

2.1 Higher-order datalog, syntax and the well-founded semantics

To begin with, as higher-order datalog under the well-founded semantics we consider the higher-order function free subset of the language $\mathcal{HOL}$ introduced in [4]. Just like in the case of higher-order positive datalog [2] which inherited a syntactic restriction from the language $\mathcal{H}$ [1], second-order WFS datalog inherited the same restriction from $\mathcal{HOL}$. Namely that *in the head of every rule in a program, each argument of predicate type must be a variable, and all such variables must be distinct.* To get the idea of what that means, let us see some examples[1].

Example 2.1.1 *The program below does not satisfy the restriction.*

$$\begin{aligned}&p(X) \leftarrow q(X).\\&q(a).\\&r(q).\end{aligned}$$

The predicate constant q appears in the head of the rule $r(q)$.

Example 2.1.2 *Similarly the program:*

$$\begin{aligned}&p(Q,Q) \leftarrow Q(X).\\&q(a).\\&q(b).\end{aligned}$$

Does not satisfy the restriction because the predicate variable Q appears twice in the head of a rule.

However the next example is a valid second-order WFS datalog program.

Example 2.1.3

$$\begin{aligned}&subset(P,Q) \leftarrow\sim\ nonsubset(P,Q).\\&nonsubset(P,Q) \leftarrow P(X), \sim\ Q(X).\end{aligned}$$

Which, intuitively, states that P is a subset of Q if it is not the case that there exists an X for which P is true while Q is false.

We will denote individual constants with the lowercase letters $a, b, c, \dots$, individual variables with the uppercase letters $X, Y, Z, \dots$, predicate constants with $p, q, s, t, \dots$ and predicate variables with $P, Q, R, \dots$.

[1]For simplicity we will use standard prolog-like syntax

Let us now proceed to some definitions of higher-order WFS datalog. First of all just like in the case of $\mathcal{HOL}$, higher-order WFS datalog is based on a simple type system. There are two base types: o, the boolean domain and ι the domain of individuals. A composite type can be either predicate π or argument ρ (we do not have functional types).

$$\pi := o \mid \rho \to \pi$$
$$\rho := \iota \mid \pi$$

Second-order datalog: *Of course in the case of higher-order datalog, every predicate type can be at most a second-order one*.

Definition 2.1.1 *The alphabet of higher-order WFS datalog consists of:*

1. *Predicate constants of every predicate type π.*
2. *Predicate variables of every predicate type π.*
3. *Individual constants of type ι.*
4. *Individual variables of type ι.*
5. *Logical constant symbols:*
   - *False and True*
   - *Equality constant $\approx$ of type $\iota \to \iota \to o$*
   - *Generalized disjunction and conjuction constants $\bigvee_\pi$, $\bigwedge_\pi$ of type $\pi \to \pi \to \pi$.*
   - *Generalized inverse implication constant $\leftarrow_\pi$ of type $\pi \to \pi \to o$.*
   - *Existential quantifier $\exists_\rho$ of type $(\rho \to o) \to o$.*
   - *Negation constant $\sim$ of type $o \to o$.*
6. *The abstractor λ and the parentheses "(" and ")".*

Definition 2.1.2 *The set of expressions of the second-order WFS datalog is defined as follows:*

1. *Every predicate variable (respectively, predicate constant) of type π is an expression of type π; every individual variable (respectively, individual constant) of type ι is an expression of type ι; the propositional constants False and True are expressions of type o.*
2. *If E_1 is an expression of type $\rho \to \pi$ and E_2 is an expression of type ρ, then $(E_1 E_2)$ is an expression of type π.*
3. *If R is an argument variable of type ρ and E is an expression of type π, then $(\lambda R.E)$ is an expression of type $\rho \to \pi$.*
4. *If E_1, E_2 are expressions of type π, then $(E_1 \bigvee_\pi E_2)$ and $(E_1 \bigwedge_\pi E_2)$ are expressions of type π.*
5. *If E is an expression of type o, then $(\sim E)$ is an expression of type o.*
6. *If E_1, E_2 are expressions of type ι, then $(E_1 \approx E_2)$ is an expression of type o.*

7. *If E is an expression of type o and R is a variable of type ρ then $(\exists\rho R E)$ is an expression of type o.*

An expression of type ι will be called a term; if it does not contain any individual variables, it will be called a ground term.

Definition 2.1.3 *A clause of second-order WFS datalog is of the form $p \leftarrow E$ where p is a predicate constant of some predicate type π and E is a closed expression also of some type π.*

Definition 2.1.4 *A higher-order WFS datalog program is a finite set of clauses.*

Definition 2.1.5 *Let P be a program. The Herbrand universe U_P of P is the set of all ground terms that can be formed out of the individual constants of P.*

Definition 2.1.6 *A Herbrand interpretation I of a program P is a function, with domain U_P, that assigns:*

1. *To every individual constant c of P itself.*
2. *To every predicate constant $p : \pi$ of P a function from the denotation of type π, with underlying domain the Herbrand universe U_P.*

Moreover a Herbrand state of P is a state whose underlying domain is U_P. A Herbrand model of P is a Herbrand interpretation that is a model of P.

As for the well-founded semantics we will just mention the main idea and some basic definitions. For deeper understanding the reader is highly advised to have a look at [4], where the well-founded semantics for higher-order logic programs are introduced.

The key concept of the well-founded semantics for higher-order logic programs is to interpretate the predicate types as three-valued Fitting-monotonic[2] functions [11]. That is because when you add negation, the predicates are not anymore[3] (standard) monotonic and continuous functions. In other words if a predicate is true for an input relation, it is not necessarily true for a superset of that relation, at least according to the standard truth ordering.

Example 2.1.4 *The program below states that p is True for every relation Q that doesn't include $\{a\}$ or doesn't include $\{b\}$.*

$$p(Q) \leftarrow \sim Q(a).$$
$$p(Q) \leftarrow \sim Q(b).$$

The predicate p is True for the relations $\{a\}, \{b\}$, but not for the relation $\{a, b\}$.

[2]Fitting monotonicity is a type of monotonicity with respect to the information ordering ($undef \preceq false$, $undef \preceq true$) rather than the standard truth ordering ($false \leq undef \leq true$).

[3]Remember that in the semantics of [1], predicates where interpretated as monotonic functions.

With that being said, an interpretation of a program is a function that assigns Fitting-monotonic functions to the predicates of the program. Furthermore a Fitting-monotonic immediate consequence operator is easy to be defined, whose least fixpoint is minimal with respect to the information ordering. However this is not the well-founded model of a program, since the well-founded model should be minimal with respect to the standard truth ordering. The solution to that was to find a bijection between three-valued interpretations and pairs of two-valued-result ones.[4] More specifically by starting from a pair of an underdefined and an overdefined interpretation and by iterating an appropriate operator, we get the limit of that sequence. That limit pair of interpretations can be converted into a three-valued interpretation M_P which is the well-founded model of P, a minimal model with respect to the standard truth ordering.

Let us now proceed to somehow more ”practical” examples of second-order WFS datalog.

Example 2.1.5 (Transitive closure) *A second-order program to compute the closure of an input relation R.*

$$\begin{aligned}&transitiveClosure \leftarrow \lambda R.\lambda X.\lambda Y(R\,X\,Y).\\&transitiveClosure \leftarrow \lambda R.\lambda X.\lambda Y\,\exists Z((R\,X\,Z) \wedge (transitiveClosure\;R\,Z\,Y)).\end{aligned}$$

Or in standard prolog-like syntax:

$$\begin{aligned}&transitiveClosure(R, X, Y) \leftarrow R(X, Y).\\&transitiveClosure(R, X, Y) \leftarrow R(X, Z), transitiveClosure(R, Z, Y).\end{aligned}$$

Example 2.1.6 (clique) *Let the input be a set of facts encoding a vertex set ($v(a), v(b), \dots$) and an edge set ($e(a, b), e(a, c), \dots$) of a graph G.*

$$clique \leftarrow \lambda R.\Big((subset\,v\,R) \wedge \sim\ \big(\exists X.\exists Y.\,((R\,X) \wedge (R\,Y) \wedge \sim\ (e\;X\,Y))\big)\Big).$$

Or in standard prolog-like syntax (infact in the rest of this thesis, to avoid any confusion, we are going to use standard prolog syntax):

$$\begin{aligned}&clique(R) \leftarrow subset(v, R), \sim\ nonClique(R).\\&nonClique(R) \leftarrow R(X), R(Y), \sim\ e(X, Y).\end{aligned}$$

R is a clique of the input graph if it is a subset of it's vertex set and if it is not the case that it is not a clique, which is true if there exist two vertices in R that are not connected with each other. Intuetively the well-founded model M_P will set to true $clique(r)$, forall $r \subseteq v = U_P$ that encode a clique of the input graph.

[4]The tool that was used here was the approximation fixpoint theory [6], [7]

Example 2.1.7 (3-coloring) *Again let the input database consist of some facts encoding the vertex set ($v(a), v(b), \dots$) and the edge set ($e(a,b), e(a,c), \dots$) of a graph. One way to compute the three chromatic classes ($red, green, blue$)of a graph is:*

$$\begin{aligned}
&emptyIntersection(P,Q) \leftarrow not\,nonEmpty(P,Q).\\
&nonEmpty(P,Q) \leftarrow P(X), Q(X).\\
&allVertices(R,G,B) \leftarrow \sim\ nonAll(R,G,B).\\
&nonAll(R,G,B) \leftarrow v(X), \sim\ R(X), \sim\ G(X), \sim\ B(X).\\
&nonThreeColor(R,G,B) \leftarrow e(X,Y), R(X), R(Y).\\
&nonThreeColor(R,G,B) \leftarrow e(X,Y), G(X), G(Y).\\
&nonThreeColor(R,G,B) \leftarrow e(X,Y), B(X), B(Y).
\end{aligned}$$

$$\begin{aligned}
threeColor(R,G,B) \leftarrow & allVertices(R,G,B),\\
& emptyIntersection(R,G),\\
& emptyIntersection(R,B),\\
& emptyIntersection(G,B),\\
& \sim\ nonThreeColor(R,G,B).
\end{aligned}$$

The predicate $allVertices$ states that every vertex must be colored, $emptyIntersection$ states that every vertex must be colored with only one color and predicate $nonThreeColor$ states that there should not be adjacent vertices with the same color.

2.2 Second-order logic and Descriptive complexity theory

We will just mention some basic definitions for both second-order logic and descriptive complexity for the shake of completeness and to avoid any possible misunderstanding.

Second-order logic (SO) is an extension of first-order (FO) that in addition has predicate and function variables. The set of terms is the set of expressions that can be build by applying function symbols (both constants and variables) to constant symbols and the set of well-formed formulas is extended with two new type constructive operations; namely if ϕ is well-formed formula then $\forall X^n \phi$ and $\forall F^n \phi$ are also well-formed, where X^n is a predicate variable of arity n and F^n is a function variable of arity n.
The role of a structure (interpretation) is the same, namely a function on the set of parameters which satisfies the known conditions of a structure and the definition of satisfaction is extended in an obvious way. That means, let $\mathcal{U}$ be a structure with domain $|\mathcal{U}|$ and s be a state function, then $s(u) \in |\mathcal{U}|$, $s(X^n) \in |\mathcal{U}|^n$ and $s(F^n) \in |\mathcal{U}|^n \to |\mathcal{U}|$. Then satisfaction for the formulas that use the new quantifiers is defined as:

$$\models_{\mathcal{U}} \forall X^n \phi[s] \textit{ iff for every relation R of arity } n \textit{ on } |\mathcal{U}|, \textit{ we have } \models_{\mathcal{U}} \phi[s(X^n|R)]$$

and

$$\models_{\mathcal{U}} \forall F^n \phi[s] \textit{ iff for every function } f : |\mathcal{U}|^n \to |\mathcal{U}|, \textit{ we have } \models_{\mathcal{U}} \phi[s(F^n|f)]$$

Example 2.2.1 (Peano's induction postulate) *Any set of natural numbers that contains 0 and is closed under successor function is the set of all natural numbers. In second-order logic this can be written as:*

$$\forall Q\Big(Q(0) \wedge \forall y\big(Q(y) \to Q(s(y))\big) \to \forall y Q(y)\Big)$$

It is quite obvious that second-order logic is a "superset" of first-order logic and thus one could reasonable assume that second-order logic is more powerful than first-order. This is completely true and in fact there is a whole area that studies those relations between formal logics, descriptive complexity [13], [14].

The key concept of descriptive complexity theory is a different point of view into complexity theory, one that understands the computational complexity of a problem as the richness of the language that is needed to specify it. Instead of characterizing the complexity of a problem by measuring the time and space a Turing machine needs to perform a computation, in descriptive complexity we measure the expressive power of a formal logic that is needed for the description of the problem. In fact there are many known results relating fragments and extensions of formal logics with important computational classes. For example, $FO(lfp)$[5] captures $PTIME$, $Horn\text{-}SO(\exists)$[6] captures $PTIME$, SO in whole captures PH, etc...But how exactly does a logic formula express a language and a formal logic a complexity class?

Example 2.2.2 (An intuitive answer) *The formula below expresses the language of 3-colorable graphs, a well known NP-Complete language.*

$$\phi \equiv \exists R \exists Y \exists B \forall x \forall y \Big(\big(R(x) \vee Y(x) \vee B(x)\big) \wedge \big(E(x,y) \rightarrow \\ \neg(R(x) \wedge R(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y))\big)\Big)$$

Let $\mathcal{U}$ be a model of ϕ. The universe $|\mathcal{U}|$ can be seen as a vertex set of a graph G, $E^{\mathcal{U}}$ as the edge set of G and since $\mathcal{U}$ is a model, there exist c_1, c_2, c_3 such that they satisfy $\phi[R|c_1, Y|c_2, B|c_3]$. In particular c_1, c_2, c_3 will be the three chromatic classes of G.

Definition 2.2.1 *A boolean query is a map $I : STRUCT[\tau] \rightarrow \{0,1\}$ that is polynomially bounded. That is, there is a polynomial p such that for all $A \in STRUCT[\tau]$, $\|I(A)\| \leq p(\|A\|)$. Where $STRUC[\tau]$ is the set of all finite structures of vocabulary τ.*

To catch the spirit of a decision problem the queries are boolean. However since Turing machines do not only decide problems, but also compute other functions on natural numbers there are also "non"-boolean queries $STRUCT[\tau] \rightarrow STRUCT[\sigma]$, where σ is, possible, another vocabulary.

Definition 2.2.2 *Let $\mathcal{LO}$ be a formal logic, $\mathcal{C}$ a computational class, ϕ a logic formula and τ it's vocabulary. We will say that every boolean query in $\mathcal{LO}$ is computable in $\mathcal{C}$, denoted as $\mathcal{LO} \subseteq \mathcal{C}$, if:*

$$\forall \mathcal{I} \in STRUCT[\tau],\ \mathcal{I} \models \phi \Rightarrow \exists M_{\mathcal{C}}, M_{\mathcal{C}}(bin(\mathcal{I})) = 1$$

Where $M_{\mathcal{C}}$ is a $\mathcal{C}$-Turing machine. Furthermore, if also the other direction is true, namely that every language $\mathcal{L} \in \mathcal{C}$ is expressible in $\mathcal{LO}$ ($\mathcal{C} \subseteq \mathcal{LO}$), then we say that $\mathcal{LO} = \mathcal{C}$.

Definition 2.2.3 *An expression of existential second-order logic $SO(\exists)$ over a vocabulary τ is of the form $\exists \bar{R} \phi$, where $\bar{R}$ is a m-tuple of first order relations ($< R_1, R_2, \ldots, R_m >$) that do not appear in the predicates of τ and ϕ is a first-order formula.*

[5] $FO(lfp)$: first-order logic extended with a least fixpoint operator

[6] $Horn\text{-}SO(\exists)$: second-order existential horn logic

One could say that $SO(\exists)$ is first-order logic with the power of existentially quantification of relations over the universe of a structure. The following sentences are valid $SO(\exists)$ formulas.

$$\exists P\,\exists x(P(x))$$
$$\exists R\,\exists Q\,\forall x\big(R(x) \land \neg Q(x)\big)$$
$$\exists R\,\exists Q\,\exists S\big(\forall x(R(x) \to Q(x)) \lor \neg S(x)\big)$$

Proposition 2.2.1 *The second-order existentially definable boolean queries are all computable in NP. In symbols, $SO(\exists) \subseteq NP$.*

Theorem 2.2.1 (Fagin's 1973) *NP is equal to the set of existential, second-order boolean queries, $NP = S0(\exists)$. Furthermore, this equality remains true when the first-order part of the second-order formulas is restricted to be universal.*

Fagin's theorem was the cornerstone of descriptive complexity. After his publication [8], a whole bunch of results started to come out relating formal logic and computational complexity.

3. THE EXPRESSIVE POWER OF SECOND-ORDER WFS DATALOG

3.1 The transformation

Let $\psi \in SO(\exists) \Leftrightarrow \psi \equiv \exists \bar{R} \forall \bar{x} \phi[\bar{R}, \bar{x}]$, where $\bar{R}$ is a tuple of predicate variables, $\bar{x}$ is a tuple of individual variables and ϕ is a first-order formula without any other quantifier.
Since $\forall x \phi \equiv \neg \exists x \neg \phi$, we swap $\forall \bar{x}$ with $\neg \exists \bar{x} \neg$ and we get formulas of the form $\psi \equiv \exists \bar{R} \neg \exists \bar{x} \neg \phi[\bar{R}, \bar{x}]$. Our transformation will take as input such a formula and recursively construct a second-order datalog program according to the following rules.

Base:

- If $\psi \equiv c_1 = c_2$, where c_1, c_2 are individual constants, our program is the single rule $p_\psi \leftarrow c_1 \approx c_2$.
- If $\psi \equiv x_1 = x_2$, where x_1, x_2 are individual variables, our program is the single rule $p_\psi(X_1, X_2) \leftarrow X_1 \approx X_2$.
- If $\psi \equiv R(\bar{x}, \bar{c})$, where R predicate variable, $\bar{x}$ tuple of variables and $\bar{c}$ tuple of constants, our program is the rule $p_\psi(R, \bar{X}) \leftarrow R(\bar{X}, \bar{c})$.
- If $\psi \equiv r(\bar{x}, \bar{c})$, where r predicate constant, $\bar{x}$ tuple of variables and $\bar{c}$ tuple of constants, our program is the rule $p_\psi(\bar{X}) \leftarrow r(\bar{X}, \bar{c})$.

For ψ_1, ψ_2 already constructed formulas:

- If $\psi \equiv \neg \psi_1[\bar{R}, \bar{x}]$ we add the rule: $p_\psi(\bar{R}, \bar{X}) \leftarrow not\ p_{\psi_1}(\bar{R}, \bar{X})$.
- If $\psi \equiv \psi_1[\bar{R}, \bar{x}] \wedge \psi_2[\bar{Q}, \bar{y}]$ we add the rule: $p_\psi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\psi_1}(\bar{R}, \bar{X}), p_{\psi_2}(\bar{Q}, \bar{Y})$.
- $\psi \equiv \psi_1[\bar{R}, \bar{x}] \vee \psi_2[\bar{Q}, \bar{y}]$ we add the rules: $p_\psi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\psi_1}(\bar{R}, \bar{X})$ and $p_\psi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\psi_2}(\bar{Q}, \bar{Y})$.
- If $\psi \equiv \exists x_i \psi_1[\bar{R}, \bar{x}]$ we add the rule: $p_\psi(\bar{R}, \bar{X} - X_i) \leftarrow p_{\psi_1}(\bar{R}, \bar{X})$.

In the case where $\psi \equiv \exists R \psi_1$ we do nothing. For a given $SO(\exists)$ formula ψ, the last rule will be of the form $p_\psi(\bar{R}) \leftarrow \ldots$ (the arguments will only be the predicate variables which are existentially quantified). Lastly we want $|\mathcal{U}| = U_{P_\psi}$ to be true. Since the Herbrand universe consists of all the individual constants that appear in the datalog program, we can just add a predicate of arity $||\mathcal{U}||$ where each argument is an element of $|\mathcal{U}|$.

Example 3.1.1 *Let $\psi \equiv \exists R \exists Q \forall x(R(x) \vee Q(x))$. Then the second-order datalog program will be:*

$$
\begin{aligned}
&p_R(R, X) \leftarrow R(X).\\
&p_Q(Q, X) \leftarrow Q(X).\\
&p_\vee(R, Q, X) \leftarrow p_R(R, X).\\
&p_\vee(R, Q, X) \leftarrow p_Q(Q, X).\\
&p_\neg(R, Q, X) \leftarrow not\, p_\vee(R, Q, X).\\
&p_{\exists x}(R, Q) \leftarrow p_\neg(R, Q, X).\\
&p_\psi(R, Q) \leftarrow not\, p_{\exists x}(R, Q).
\end{aligned}
$$

Let $\phi \in SO(\exists)$ and $\mathcal{U}$ interpretation. Remember that if L_ϕ is the language that ϕ expresses then $\models_\mathcal{U} \phi[s] \Leftrightarrow bin(\mathcal{U}) \in L_\phi$. Different interpretations give, possibly, different meaning in the predicates that a formula uses. We are going to use this fact and say: *The input database of the constructed datalog program will be a set of facts. Those facts encode the meaning a specific interpretation gives to predicates.*

Example 3.1.2 *Let $\phi \equiv \exists R \forall x(R(x) \wedge \neg q(x))$ and $\mathcal{U}$ interpretation with $|\mathcal{U}| = \{a, b, c, d\}$ and $q^\mathcal{U} = \{a, b, c\}$. The constructed program is:*

$$
\begin{aligned}
&in(a, b, c, d).\\
&q(a).\\
&q(b).\\
&q(c).\\
\\
&p_R(R, X) \leftarrow R(X).\\
&p_q(X) \leftarrow q(X).\\
&p_{\neg q}(X) \leftarrow not\, p_q(X).\\
&p_\wedge(R, X) \leftarrow p_R(R, X), p_{\neg q}(X).\\
&p_\neg(R, X) \leftarrow not\, p_\wedge(R, X).\\
&p_{\exists x}(R) \leftarrow p_\neg(R, X).\\
&p_\phi(R) \leftarrow not\, p_{\exists x}(R).
\end{aligned}
$$

The Herbrand universe of the program is $U_P = \{a, b, c, d\} = |\mathcal{U}|$. The input consists of the three facts $q(a), q(b), q(c)$ and the predicate $in/|U_P|$, which just contains every element of the universe of U

Theorem 3.1.1 *Let $\phi \in SO(\exists)$, $\mathcal{U}$ an interpretation and $P_{(\phi,\mathcal{U})}$ the constructed program according to $\mathcal{U}$. Then we have :*

$$\mathcal{U}_s(\phi) = v \Leftrightarrow M_{P_{(\phi,\mathcal{U})}}(p_\phi) = v, \texttt{where}\ v = \{false, true\}$$

In other words, the constructed program behaves ”correctly” and the well-founded model assigns either the value false or the value true in the rule corresponding to the logic formula.

Corollary 3.1.1 *Second-order datalog under WFS captures the complexity class NP.*

Proof 3.1.1 *(By structural induction)*
$(\Rightarrow)$
Let $\phi \in SO(\exists)$, $\mathcal{U}$ an interpretation.

- *$\phi \equiv t_1 = t_2$, t_1, t_2 constants. We have that $\models_{\mathcal{U}} t_1 = t_2[s] \Leftrightarrow \ldots\ t_1 = t_2 = a \in |\mathcal{U}|$.Our program is the single rule $p_\phi \leftarrow t_1 \approx t_2$. Since $t_1 = t_2 = a$ we get that $M_P(p_\phi) = true.$*
- *$\phi \equiv x_1 = x_2$, where x_1, x_2 are variables.*
  We have that $\models_{\mathcal{U}} x_1 = x_2[s] \Leftrightarrow \ldots\ s(x_1) = s(x_2)$ for some function state s and the program is the rule $p_\phi(X_1, X_2) \leftarrow X_1 \approx X_2$. In the well-founded model we will have $M_P(p_\phi)(s(x_1), s(x_2)) = true.$

The value $v = false \Leftrightarrow \not\models_{\mathcal{U}} \phi[s]$ is treated accordingly.
For induction hypothesis and ϕ_1, ϕ_2 already constructed formulas let it be true that if $\mathcal{U}_s(\phi_{1,2}) = v$ then $M_{P_{(\phi_{1,2},\mathcal{U})}}(p_{\phi_{1,2}}) = v.$

- *$\phi \equiv \neg\phi_1$*
  We have $\models_{\mathcal{U}} \neg\phi_1[s] \Leftrightarrow \not\models_{\mathcal{U}} \phi_1[s]$. And the rules $\ldots, p_{\phi_1}, p_\phi(\bar{R}, \bar{X}) \leftarrow not\, p_{\phi_1}(\bar{R}, \bar{x})$. By induction hypothesis $M_P(p_{\phi_1})(\bar{r}, \bar{x}) = false$, for appropriate $\bar{r}, \bar{x}$, so $M_P(p_\phi)(\bar{r}, \bar{x})$ is true.
  In the case where $\not\models_{\mathcal{U}} \neg\phi_1[s] \Leftrightarrow \models_{\mathcal{U}} \phi_1[s]$, we have that $M_P(p_{\phi_1}) = true$ and $M_P(not\, p_{\phi_1}) = false$. But since M_P is minimal we get that $M_P(p_\phi) = false.$
- *$\phi \equiv \phi_1 \wedge \phi_2$*
  We have $\models_{\mathcal{U}} \phi_1 \wedge \phi_2 \Leftrightarrow \models_{\mathcal{U}} \phi_1[s]$ and $\models_{\mathcal{U}} \phi_2[s]$ which by induction hypothesis gives us $M_P(p_{\phi_1}) = true$ and $M_P(p_{\phi_2}) = true$. Thus we get that $M_P(p_\phi) = true.$
  In the case where $\not\models_{\mathcal{U}} \phi_1 \wedge \phi_2[s]$, we have that either $\not\models_{\mathcal{U}} \phi_1[s]$ or $\not\models_{\mathcal{U}} \phi_2[s]$. Which gives us either $M_P(p_{\phi_1}) = false$ or $M_P(p_{\phi_2}) = false$ (or both). In any case, we get that $M_P(p_\phi) = false.$
- *$\phi \equiv \phi_1 \vee \phi_2$*
  We have $\models_{\mathcal{U}} \phi_1 \vee \phi_2 \Leftrightarrow \models_{\mathcal{U}} \phi_1[s]$ or $\models_{\mathcal{U}} \phi_2[s]$ which by induction hypothesis gives us $M_P(p_{\phi_1}) = true$ or $M_P(p_{\phi_2}) = true$. Thus we get that $M_P(p_\phi) = true.$
  In the case where $\not\models_{\mathcal{U}} \phi_1 \vee \phi_2[s]$, we have that $\not\models_{\mathcal{U}} \phi_1[s]$ and $\not\models_{\mathcal{U}} \phi_2[s]$. Which gives us $M_P(p_{\phi_1}) = false$ and $M_P(p_{\phi_2}) = false$.Thus we get that $M_P(p_\phi) = false.$
- *$\phi \equiv \exists y\phi_1$*
  We have $\models_{\mathcal{U}} \exists y\phi_1[s]$ iff there exists $a \in |\mathcal{U}|$ such that $\models_{\mathcal{U}} \phi_1[s(y/a)]$. In our program we have $p_\phi(\bar{R}, \bar{X})$:-$p_{\phi_1}(\bar{R}, \bar{X}, Y)$. We have $M_P(p_{\phi_1})(\bar{r}, \bar{x}, a) = true$ and so $M_P(p_\phi) = true.$

Now for the other direction we have:
$(\Leftarrow)$

- $p_\phi \leftarrow t_1 \approx t_2$. *and* $\phi \equiv t_1 = t_2$ *where* t_1, t_2 *constants.*
  If $M_P(p_\phi) = true$ *then* $M_P(t_1 = t_2) = true \Rightarrow t_1 = t_2 = a \in U_P = |\mathcal{U}|$.
  Which means that $\models_{\mathcal{U}} \phi[s]$ *is true for every (Herbrand) interpretation.*

- $p_\phi(X_1, X_2) \leftarrow X_1 \approx X_2$. *and* $\phi \equiv X_1 = X_2$ *where* X_1, X_2 *variables.*
  If $M_P(p_\phi)(X_1, X_2) = true \Rightarrow M_P(s(X_1) = s(X_2)) = true$*, where* $s(X_1) = s(X_2) = a \in U_P$*. So* $\models_{\mathcal{U}} \phi[s]$ *is true.*

For induction hypothesis and ϕ *already constructed formula let it be true that if* $M_P(\phi) = v$ *then* $\mathcal{U}_s(\phi) = v$*.*

- $p_\phi(\bar{R}, \bar{X}) \leftarrow not\, p_{\phi_1}(\bar{R}, \bar{X})$ *and* $\phi \equiv \neg\phi_1$*.*
  We have $M_P(p_\phi)(\bar{r}, \bar{x}) = true \Rightarrow M_P(p_{\phi_1})(\bar{r}, \bar{x}) = false \Rightarrow$
  $\not\models_{\mathcal{U}} \phi_1[s(\bar{R}/\bar{r}, \bar{X}/\bar{x})] \Rightarrow \models_{\mathcal{U}} \phi[s(\bar{R}/\bar{r}, \bar{X}/\bar{x})]$.
  In case $M_P(p_\phi)(\bar{r}, \bar{x}) = false \Rightarrow M_P(p_{\phi_1})(\bar{r}, \bar{x}) = true \Rightarrow$
  $\models_{\mathcal{U}} \phi_1[s(\bar{R}/\bar{r}, \bar{X}/\bar{x})] \Rightarrow \not\models_{\mathcal{U}} \phi[s(\bar{R}/\bar{r}, \bar{X}/\bar{x})]$.

- $p_\phi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\phi_1}(\bar{R}, \bar{X}), p_{\phi_2}(\bar{Q}, \bar{Y})$*. We have* $M_P(p_\phi)(\bar{r}, \bar{q}, \bar{x}, \bar{y}) = true \Rightarrow M_P(p_{\phi_1})(\bar{r}, \bar{x}) = true$ *and* $M_P(p_{\phi_2})(\bar{q}, \bar{y}) = true$*. By induction hypothesis we get* $\models_{\mathcal{U}} \phi_1[s]$ *and* $\models_{\mathcal{U}} \phi_2[s]$ *are both true and so* $\models_{\mathcal{U}} \phi[s]$ *is true as well.*
  In case $M_P(p_\phi)(\bar{r}, \bar{q}, \bar{x}, \bar{y}) = false$*, we get that either* $M_P(p_{\phi_1})(\bar{r}, \bar{x}) = false$ *or* $M_P(p_{\phi_2})(\bar{q}, \bar{y}) = false$ *and by induction hypothesis* $\not\models_{\mathcal{U}} \phi_1[s]$ *or* $\not\models_{\mathcal{U}} \phi_2[s]$*.*

- $p_\phi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\phi_1}(\bar{R}, \bar{X})$*.,* $p_\phi(\bar{R}, \bar{Q}, \bar{X}, \bar{Y}) \leftarrow p_{\phi_2}(\bar{Q}, \bar{Y})$*. We have* $M_P(p_\phi)(\bar{r}, \bar{q}, \bar{x}, \bar{y}) = true \Rightarrow M_P(p_{\phi_1})(\bar{r}, \bar{x}) = true$ *or* $M_P(p_{\phi_2})(\bar{q}, \bar{y}) = true$ *and by induction hypothesis* $\models_{\mathcal{U}} \phi_1[s]$ *or* $\models_{\mathcal{U}} \phi_2[s]$ *which gives that* $\models_{\mathcal{U}} \phi[s]$*. For* $M_P(p_\phi)(\bar{r}, \bar{q}, \bar{x}, \bar{y}) = false$ *accordingly.*

- $p_\phi(\bar{R}, \bar{X}) \leftarrow p_{\phi_1}(\bar{R}, \bar{X}, Y)$*, where* $Y \notin \bar{X}$*. If* $M_P(p_\phi)(\bar{r}, \bar{x}) = true$ *then there exists* $a \in U_P$ *such that* $M_P(p_{\phi_1})(\bar{r}, \bar{x}, a) = true$ *and by induction hypothesis there exists an* $a \in |\mathcal{U}|$ *such that* $\models_{\mathcal{U}} \phi_1[s(y/a)] \Rightarrow \models_{\mathcal{U}} \phi[s]$*.*

3.2 Encoding function symbols, input order and nondeterminism

Notice that in $SO(\exists)$ logic we can have first-order function constants while in datalog we cannot. The above transformation doesn't refer to any rule about function symbols and we have to somehow work around it. Remember that our interpretations are finite and thus the meaning they give in a function symbol f of arity n, is a finite function $f : D^n \to D$ where D is the interpretation's universe. We can easily replace each such function of arity n with a predicate of arity $n + 1$. The function (predicate) constants will also be used as input (since their meaning is determined by interpretations). What about equality between two functions (which in our case is equality between two relations)?

Let p, q be two first-order predicates of arity $n + 1$ (encoding two functions f, g of arity n). Then $f = g$ can be expressed as:

$$\forall x_1 \forall x_2 \dots \forall x_{n+1} \big(p(x_1, x_2, \dots, x_{n+1}) \leftrightarrow q(x_1, x_2, \dots, x_{n+1})\big)$$

Which can easily be transformed into a second-order datalog program.

Let us now talk about the ordering assumption we mentioned in the introduction section and how it was bypassed thanks to the power of nondeterminism. Remember that in [2], in order to prove that *k-order datalog captures $(k-1)$-EXPTIME* an assumption about the input was necessary. The input would come in a specific form, described below.
Let $\Sigma = \{a, b\}$ be an alphabet and $w = abab$ be a string. Then the input is:

$$\begin{aligned}&input(0\,a\,1).\\&input(1\,b\,2).\\&input(2\,a\,3).\\&input(3\,b\,end).\end{aligned}$$

Where $\{0, \dots, |w|\} \cup \{end\}$ are constants.

Notice that in this thesis we did not mention anything similar. That is because $SO(\exists)$ is powerful enough to existentially quantify a linear ordering on the universe of a structure. Instead of assuming a total order in the input, we use nondeterminism (that second-order logic gives us) to guess such an ordering. In fact the original proof of Fagin's theorem [10], does exactly that. To conclude, if we want to compute a problem that requires an ordering, let's say k-clique, all we have to do is to express in first-order a total order, existentially quantify it, express the k-clique problem with $SO(\exists)$ and transform them into a second-order WFS datalog program according to the transformation of this thesis.

4. CONCLUSIONS AND FUTURE WORK

Observe that in [2], second-order positive datalog captures $EXPTIME$, while here we say that if we add negation under the well-founded semantics it captures NP, a subset of exponential time. Now one would say that by adding negation to a language, you should make it stronger rather than weaker. But there is a catch. First of all the result in [2] is depending on the ordering assumption that was made. Without that second-order positive datalog would not be able to express even simple languages, not to mention languages in $EXPTIME$. Moreover Theorem 3.1.1 suggests that second-order WFS datalog can express at least all languages in NP. This means that it is possible, and most likely, that second-order WFS datalog is more powerful than that. An intuition behind the above idea is that $SO(\exists)$ is quite a restrictive fragment of second-order logic regarding the structure of the $SO(\exists)$-formulas. As a result the programs that are constructed with the transformation, inherit the same restriction on their structure. Let us mention that second-order logic, in whole, captures the polynomial-time hierarchy PH and extensions of it, like $SO(TC)$(extended with a transitive closure operator) and $SO(lfp)$(second-order extended with a least fix point operator) can go up to $PSPACE$ and $EXPTIME$ accordingly [12] [13]. So what if we didn't restrict our datalog programs as well? Furthermore a similar transformation from second-order logic (or even from one of the above mentioned extensions) to second-order WFS datalog might be feasible. One possible helpful tool would be to introduce existential predicate variables[1] in the semantics of higher-order WFS datalog and thus make it easier to express languages inside PH. Another approach would be to follow exactly the logic of the proof that was used for positive higher-order datalog [2], with only one difference. Instead of assuming that the input comes in a specific way, use second-order datalog (with negation) to somehow guess that order and then continue with the simulation of the Turing machine.

To conclude, second-order WFS datalog seems to be a powerful logic programming language suitable for solving problems with a simple and compact way of programming. As for the transformation introduced in this thesis, on the one hand it gives a direct way to construct logic programs that solve $\mathcal{NP}$ problems, but on the other it is not the most elegant way of programming. However to have a theoretical backup that your language can indeed express the problem you try to solve is quite meaningful and knowing that you can try to find the optimal way to solve it. Lastly, it seems that there is a lot more research that fits in the complexity theoretic study of higher-order logic programming which might give useful results, both theoretical and practical.

[1] R is an existential predicate variable in the rule $p \leftarrow \lambda X.\exists R(R\ X)$

BIBLIOGRAPHY

[1] A. Charalambidis, K. Handjopoulos, P. Rondogiannis, and W. W. Wadge. Extensional higher-order logic programming, 2011.

[2] ANGELOS CHARALAMBIDIS, CHRISTOS NOMIKOS, and PANOS RONDOGIANNIS. The expressive power of higher-order datalog. *Theory and Practice of Logic Programming*, 19(5-6):925–940, sep 2019.

[3] Angelos Charalambidis and Panos Rondogiannis. Constructive negation in extensional higher-order logic programming. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014.

[4] ANGELOS CHARALAMBIDIS, PANOS RONDOGIANNIS, and IOANNA SYMEONIDOU. Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. *Theory and Practice of Logic Programming*, 18(3-4):421–437, jul 2018.

[5] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, sep 2001.

[6] Marc Denecker, Victor Marek, and Mirosław Truszczyński. *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*, pages 127–144. Springer US, Boston, MA, 2000.

[7] Marc Denecker, Victor W Marek, and Mirosław Truszczyński. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, 2004.

[8] R. Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974.

[9] Ronald Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proc.*, 7, 01 1974.

[10] Ronald Fagin. Finite-model theory - a personal perspective. *Theoretical Computer Science*, 116(1):3–31, 1993.

[11] Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1):25–51, 2002. Mathematical Foundations of Programming Semantics 1996.

[12] D. Harel and D. Peleg. On static logics, dynamic logics, and complexity classes. *Information and Control*, 60(1):86–102, 1984.

[13] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 1998.

[14] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[15] Christos H. Papadimitriou. A note the expressive power of prolog. *Bull. EATCS*, 26:21–22, 1985.

[16] Panos Rondogiannis and Ioanna Symeonidou. Extensional semantics for higher-order logic programs with negation. *Log. Methods Comput. Sci.*, 14(2), 2018.

[17] J.S. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995.

[18] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 137–146, New York, NY, USA, 1982. Association for Computing Machinery.

[19] Guizhen Yang, Michael Kifer, and Chang Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. volume 2888, pages 671–688, 11 2003.