Guarded Open Answer Set Programming

Stijn Heymans, Davy Van Nieuwenborgh[⋆], and Dirk Vermeir[⋆⋆]

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{sheymans,dvnieuwe,dvermeir}@vub.ac.be

Abstract. Open answer set programming (OASP) is an extension of answer set programming where one may ground a program with an arbitrary superset of the program's constants. We define a fixed point logic (FPL) extension of Clark's completion such that open answer sets correspond to models of FPL formulas and identify a syntactic subclass of programs, called (loosely) guarded programs. Whereas reasoning with general programs in OASP is undecidable, the FPL translation of (loosely) guarded programs falls in the decidable (loosely) guarded fixed point logic (μ(L)GF).
Moreover, we reduce normal closed ASP to loosely guarded OASP, enabling a characterization of an answer set semantics by μLGF formulas. Finally, we relate guarded OASP to Datalog LITE, thus linking an answer set semantics to a semantics based on fixed point models of extended stratified Datalog programs. From this correspondence, we deduce 2-EXPTIME-completeness of satisfiability checking w.r.t. (loosely) guarded programs.

1 Introduction

A problem with finite closed answer set programming (ASP)[10] is that all significant constants have to be present in the program in order to capture the intended semantics. E.g., a program with a rule $r : p(X) \leftarrow not\ q(X)$ and a fact $q(a)$ has the unique answer set $\{q(a)\}$ and thus leads to the conclusion that p is not satisfiable. However, if r is envisaged as a schema constraint and a is just one possible data instance, this conclusion is wrong: other data makes p satisfiable.

This problem was solved in [11] by introducing k new constants, k finite, and grounding the program with this extended universe; the answer sets of the grounded program were called *k-belief sets*. We extended this idea, e.g. in [16], by allowing for arbitrary, thus possibly infinite, universes. *Open answer sets* are then pairs (U, M) with M an answer set of the program grounded with U. The above program has an open answer set $(\{x, a\}, \{q(a), p(x)\})$ where p is satisfiable.

Characteristic about (O)ASP is its treatment of negation as failure (naf): one guesses an interpretation for a program, computes the program without naf (the GL-reduct[10]),

[⋆] Supported by the FWO.

[⋆⋆] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

calculates the iterated fixed point of this reduct, and checks whether this fixed point equals the initial interpretation. We compile these external manipulations, i.e. not expressible in the language of programs itself, into fixed point logic (FPL)[14] formulas that are at most quadratic in the size of the original program. First, we rewrite an arbitrary program as a program containing only one designated predicate p and (in)equality; this makes sure that when calculating a fixed point of the predicate variable p, it constitutes a fixed point of the whole program. In the next phase, such a *p-program P* is translated to FPL formulas $\texttt{comp}(P)$. $\texttt{comp}(P)$ ensures satisfiability of program rules by formulas comparable to those in Clark's completion. The specific answer set semantics is encoded by formulas indicating that for each atom $p(\boldsymbol{x})$ in the model there must be a true rule body that motivates the atom, and this in a minimal way, i.e. using a fixed point predicate. Negation as failure is correctly handled by making sure that only those rules that would be present in the GL-reduct can be used to motivate atoms.

In [5], Horn clauses were translated to FPL formulas and in [12] reasoning with an extension of stratified Datalog was reduced to FPL, but, to the best of our knowledge, this is the first encoding of an answer set semantics in FPL.

In [21, 19], ASP with (finite) propositional programs is reduced to propositional satisfiability checking. The translation makes the loops in a program explicit and ensures that atoms $p(\boldsymbol{x})$ are motivated by bodies outside of these loops. Although this is an elegant characterization of answer sets in the propositional case, the approach does not seem to hold for OASP, where programs are not propositional but possibly ungrounded and with infinite universes. Instead, we directly use the built-in "loop detection" mechanism of FPL, which enables us to go beyond propositional programs.

Translating OASP to FPL is thus interesting in its own right, but it also enables the analysis of decidability of OASP via decidability results of fragments of FPL. Satisfiability checking of a predicate p w.r.t. a program, i.e. checking whether there exists an open answer set containing some $p(\boldsymbol{x})$, is undecidable, e.g. the undecidable domino problem can be reduced to it[15]. It is well-known that satisfiability checking in FOL is undecidable, and thus the extension to FPL is too. However, expressive decidable fragments of FPL have been identified[14]: *(loosely) guarded fixed point logic* (μ(L)GF) extends the *(loosely) guarded fragment* (L)GF of FOL with fixed point predicates.

GF was identified in [2] as a fragment of FOL satisfying properties such as decidability of reasoning and the tree-model property, i.e. every model can be rewritten as a tree-model. The restriction of quantified variables by a *guard*, an atom containing the variables in the formula, ensures decidability in GF. Guards are responsible for the tree-model property of GF (where the concept of tree is adapted for predicates with arity larger than 2), which in turn enables tree-automata techniques for showing decidability of satisfiability checking. In [4], GF was extended to LGF where guards can be conjunctions of atoms and, roughly, every pair of variables must be together in some atom in the guard. Satisfiability checking in both GF and LGF is 2-EXPTIME-complete[13], as are their extensions with fixed point predicates μGF and μLGF[14].

We identify a syntactically restricted class of programs, *(loosely) guarded programs ((L)GPs)*, for which the FPL translation falls in μ(L)GF, making satisfiability checking w.r.t. (L)GPs decidable and in 2-EXPTIME. In LGPs, rules have a set of atoms, the guard, in the positive body, such that every pair of variables in the rule appears together in an atom in that guard. GPs are the restriction of LGPs where guards must consist of exactly one atom. Programs under the normal answer set semantics can be rewritten as LGPs under the open answer set semantics by guarding all variables with atoms that can only deduce constants from the original program. Besides the desirable property that OASP with LGPs is thus a proper decidable extension of normal ASP, this yields that satisfiability checking w.r.t. LGPs is, at least, NEXPTIME-hard.

Datalog LITE[12] is a language based on stratified Datalog with input predicates where rules are monadic or guarded and may have generalized literals in the body, i.e. literals of the form $\forall \boldsymbol{Y} \cdot a \Rightarrow b$ for atoms a and b. It has an appropriately adapted bottom-up fixed point semantics. Datalog LITE was devised to ensure linear time model checking while being expressive enough to capture *computational tree logic*[8] and alternation-free μ-calculus[18]. Moreover, it was shown to be equivalent to alternation-free μGF. Our reduction of GPs to μGF, ensures that we have a reduction from GPs to Datalog LITE, and thus couples the answer set semantics to a fixed point semantics based on stratified programs. Intuitively, the guess for an interpretation in the answer set semantics corresponds to the input structure one feeds to the stratified Datalog program. The translation from GPs to Datalog LITE needs only one stratum to subsequently perform the minimality check of answer set programming.

The other way around, we reduce satisfiability checking in recursion-free Datalog LITE to satisfiability checking w.r.t. GPs. Recursion-free Datalog LITE is equivalent to GF[12], and, since satisfiability checking of GF formulas is 2-EXPTIME-hard[13], we obtain 2-EXPTIME-completeness for satisfiability checking w.r.t. (L)GPs.

In [16, 17], other decidable classes of programs under the open answer set semantics were identified; decidability was attained differently than for (L)GPs, by reducing OASP to finite ASP. Although the therein identified *conceptual logic programs* are more expressive in some aspects (they allow for a more liberal use of inequality), they are less expressive in others, e.g. the use of predicates is restricted to unary and binary ones. Moreover, the definition of (L)GPs is arguably more simple compared to the often intricate restrictions on the rules in conceptual logic programs.

The remainder of the paper is organized as follows. After recalling the open answer set semantics in Section 2, we reduce reasoning under the open answer set semantics to reasoning with FPL formulas in Section 3. Section 4 describes guarded OASP, together with a 2-EXPTIME complexity upper bound and a reduction from finite ASP to loosely guarded OASP. Section 5 discusses the relationship with Datalog LITE and establishes 2-EXPTIME-completeness for (loosely) guarded open answer set programming. Section 6 contains conclusions and directions for further research. Due to space restrictions, proofs have been omitted; they can be found in [15].

2 Open Answer Set Semantics

We recall the open answer set semantics from [16]. *Constants*, *variables*, *terms*, and *atoms* are defined as usual. A *literal* is an atom $p(\boldsymbol{t})$ or a *naf-atom* $not\ p(\boldsymbol{t})$.[1] The *positive part* of a set of literals α is $\alpha^+ = \{p(\boldsymbol{t}) \mid p(\boldsymbol{t}) \in \alpha\}$ and the *negative part*

[1] We have no negation $\neg$, however, programs with $\neg$ can be reduced to programs without it, see e.g. [20].

of α is $\alpha^- = \{p(\boldsymbol{t}) \mid not\ p(\boldsymbol{t}) \in \alpha\}$. We assume the existence of binary predicates $=$ and $\neq$, where $t = s$ is considered as an atom and $t \neq s$ as $not\ t = s$. E.g. for $\alpha = \{X \neq Y, Y = Z\}$, we have $\alpha^+ = \{Y = Z\}$ and $\alpha^- = \{X = Y\}$. A *regular* atom is an atom that is not an equality atom. For a set X of atoms, $not\ X = \{not\ l \mid l \in X\}$.

A *program* is a countable set of rules $\alpha \leftarrow \beta$, where α and β are finite sets of literals, $|\alpha^+| \leq 1$, and $\forall t, s \cdot t = s \notin \alpha^+$, i.e. α contains at most one positive atom, and this atom cannot be an equality atom.[2] The set α is the *head* of the rule and represents a disjunction of literals, while β is called the *body* and represents a conjunction of literals. If $\alpha = \emptyset$, the rule is called a *constraint*. *Free rules* are rules of the form $q(\boldsymbol{t}) \lor not\ q(\boldsymbol{t}) \leftarrow$ for a tuple $\boldsymbol{t}$ of terms; they enable a choice for the inclusion of atoms. We call a predicate p free if there is a free rule $p(\boldsymbol{t}) \lor not\ p(\boldsymbol{t}) \leftarrow$. Atoms, literals, rules, and programs that do not contain variables are *ground*.

For a program P, let $cts(P)$ be the constants in P, $vars(P)$ its variables, and $preds(P)$ its predicates. A *universe* U for P is a non-empty countable superset of the constants in P: $cts(P) \subseteq U$. We call P_U the ground program obtained from P by substituting every variable in P by every possible constant in U. Let $\mathcal{B}_P$ be the set of regular atoms that can be formed from a ground program P.

An *interpretation* I of a ground P is any subset of $\mathcal{B}_P$. For a ground regular atom $p(\boldsymbol{t})$, we write $I \models p(\boldsymbol{t})$ if $p(\boldsymbol{t}) \in I$; For an equality atom $p(\boldsymbol{t}) \equiv t = s$, we have $I \models p(\boldsymbol{t})$ if s and t are equal terms. We have $I \models not\ p(\boldsymbol{t})$ if $I \not\models p(\boldsymbol{t})$. For a set of ground literals X, $I \models X$ if $I \models l$ for every $l \in X$. A ground rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$, i.e. r is *applied* whenever it is *applicable*. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. For a ground program P without *not*, an interpretation I of P is a *model* of P if I satisfies every rule in P; it is an *answer set* of P if it is a subset minimal model of P. For ground programs P containing *not*, the *GL-reduct*[10] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $I \models not\ \beta^-$ and $I \models \alpha^-$. I is an *answer set* of a ground P if I is an answer set of P^I.

In the following, a program is assumed to be a finite set of rules; infinite programs only appear as byproducts of grounding a finite program with an infinite universe. An *open interpretation* of a program P is a pair (U, M) where U is a universe for P and M is an interpretation of P_U. An *open answer set* of P is an open interpretation (U, M) of P with M an answer set of P_U. An n-ary predicate p in P is *satisfiable* if there is an open answer set (U, M) of P and a $\boldsymbol{x} \in U^n$ such that $p(\boldsymbol{x}) \in M$. We assume that when satisfiability checking a predicate p, p is always non-free, i.e. there are no free rules with p in the head. Note that satisfiability checking of a free n-ary predicate p w.r.t. P can always be reduced to satisfiability checking of a new non-free n-ary predicate p' w.r.t. $P \cup \{p'(\boldsymbol{X}) \leftarrow p(\boldsymbol{X})\}$. Note that this is a linear reduction.

3 Open Answer Set Programming via Fixed Point Logic

We assume without loss of generality that the set of constants and the set of predicates in a program are disjoint and that each predicate q has one associated arity, e.g. $q(x)$

[2] The condition $|\alpha^+| \leq 1$ ensures that the GL-reduct is non-disjunctive.

and $q(x, y)$ are not allowed. A program P is a *p-program* if p is the only predicate in P different from the (in)equality predicate. We can rewrite any program P as an equivalent p-program P_p by replacing every regular m-ary atom $q(\boldsymbol{t})$ in P by $p(\boldsymbol{t}, \mathbf{0}, q)$ where p has arity n, with n the maximum of the arities of predicates in P augmented by 1, **0** a sequence of new constants 0 of length $n - m - 1$, and q a new constant with the same name as the original predicate. Furthermore, in order to avoid interference of the new constants, we add for every variable X in a non-free rule $r \in P$ and for every newly added constant a in P_p, $X \neq a$ to the body. E.g., the rule $h(a, b) \leftarrow q(X)$ in P corresponds to $p(a, b, h) \leftarrow p(X, \mathit{0}, q), X \neq \mathit{0}, X \neq h, X \neq q$ in P_p.

Proposition 1. *Let P be a program and q a predicate in P. q is satisfiable w.r.t. P iff there is an open answer set (U', M') of the p-program P_p with $p(\boldsymbol{x}, \mathbf{0}, q) \in M'$.*

The translation of a program to a p-program does not influence the complexity of reasoning, i.e. the size of P_p is linear in the size of P. By Proposition 1, we can focus attention on p-programs only. Since p-programs have open answer sets consisting of one predicate p, fixed points calculated w.r.t. p yield minimal models of the whole program as we will show in Proposition 2.

In [5], a similar motivation drives the reduction of Horn clauses to clauses consisting of only one defined predicate. Their encoding does not introduce new constants to identify old predicates and depends entirely on the use of (in)equality. However, to account for databases consisting of only one element, [5] needs an additional transformation that unfolds bodies of clauses.

We assume that FOL interpretations have the same form as open interpretations: a pair (U, M) corresponds with the FOL interpretation M over the domain U. Furthermore, we consider FOL with equality such that equality is always interpreted as the identity relation over U. *(Least) Fixed Point Logic (FPL)* is defined along the lines of [14]. *Fixed point formulas* are of the form

$$[\mathrm{LFP}\ W\boldsymbol{X}.\psi(W, \boldsymbol{X})](\boldsymbol{X}) \ , \tag{1}$$

where W is an n-ary predicate variable, $\boldsymbol{X}$ is an n-ary sequence of variables, $\psi(W, \boldsymbol{X})$ is a FOL formula where all free variables are contained in $\boldsymbol{X}$ and where W appears only positively in $\psi(W, \boldsymbol{X})$.[3]

We associate with (1) and an interpretation (U, M) that does not interpret W, an operator $\psi^{(U,M)} : 2^{U^n} \to 2^{U^n}$ defined on sets S of n-ary tuples as $\psi^{(U,M)}(S) \equiv \{\boldsymbol{x} \in U^n \mid (U, M) \models \psi(S, \boldsymbol{x})\}$. By definition, W appears only positively in ψ such that $\psi^{(U,M)}$ is monotonic on sets of n-ary U-tuples and has a least fixed point, which we denote by $\mathrm{LFP}(\psi^{(U,M)})$. Finally, we have $(U, M) \models [\mathrm{LFP}\ W\boldsymbol{X}.\psi(W, \boldsymbol{X})](\boldsymbol{x})$ iff $\boldsymbol{x} \in \mathrm{LFP}(\psi^{(U,M)})$.

We can reduce a p-program P to equivalent FPL formulas $\mathtt{comp}(P)$. The *completion* $\mathtt{comp}(P)$ consists of formulas $a \neq b$ for different constants a and b in P making sure that constants are interpreted as different elements, where $a \neq b \equiv \neg(a = b)$. $\mathtt{comp}(P)$

[3] Since $\psi(W, \boldsymbol{X})$ is a FOL formula, we do not allow nesting of fixed point formulas. This restriction is sufficient for the FPL simulation of OASP, and, furthermore, it simplifies the notation since one does not have to take into account an extra function χ that gives meaning to free second-order variables different from W.

also contains the formula $\exists X \cdot \textbf{true}$ ensuring the existence of at least one element in the domain of an interpretation. Besides these technical requirements that match FOL interpretations with open interpretations, $\texttt{comp}(P)$ contains the formulas in $\texttt{fix}(P) \equiv \texttt{sat}(P) \cup \texttt{gl}(P) \cup \texttt{fpf}(P)$, which can be intuitively categorized as follows: $\texttt{sat}(P)$ ensures that a model of $\texttt{fix}(P)$ satisfies all rules in P, $\texttt{gl}(P)$ is an auxiliary component defining atoms that indicate when a rule in P belongs to the GL-reduct of P, and finally, $\texttt{fpf}(P)$ ensures that every model of $\texttt{fix}(P)$ is a minimal model of the GL-reduct in P; it uses the atoms defined in $\texttt{gl}(P)$ to select, for the calculation of the fixed point, only those rules in P that are in the GL-reduct of P.

We interpret a naf-atom $not\ a$ in a FOL formula as the literal $\neg a$. Moreover, we assume that, if a set X is empty, $\bigwedge X = \textbf{true}$ and $\bigvee X = \textbf{false}$. We further assume that the arity of p, the only predicate in a p-program, is n.

Definition 1. *Let P be a p-program. Then, $\texttt{fix}(P) \equiv \texttt{sat}(P) \cup \texttt{gl}(P) \cup \texttt{fpf}(P)$, where*

– $\texttt{sat}(P)$ *contains formulas*
$$\forall \boldsymbol{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{2}$$
for rules $\alpha \leftarrow \beta \in P$ with variables $\boldsymbol{Y}$,
– $\texttt{gl}(P)$ *contains formulas*
$$\forall \boldsymbol{Y} \cdot r(\boldsymbol{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg \beta^- \tag{3}$$
for rules $r : \alpha \leftarrow \beta \in P$ with variables $\boldsymbol{Y}$ and a new predicate r,
– $\texttt{fpf}(P)$ *contains the formula*
$$\forall \boldsymbol{X} \cdot p(\boldsymbol{X}) \Rightarrow [\mathrm{LFP}\ W\boldsymbol{X}.\phi(W, \boldsymbol{X})](\boldsymbol{X}) \tag{4}$$
with $\phi(W, \boldsymbol{X}) \equiv W(\boldsymbol{X}) \vee \bigvee_{r:p(\boldsymbol{t})\vee\alpha\leftarrow\beta\in P} E(r)$ and $E(r) \equiv \exists \boldsymbol{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p|W] \wedge r(\boldsymbol{Y})$, where $\boldsymbol{X} = X_1, \ldots, X_n$ are n new variables, $\boldsymbol{Y}$ are the variables in r, W is a new (second-order) variable and $\beta^+[p|W]$ is β^+ with p replaced by W.

The completion *of P is* $\texttt{comp}(P) \equiv \texttt{fix}(P) \cup \{a \neq b \mid a \neq b \in cts(P)\} \cup \{\exists X \cdot \textbf{\textit{true}}\}$.

The predicate W appears only positively in $\phi(W, \boldsymbol{X})$ such that the fixed point formula in (4) is well-defined. The first conjunct, $W(\boldsymbol{X})$, in $\phi(W, \boldsymbol{X})$ ensures that previously deduced tuples are deduced by the next application of the fixed point operator, i.e. $S \subseteq \phi^{(U,M)}(S)$. The disjunction $\bigvee_r E(r)$ makes sure that for each atom there is a rule r in the GL-reduct ($\exists \boldsymbol{Y} \cdot r(\boldsymbol{Y})$) with a true positive body that can motivate that atom.

Example 1. Take a p-program P with rule $r : p(X) \leftarrow p(X)$. $\texttt{comp}(P)$ is then such that $\texttt{sat}(P) = \{\forall X \cdot p(X) \Rightarrow p(X)\}$, ensuring that r is satisfied, and $\texttt{gl}(P) = \{\forall X \cdot r(X) \Leftrightarrow \textbf{true}\}$ says that r belongs to every GL-reduct since there are no naf-atoms. Finally, $\texttt{fpf}(P) = \{\forall X_1 \cdot p(X_1) \Rightarrow [\mathrm{LFP}\ WX_1.\phi(W, X_1)](X_1)\}$, with $\phi(W, X_1) \equiv W(X_1) \vee \exists X \cdot X_1 = X \wedge W(X) \wedge r(X)$.

Proposition 2. *Let P be a p-program. Then, (U, M) is an open answer set of P iff $(U, M \cup R)$ is a model of $\texttt{comp}(P)$, where $R \equiv \{r(\boldsymbol{y}) \mid r[\boldsymbol{Y}|\boldsymbol{y}] \in P_U^M, vars(r) = \boldsymbol{Y}\}$, i.e. the atoms corresponding to rules in the GL-reduct of P_U w.r.t. M.*[4]

Example 2. For a universe $U = \{x\}$, we have the unique open answer set $(U, \emptyset)$ of P in Example 1. Since U is non-empty, every open answer set with a universe U satisfies $\exists X \cdot \textbf{true}$. Both $(U, M_1 = \{p(x), r(x)\})$ and $(U, M_2 = \{r(x)\})$ satisfy $\texttt{sat}(P) \cup \texttt{gl}(P)$. Since $\text{LFP}(\phi^{(U,M_1)}) = \text{LFP}(\phi^{(U,M_2)}) = \emptyset$, only (U, M_2) satisfies $\texttt{fpf}(P)$; (U, M_2) corresponds exactly to the open answer set $(U, \emptyset)$ of P.

The completion in Definition 1 differs from Clark's completion[6] both in the presence of the fixed point construct in (4) and the atoms representing membership of the GL-reduct. For p-programs P, Clark's Completion $\texttt{ccomp}(P)$ does not contain $\texttt{gl}(P)$, and $\texttt{fpf}(P)$ is replaced by the formula $\forall \boldsymbol{X} \cdot p(\boldsymbol{X}) \Rightarrow \bigvee_{r : p(\boldsymbol{t}) \vee \alpha \leftarrow \beta \in P} D(r)$ with $D(r) \equiv \exists \boldsymbol{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta \wedge \bigwedge \alpha^-$. Program P in Example 1 is the OASP version of the classical example $p \leftarrow p$[19], for which there are FOL models of $\texttt{ccomp}(P)$ that do not correspond to any answer sets: both $(\{x\}, \{p(x)\})$ and $(\{x\}, \emptyset)$ are FOL models while only the latter is an open answer set of P.

Using Propositions 1 and 2, we can reduce satisfiability checking in OASP to satisfiability checking in FPL. Moreover, with c the number of constants in a program P, the number of formulas $a \neq b$ is $\frac{1}{2}c(c-1)$, and, since the rest of $\texttt{comp}(P)$ is linear in P, this yields a quadratic bound for the size of $\texttt{comp}(P)$.

Theorem 1. *Let P be a program and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \texttt{comp}(P_p)$ is satisfiable. Moreover, this reduction is quadratic.*

4 Guarded Open Answer Set Programming

We repeat the definitions of the *loosely guarded fragment*[4] of FOL as in [14]: *The loosely guarded fragment LGF of FOL is defined inductively as follows:*

(1) *Every relational atomic formula belongs to LGF.*
(2) *LGF is closed under propositional connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$.*
(3) *If $\psi(\boldsymbol{X}, \boldsymbol{Y})$ is in LGF, and $\alpha(\boldsymbol{X}, \boldsymbol{Y}) = \alpha_1 \wedge \ldots \wedge \alpha_m$ is a conjunction of atoms, then the formulas*
$$\exists \boldsymbol{Y} \cdot \alpha(\boldsymbol{X}, \boldsymbol{Y}) \wedge \psi(\boldsymbol{X}, \boldsymbol{Y})$$
$$\forall \boldsymbol{Y} \cdot \alpha(\boldsymbol{X}, \boldsymbol{Y}) \Rightarrow \psi(\boldsymbol{X}, \boldsymbol{Y})$$
belong to LGF (and $\alpha(\boldsymbol{X}, \boldsymbol{Y})$ is the guard of the formula), provided that free$(\psi) \subseteq$ free$(\alpha) = \boldsymbol{X} \cup \boldsymbol{Y}$ and for every quantified variable $Y \in \boldsymbol{Y}$ and every variable $Z \in \boldsymbol{X} \cup \boldsymbol{Y}$ there is at least one atom α_j that contains both Y and Z (where free(ψ) are the free variables of ψ).

The *loosely guarded (least) fixed point logic* μLGF is LGF extended with fixed point formulas (1) where $\psi(W, \boldsymbol{X})$ is a LGF formula[5] such that W does not appear in guards.

[4] We denote the substitution of $\boldsymbol{Y} = Y_1, \ldots, Y_d$ with $\boldsymbol{y} = y_1, \ldots, y_d$ in a rule r by $r[\boldsymbol{Y}|\boldsymbol{y}]$.
[5] Thus, in accordance with our definition of FPL, nesting of (guarded) fixed point logic formulas is not allowed.

The *guarded fragment* GF is defined as LGF where the guards are atoms instead of conjunctions of atoms. The *guarded fixed point logic* μGF is GF extended with fixed point formulas where $\psi(W, \boldsymbol{X})$ is a GF formula such that W does not appear in guards.

Definition 2. *A rule $r : \alpha \leftarrow \beta$ is* loosely guarded *if there is a $\gamma_b \subseteq \beta^+$ such that every two variables X and Y from r appear together in an atom from γ_b; we call γ_b a* body guard *of r. It is* fully loosely guarded *if it is loosely guarded and there is a $\gamma_h \subseteq \alpha^-$ such that every two variables X and Y from r appear together in an atom from γ_h; γ_h is called a* head guard *of r.*

A program P is a (fully) loosely guarded program ((F)LGP) *if every non-free rule in P is (fully) loosely guarded.*

Example 3. The rule in Example 1 is loosely guarded but not fully loosely guarded. A rule $a(Y) \lor not\ g(X, Y) \leftarrow not\ b(X), f(X, Y)$ has body guard $\{f(X,Y)\}$ and head guard $\{g(X,Y)\}$, and is thus fully loosely guarded.

Definition 3. *A rule is* guarded *if it is loosely guarded with a singleton body guard. It is* fully guarded *if it is fully loosely guarded with singleton body and head guards.*

A program P is a (fully) guarded program ((F)GP) *if every non-free rule in P is (fully) guarded.*

Every F(L)GP is a (L)GP, and we can rewrite every (L)GP as a F(L)GP.

Example 4. The rule $p(X) \leftarrow p(X)$ can be rewritten as $p(X) \lor not\ p(X) \leftarrow p(X)$ where the body guard is added to the negative part of the head to function as the head guard. Both programs are equivalent: for a universe U, both have the unique open answer set $(U, \emptyset)$.

Formally, we can rewrite every (L)GP P as an equivalent F(L)GP P^{f}, where P^{f} is P with every $\alpha \leftarrow \beta$ replaced by $\alpha \cup not\ \beta^+ \leftarrow \beta$. The body guard of a rule in a (loosely) guarded program P is then also a head guard of the corresponding rule in P^{f}, and P^{f} is indeed a fully (loosely) guarded program.

A rule is vacuously satisfied if the body of a rule in P^{f} is false and consequently the head does not matter; if the body is true then the newly added part in the head becomes false and the rule in P^{f} reduces to its corresponding rule in P.

Proposition 3. *Let P be a program. An open interpretation (U, M) of P is an open answer set of P iff (U, M) is an open answer set of P^{f}.*

Since we copy the positive bodies to the heads, the size of P^{f} only increases linearly in the size of P. Furthermore, the construction of a p-program retains the guardedness properties: P is a (F)LGP iff P_p is a (F)LGP. A similar property holds for (F)GPs.

For a fully (loosely) guarded p-program P, we can rewrite $\texttt{comp}(P)$ as the equivalent μ(L)GF formulas $\texttt{gcomp}(P)$. $\texttt{gcomp}(P)$ is $\texttt{comp}(P)$ with the following modifications:

– Formula $\exists X \cdot \textbf{true}$ is replaced by $\exists X \cdot X = X$, a formula guarded by $X = X$.

– Formula (2) is removed if $r : \alpha \leftarrow \beta$ is free and otherwise replaced by

$$\forall \boldsymbol{Y} \cdot \bigwedge \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \backslash \gamma_b) \vee \bigvee \beta^- ,$$

where γ_b is a body guard of r; we logically rewrite formula (2) such that it is (loosely) guarded. If r is a free rule of the form $q(\boldsymbol{t}) \vee \mathit{not}\ q(\boldsymbol{t}) \leftarrow$, we have $\forall Y \cdot \textbf{true} \Rightarrow q(\boldsymbol{t}) \vee \neg q(\boldsymbol{t}) \in \texttt{comp}(P)$, which is always true and can be removed.

– Formula (3) is replaced by the formulas $\forall \boldsymbol{Y} \cdot r(\boldsymbol{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^-$ and $\forall \boldsymbol{Y} \cdot \bigwedge \gamma_h \Rightarrow r(\boldsymbol{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \backslash \gamma_h)$, where γ_h is a head guard of r. We thus rewrite an equivalence as two implications. The first implication is guarded by $r(\boldsymbol{Y})$ and the second one is (loosely) guarded by the head guard of the rule – hence the need for a fully (loosely) guarded program, instead of just a (loosely) guarded one.

– For every $E(r)$ in (4), define $T \equiv \{t_i \notin cts(P) \mid 1 \leq i \leq n\}$, and replace $E(r)$ by

$$E'(r) \equiv \bigwedge_{t_i \notin T} X_i = t_i \wedge \exists \boldsymbol{Z} \cdot (\bigwedge \beta^+[p|W] \wedge r(\boldsymbol{Y}))[t_i \in T|X_i] ,$$

with $\boldsymbol{Z} = \boldsymbol{Y} \backslash T$, i.e. move all $X_i = t_i$ where t_i is constant out of the quantifier's scope, and remove the others by substituting each t_i in $\bigwedge \beta^+[p|W] \wedge r(\boldsymbol{Y})$ by X_i. This rewriting makes sure that every variable in the quantified part of $E'(R)$ is guarded by $r(\boldsymbol{Y})[t_i \in T|X_i]$.

Example 5. For the fully guarded p-program P containing a rule $p(X) \vee \mathit{not}\ p(X) \leftarrow p(X)$ with body and head guard $\{p(X)\}$, one has that $\texttt{sat}(P) = \{\forall X \cdot p(X) \Rightarrow p(X) \vee \neg p(X)\}$, $\texttt{gl}(P) = \{\forall X \cdot r(X) \Leftrightarrow p(X)\}$ and the formula $\phi(W, X_1)$ in $\texttt{fpf}(P)$ is $\phi(W, X_1) \equiv W(X_1) \vee \exists X \cdot X_1 = X \wedge W(X) \wedge r(X)$. $\texttt{gcomp}(P)$ does not modify $\texttt{sat}(P)$ and rewrites the equivalence in $\texttt{gl}(P)$ as two guarded implications. The rewritten $\phi(W, X_1)$ is $W(X_1) \vee (W(X_1) \wedge r(X_1))$.

For a fully (loosely) guarded p-program P, $\texttt{gcomp}(P)$ is a μ(L)GF formula, and it is logically equivalent to $\texttt{comp}(P)$, i.e. (U, M) is a model of $\texttt{comp}(P)$ iff (U, M) is a model of $\texttt{gcomp}(P)$. $\texttt{gcomp}(P)$ is a simple logical rewriting of $\texttt{comp}(P)$, with a size linear in the size of $\texttt{comp}(P)$. Using Proposition 3 and Theorem 1, satisfiability checking w.r.t. (L)GPs can be quadratically reduced to satisfiability checking of a μ(L)GF formula.

Theorem 2. *Let P be a (L)GP and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \texttt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable. Moreover, this reduction is quadratic.*

Since satisfiability checking for μ(L)GF is 2-EXPTIME-complete (Proposition [1.1] in [14]), we have the following upper complexity bound.

Theorem 3. *Satisfiability checking w.r.t. (L)GPs is in 2-*EXPTIME.

An answer set of a program P (in contrast with an *open* answer set) is defined as an answer set of the grounding of P with its own constants, i.e. M is an answer set of P if it is a minimal model of $P^M_{cts(P)}$. As is common in literature, we assume P contains at least one constant.

We can make any program loosely guarded and reduce the answer set semantics for programs to the open answer set semantics for loosely guarded programs. For a program P, let P^{g} be the program P, where for each rule r in P and for each pair of variables X and Y in r, $g(X, Y)$ is added to the body of r. Furthermore, P^{g} contains rules $g(a, b) \leftarrow$ for every $a, b \in cts(P)$, making its size quadratic in the size of P. Note that we assume w.l.o.g. that P does not contain a predicate g.

The newly added guards in the bodies of rules together with the definition of those guards for constants only ensure a correspondence between answer sets and open answer sets where the universe of the latter equals the constants in the program.

Proposition 4. *Let P be a program. M is an answer set of P iff $(cts(P), M \cup \{g(a, b) \mid a, b \in cts(P)\})$ is an open answer set of P^{g}. Moreover, this reduction is quadratic.*

By construction, P^{g} is loosely guarded. We can reduce checking whether there exists an answer set containing a literal to satisfiability checking w.r.t. the open answer set semantics for loosely guarded programs.

Proposition 5. *Let P be a program and q an n-ary predicate in P. There is an answer set M of P with $q(\boldsymbol{a}) \in M$ iff q is satisfiable w.r.t. P^{g}. Moreover, this reduction is quadratic.*

The “only if” direction is trivial; the other direction uses that for every open answer set (U, M') of a loosely guarded program P^{g}, M' contains only terms from $cts(P)$, and can be rewritten as an open answer set $(cts(P), M \cup \{g(a, b) \mid a, b \in cts(P)\})$, after which Proposition 4 becomes applicable.

By [7, 3] and the disjunction-freeness of the GL-reduct of the programs we consider, we have that checking whether there exists an answer set M of P containing a $q(\boldsymbol{a})$ is NEXPTIME-complete. Thus, by Proposition 5, satisfiability checking w.r.t. a LGP is NEXPTIME-hard. In the next section, we improve on this result and show that both satisfiability checking w.r.t. GPs and w.r.t. LGPs is actually 2-EXPTIME-hard.

5 Relationship with Datalog LITE

We define *Datalog* LITE as in [12]. A *Datalog rule* is a rule $\alpha \leftarrow \beta$ where $\alpha = \{a\}$ for some atom a. A *basic Datalog program* is a finite set of Datalog rules such that no head predicate appears in negative bodies of rules. Predicates that appear only in the body of rules are *extensional* or *input* predicates. Note that equality is, by the definition of rules, never a head predicate and thus always extensional. The semantics of a basic Datalog program P, given a relational input structure $\mathcal{U}$ defined over extensional predicates of P[6], is given by its *fixed point model*, see e.g. [1]; for a query (P, q), where P is a basic Datalog program and q is a n-ary predicate, we write $\boldsymbol{a} \in (P, q)(\mathcal{U})$ if there is a fixed point model M of P with input $\mathcal{U}$ such that $q(\boldsymbol{a}) \in M$. We call (P, q) satisfiable if there exists a $\mathcal{U}$ and an $\boldsymbol{a}$ such that $\boldsymbol{a} \in (P, q)(\mathcal{U})$.

[6] We assume that, if $\mathcal{U}$ defines equality, it does so as the identity relation on, at least, the terms in the regular atoms of $\mathcal{U}$ and on the constants in P. Moreover, $\mathcal{U}$ may define equality even if no (in)equality is present in P; one can thus introduce arbitrary universes.

A program P is a *stratified Datalog program* if it can be written as a union of basic Datalog programs $(P_0, \ldots, P_n)$, so-called *strata*, such that each of the head predicates in P is a head predicate in exactly one stratum P_i. Furthermore, if a head predicate in P_i is an extensional predicate in P_j, then $i < j$. This definition entails that head predicates in the positive body of rules are head predicates in the same or a lower stratum, and head predicates in the negative body are head predicates in a lower stratum. The semantics of stratified Datalog programs is defined stratum per stratum, starting from the lowest stratum and defining the extensional predicates on the way up.

A *generalized literal* is of the form $\forall Y_1, \ldots, Y_n \cdot a \Rightarrow b$ where a and b are atoms and $vars(b) \subseteq vars(a)$. A *Datalog* LITE program is a stratified Datalog program, possibly containing generalized literals in the positive body, where each rule is *monadic* or *guarded*. A rule is monadic if each of its (generalized) literals contains only one (free) variable; it is guarded if there exists an atom in the positive body that contains all variables (free variables in the case of generalized literals) of the rule. The definition of stratified is adapted for generalized literals: for a $\forall Y_1, \ldots, Y_n \cdot a \Rightarrow b$ in the body of a rule where the underlying predicate of a is a head predicate, this head predicate must be a head predicate in a lower stratum (i.e. a is treated as a naf-atom) and a head predicate underlying b must be in the same or a lower stratum (i.e. b is treated as an atom). The semantics can be adapted accordingly since a is completely defined in a lower stratum.

In [12], Theorem 8.5., a Datalog LITE query (π_φ, q_φ) was defined for an alternation-free[7] μGF sentence[8] φ such that $(U, M) \models \varphi$ iff $(\pi_\varphi, q_\varphi)(M \cup id(U))$ evaluates to true, where the latter means that q_φ is in the fixed point model of π_φ with input $M \cup id(U)$, and where $id(U) \equiv \{x = x \mid x \in U\}$. For the formal details of this reduction, we refer to [12].

Satisfiability checking with GPs can be polynomially reduced to satisfiability checking in Datalog LITE. Indeed, by Theorem 2, q is satisfiable w.r.t. a GP P iff $p(\boldsymbol{X}, \mathbf{0}, q) \wedge \texttt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable, and the latter is satisfiable iff $\varphi \equiv \exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \texttt{gcomp}((P^{\mathrm{f}})_p)$ is. Since φ is a μGF sentence, we have that φ is satisfiable iff (π_φ, q_φ) is satisfiable. By Theorem 2, the translation of P to φ is quadratic in the size of P and the query (π_φ, q_φ) is quadratic in φ[12], resulting in a polynomial reduction.

Theorem 4. *Let P be a GP, q an n-ary predicate in P and φ the μGF sentence $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \texttt{gcomp}((P^{\mathrm{f}})_p)$. q is satisfiable w.r.t. P iff (π_φ, q_φ) is satisfiable. Moreover, this reduction is polynomial.*

Satisfiability checking in stratified Datalog under the fixed point model semantics can be linearly reduced to satisfiability checking w.r.t. programs under the open answer set semantics. For a stratified Datalog program P, let P^{o} be the program P with free rules $f(\boldsymbol{X}) \vee not\ f(\boldsymbol{X}) \leftarrow$ added for all predicates f that are extensional in the entire program P (with the exception of equality predicates). The free rules in P^{o} mimic the role of extensional predicates from the original P: they allow for an initial free choice of the relational input structure.

[7] Since we did not allow nested least fixed point formulas in our definition of μ(L)GF, it is trivially alternation-free.

[8] A sentence is a formula without free variables.

Proposition 6. *Let P be a stratified Datalog query (P, q). (P, q) is satisfiable iff q is satisfiable w.r.t. P^{o}. Moreover, this reduction is linear.*

Recursion-free stratified Datalog is stratified Datalog where the head predicates in the positive bodies of rules must be head predicates in a lower stratum. We call recursion-free Datalog LITE where all rules are guarded, i.e. without monadic rules that are not guarded, Datalog LITER, where the definition of recursion-free is appropriately extended to take into account the generalized literals.

For a Datalog LITER program P, let $\neg\neg P$ be the program P where all generalized literals are replaced by a double negation. E.g. $q(X) \leftarrow f(X), \forall Y \cdot r(X, Y) \Rightarrow s(Y)$ is rewritten as the rules $q(X) \leftarrow f(X), not\ q'(X)$ and $q'(X) \leftarrow r(X, Y), not\ s(Y)$. As indicated in [12], $\neg\neg P$ is equivalent to P and the recursion-freeness ensures that $\neg\neg P$ is stratified. Clearly, $(\neg\neg P)^{\mathrm{o}}$ is a GP.

For a Datalog LITER query (P, q), $(\neg\neg P, q)$ is an equivalent stratified Datalog query. Hence, by Proposition 6, $(\neg\neg P, q)$ is satisfiable iff q is satisfiable w.r.t. $(\neg\neg P)^{\mathrm{o}}$. This reduction is linear since $\neg\neg P$ is linear in the size of P and so is $(\neg\neg P)^{\mathrm{o}}$. Thus satisfiability checking of Datalog LITER queries can be linearly reduced to satisfiability checking w.r.t. GPs.

Theorem 5. *Let (P, q) be a Datalog* LITER *query. (P, q) is satisfiable iff q is satisfiable w.r.t. $(\neg\neg P)^{\mathrm{o}}$. Moreover, this reduction is linear.*

The reduction from μGF sentences φ to Datalog LITE queries (π_φ, q_φ) specializes, as noted in [12], to a reduction from GF sentences to recursion-free Datalog LITE queries. Moreover, the reduction contains only guarded rules such that GF sentences φ are actually translated to Datalog LITER queries (π_φ, q_φ).

Satisfiability checking in the guarded fragment GF is 2-EXPTIME-complete[13], such that, using Theorem 5 and the intermediate Datalog LITER translation, we have that satisfiability checking w.r.t. GPs is 2-EXPTIME-hard. Completeness readily follows from the 2-EXPTIME membership in Theorem 3.

Every GP is a LGP and satisfiability checking w.r.t. to the former is 2-EXPTIME-complete, thus satisfiability checking w.r.t. LGPs is 2-EXPTIME-hard. Completeness follows again from Theorem 3.

Theorem 6. *Satisfiability checking w.r.t. (L)GPs is 2-*EXPTIME*-complete.*

6 Conclusions and Directions for Further Research

We embedded OASP in FPL and used this embedding to identify (loosely) guarded OASP, a decidable fragment of OASP. Finite ASP was reduced to loosely guarded OASP and the relationship with Datalog LITE was made explicit. Finally, satisfiability checking w.r.t. (loosely) guarded OASP was shown to be 2-EXPTIME-complete.

We plan to further exploit the correspondence between (loosely) guarded OASP and μ(L)GF by seeking to apply implementation techniques used for μ(L)GF satisfiability checking directly to (loosely) guarded OASP. Possibly, we can take advantage of the fact that the embedding does not seem to need the full power of μ(L)GF – there are, e.g. , no nested fixed point formulas in the FPL translation of OASP. It is interesting to search for fragments of guarded OASP that can be implemented using existing answer set solvers such as DLV[9] or SMODELS[23]. Another promising direction is to study generalized literals in the context of the answer set semantics: what is an appropriate semantics in the absence of stratification, can this still be embedded in FPL?

Finally, ω-restricted programs[22] are programs where function symbols are allowed but reasoning is kept decidable by “guarding” variables in a rule with a predicate that is in a lower stratification than the predicate of the head of that rule. Since reasoning with ω-restricted programs is 2-NEXPTIME-complete, it should be possible to simulate guarded open answer set programming in this framework.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. H. Andréka, I. Németi, and J. Van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. of Philosophical Logic*, 27(3):217–274, 1998.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
4. J. Van Benthem. Dynamic Bits and Pieces. In *ILLC research report*. University of Amsterdam, 1997.
5. A. K. Chandra and D. Harel. Horn Clauses and the Fixpoint Query Hierarchy. In *Proc. of PODS ’82*, pages 158–163. ACM Press, 1982.
6. K. L. Clark. Negation as Failure. In *Readings in Nonmonotonic Reasoning*, pages 311–325. Kaufmann, 1987.
7. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
8. E. A. Emerson and E. M. Clarke. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Sciene of Computer Programming*, 2(3):241–266, 1982.
9. W. Faber, N. Leone, and G. Pfeifer. Pushing Goal Derivation in DLP Computations. In *Proc. of LPNMR*, volume 1730 of *LNCS*, pages 177–191. Springer, 1999.
10. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP’88*, pages 1070–1080, Cambridge, Massachusetts, 1988. MIT Press.
11. M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993.
12. G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3(1):1–35, 2002.
13. E. Grädel. On the Restraining Power of Guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.
14. E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *Proc. of LICS ’99*, pages 45–54. IEEE Computer Society, 1999.
15. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Guarded Open Answer Set Programming. Technical report. http://tinf2.vub.ac.be/˜sheymans/tech/guarded-oasp.ps.gz.
16. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Semantic Web Reasoning with Conceptual Logic Programs. In *Proc. of RuleML 2004*, pages 113–127. Springer, 2004.
17. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs. In *Proc. of ESWC 2005*, number 3532 in LNCS, pages 392–407. Springer, 2005.
18. D. Kozen. Results on the Propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
19. J. Lee and V. Lifschitz. Loop Formulas for Disjunctive Logic Programs. In *Proc. of ICLP 2003*, volume 2916 of *LNCS*, pages 451–465. Springer, 2003.

20. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
21. F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. of 18th National Conference on Artificial Intelligence*, pages 112–117. AAAI, 2002.
22. T. Syrjänen. Omega-restricted Logic Programs. In *Proc. of LPNMR*, volume 2173 of *LNAI*, pages 267–279. Springer, 2001.
23. T. Syrjänen and I. Niemelä. The SMODELS System. In *Proc. of LPNMR*, volume 2173 of *LNCS*, pages 434–438. Springer, 2001.