Guarded Open Answer Set Programming with Generalized Literals

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. We extend the open answer set semantics for programs with generalized literals. Such *extended programs (EPs)* have interesting properties, e.g. the ability to express infinity axioms - EPs that have but infinite answer sets. However, reasoning under the open answer set semantics, in particular satisfiability checking of a predicate w.r.t. a program, is already undecidable for programs without generalized literals. In order to regain decidability, we restrict the syntax of EPs such that both rules and generalized literals are *guarded*. Via a translation to guarded fixed point logic (μGF), in which satisfiability checking is 2-EXPTIME-complete, we deduce 2-EXPTIME-completeness of satisfiability checking in such *guarded EPs* (GEPs). *Bound GEPs* are restricted GEPs with EXPTIME-complete satisfiability checking, but still sufficiently expressive to optimally simulate *computation tree logic* (CTL). We translate Datalog LITE programs to GEPs, establishing equivalence of GEPs under an open answer set semantics, alternation-free μGF, and Datalog LITE. Finally, we discuss ω-restricted logic programs under an open answer set semantics.

1 Introduction

In closed answer set programming (ASP) [6], a program consisting of a rule $p(X) \leftarrow not\ q(X)$ and a fact $q(a)$ is grounded with the program's constant a, yielding $p(a) \leftarrow not\ q(a)$ and $q(a)$. This program has one answer set $\{q(a)\}$ such that one concludes that the predicate p is not satisfiable, i.e. there is no answer set of the program that contains a literal with predicate p. Adding more constants to the program could make p satisfiable, e.g., in the absence of a deducible $q(b)$, one has $p(b)$. However, in the context of conceptual modeling, such as designing database schema constraints, this implicit dependence on constants in the program in order to reach sensible conclusions, i.e. the closedness of reasoning, is infeasible. One wants to be able to test satisfiability of a predicate p in a schema independent of any associated data.

For answer set programming, this problem was solved in [7], where k-belief sets are the answer sets of a program that is extended with k extra constants. We extended this idea, e.g. in [13], by allowing for arbitrary, thus possibly infinite, universes. *Open answer sets* are 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.

* Supported by the FWO.

J. Dix and S.J. Hegner (Eds.): FoIKS 2006, LNCS 3861, pp. 179–200, 2006.

In this paper, we extend programs with generalized literals, resulting in *extended programs (EPs)*. A generalized literal is a first-order formula of the form $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$ where $\boldsymbol{Y}$ is a sequence of variables, ϕ is a finite boolean formula and ψ is an atom. Intuitively, such a generalized literal is true in an open interpretation (U, M) if for all substitutions $[\boldsymbol{Y} \mid \boldsymbol{y}]$, $\boldsymbol{y}$ in U, such that $\phi[\boldsymbol{Y} \mid \boldsymbol{y}]$ is true in M, $\psi[\boldsymbol{Y} \mid \boldsymbol{y}]$ is true in M.

Generalized literals $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$, with ϕ an atom instead of a boolean formula, were introduced in Datalog[1] with the language Datalog LITE [8]: stratified Datalog with generalized literals, where rules are monadic or guarded, and under an appropriate extension of the least fixed point semantics. In open answer set programming (OASP), we define a reduct that removes the generalized literals. E.g., a rule $r : ok \leftarrow \forall X \cdot critical(X) \Rightarrow work(X)$ expresses that a system is OK if all critical devices are functioning: the *GLi-reduct (generalized literal reduct)* of such a rule for an open interpretation $(\{x_0, \ldots\}, M)$ where M contains $critical(x_i)$ for even i, contains a rule $r' : ok \leftarrow work(x_0), work(x_2), \ldots$, indicating that the system is OK if the critical devices x_0, $x_2, \ldots$ are working. The GLi-reduct does not contain generalized literals and one can apply the normal answer set semantics, modified to take into account the infinite body.

Just like it is not feasible to introduce all relevant constants in a program to ensure correct conceptual reasoning, it is not feasible, not even possible, to write knowledge directly as in r' for it has an infinite body. Furthermore, even in the presence of a finite universe, generalized literals allow for a more robust representation of knowledge than would be possible without them. E.g., with critical devices y_1 and y_2, a rule $s : ok \leftarrow work(y_1), work(y_2)$ does the job as good as r (and in fact s is the GLi-reduct of r), but adding new critical devices, implies revisiting s and replacing it by a rule that reflects the updated situation. Not only is this cumbersome, it may well be impossible as s contains no explicit reference to critical devices, and the knowledge engineer may not have a clue as to which rules to modify.

Characteristic about (O)ASP is its treatment of negation as failure (naf): one guesses an interpretation for a program, removes naf by computing the GL-reduct, calculates the iterated fixed point of this reduct, and checks whether this fixed point equals the initial interpretation. In [14], these external manipulations, i.e. not expressible in the language of programs itself, were compiled into fixed point logic (FPL) [11], i.e. into an extension of first-order logic with fixed point formulas. We will show how to modify the FPL translation to take into account generalized literals.

Satisfiability checking w.r.t. arbitrary EPs, even without generalized literals, under the open answer set semantics is undecidable (e.g. the domino problem can be reduced to it), and satisfiability checking in FPL is as well, as it is an extension of the undecidable first-order logic. Thus, with the FPL translation, we have a mapping from one undecidable framework into another undecidable framework. This is interesting in its own right, as it provides a characterization of an answer set semantics in FPL. But more interesting, is the deployment of the translation in order to identify decidable subclasses of EPs: if the FPL translation of a class of EPs falls into a decidable fragment of FPL, this class of EPs is decidable.

[1] The extension of logic programming syntax with first-order formulas dates back to [17].

Guarded fixed point logic (μGF) [11] is such a decidable fragment of FPL that is able to express fixed point formulas. It restricts the use of quantified variables by demanding that they are guarded by an atom. We restrict EPs, resulting in *guarded EPs (GEPs)*, such that all variables in a rule appear in an atom in the positive body and all generalized literals are guarded, where a generalized literal is guarded, basically, if it can be written as a guarded formula in μGF. The FPL translation of GEPs then falls into the μGF fragment, yielding a 2-EXPTIME upper complexity bound for satisfiability checking. Together with the 2-EXPTIME-completeness of guarded programs without generalized literals from [14], this establishes 2-EXPTIME-completeness for satisfiability checking w.r.t. GEPs. As a consequence, adding generalized literals to a guarded program does not increase the complexity of reasoning. We further illustrate the expressiveness of (bound) GEPs by simulating reasoning in *computational tree logic (CTL)* [4], a logic for expressing temporal knowledge.

Finally, we reduce Datalog LITE reasoning, without monadic rules, to reasoning with GEPs. In particular, we prove a generalization of the well-known result from [6] that the unique answer set of a stratified program coincides with its least fixed point model: for a universe U, the unique open answer set (U, M) of a stratified Datalog program with generalized literals is identical[2] to its least fixed point model with input structure $id(U)$, the identity relation on U. Furthermore, the Datalog LITE simulation, together with the reduction of GEPs to alternation-free[3] μGF, as well as the equivalence of alternation-free μGF and Datalog LITE [8], lead to the conclusion that alternation-free μGF, Datalog LITE, and OASP with GEPs, are equivalent, i.e. their satisfiability checking problems can be polynomially reduced to one another.

GEPs are just as expressive as Datalog LITE, however, from a knowledge representation viewpoint, GEPs allow for a compact expression of circular knowledge. E.g., the omni-present construction with rules $a(X) \leftarrow not\ b(X)$ and $b(X) \leftarrow not\ a(X)$ is not stratified and cannot be (directly) expressed in Datalog LITE. The reduction to Datalog LITE does indicate that negation as failure under the (open) answer set semantics is not that special regarding expressiveness, but can be regarded as convenient semantic sugar.

The remainder of the paper is organized as follows. After extending the open answer set semantics to support generalized literals in Section 2, we give the FPL translation in Section 3. Section 4 defines GEPs, proves a 2-EXPTIME complexity upper bound for satisfiability checking, and concludes with a CTL simulation. Section 5 describes a simulation of Datalog LITE, without monadic rules, yielding equivalence of alternation-free μGF, Datalog LITE, and GEPs. Section 6 describes the relationship with ω-restricted programs. Section 7 contains conclusions and directions for further research. Due to space restrictions, proofs and further related work have been omitted; the former can be found in http://tinf2.vub.ac.be/ sheymans/tech/goasp-gl.ps.gz, for the latter we refer to [14] and the references therein.

2 Open Answer Set Programming with Generalized Literals

A *term* t is a constant or a variable, where the former is denoted with $a, b, \ldots$ and the latter with $X, Y, \ldots$ A *k-ary atom* is of the form $p(\boldsymbol{t})$ for a sequence of terms $\boldsymbol{t} =$

[2] Modulo equality atoms, which are implicit in OASP, but explicit in Datalog LITE.

[3] μGF without nested fixed point variables in alternating least and greatest fixed point formulas.

$t_1, \ldots, t_k$, $0 \leq k < \omega$[4], and a k-ary predicate symbol p. A *literal* is an atom $p(\boldsymbol{t})$ or a *naf-atom* $not\ p(\boldsymbol{t})$ for an atom $p(\boldsymbol{t})$.[5] The *positive part* of a set of literals α is $\alpha^+ = \{p(\boldsymbol{t}) \mid p(\boldsymbol{t}) \in \alpha\}$ and the *negative part* of α is $\alpha^- = \{p(\boldsymbol{t}) \mid not\ p(\boldsymbol{t}) \in \alpha\}$, i.e. the positive part of a set of literals are the atoms, the negative part are the naf-atoms without the *not* symbol. 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 *generalized literal* is a first-order formula of the form $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$, where ϕ is a finite boolean formula of atoms (i.e. using $\neg$, $\vee$, and $\wedge$) and ψ is an atom; we call ϕ the *antecedent* and ψ the *consequent*. We refer to literals and generalized literals as *extended literals*. For a set of extended literals α, $\alpha^{\mathrm{x}} \equiv \{l \mid l \text{ generalized literal in } \alpha\}$, the set of generalized literals in α. We extend α^+ and α^- for extended literals as follows: $\alpha^+ = (\alpha \backslash \alpha^{\mathrm{x}})^+$ and $\alpha^- = (\alpha \backslash \alpha^{\mathrm{x}})^-$; thus $\alpha = \alpha^+ \cup not\ \alpha^- \cup \alpha^{\mathrm{x}}$.

An *extended program (EP)* is a countable set of *rules* $\alpha \leftarrow \beta$, where α is a finite set of literals, $|\alpha^+| \leq 1$, β is a countable[6] set of extended literals, 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. The set α is the *head* of the rule and represents a disjunction[7] of literals, while β is called the *body* and represents a conjunction of extended literals. If $\alpha = \emptyset$, the rule is called a *constraint*. *Free rules* are rules of the form $q(\boldsymbol{t}) \vee not\ q(\boldsymbol{t}) \leftarrow$ 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 $p(\boldsymbol{t}) \vee not\ p(\boldsymbol{t}) \leftarrow$. Literals are *ground* if they do not contain variables, generalized literals are ground if they do not contain free variables, and rules and EPs are ground if all extended literals in it are ground.

For an EP P, let $cts(P)$ be the constants in P, and $preds(P)$ its predicates. For a (generalized) literal l, we define $vars(l)$ as the (free) variables in l. For a rule r, we define $vars(r) \equiv \cup\{vars(l) \mid l \text{ extended literal in } r\}$. Let $\mathcal{B}_P$ be the set of regular ground atoms that can be formed from an EP P. An *interpretation* I of P is then 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 $I \not\models p(\boldsymbol{t})$. We further extend this, by induction, for any boolean formula of ground atoms. For such ground boolean formulas ϕ and ψ, we have $I \models \phi \wedge \psi$ iff $I \models \phi$ and $I \models \psi$, $I \models \phi \vee \psi$ iff $I \models \phi$ or $I \models \psi$, and $I \models \neg\phi$ iff $I \not\models \phi$. For a set of ground literals X, we have $I \models X$ iff $I \models x$ for every $x \in X$. A ground rule $r : \alpha \leftarrow \beta$, not containing generalized literals, 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

[4] We thus allow for 0-ary predicates, i.e. propositions.

[5] We have no classical negation $\neg$, however, programs with $\neg$ can be reduced to programs without it, see e.g. [16]. To be precise, we should then refer to stable models instead of answer sets.

[6] Thus the rules may have an infinite body.

[7] The condition $|\alpha^+| \leq 1$ ensures that the GL-reduct is non-disjunctive. This allows for the definition of an immediate consequence operator, on which we rely in our proofs to make the correspondence with FPL. In the presence of positive disjunction, the currently defined operator does not suffice and it is not clear whether this can be fixed (and how).

not and without generalized literals, 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 I is subset minimal, i.e. there is no model I' of P with $I' \subset I$. For ground EPs P containing *not* but still without generalized literals, the *GL-reduct* [6] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P if $I \models not\ \beta^-$ and $I \models \alpha^-$. I is an *answer set* of a ground P without generalized literals if I is an answer set of P^I.

Example 1. Take the program P with rules $p(a) \leftarrow not\ q(a)$ and $q(a) \leftarrow not\ p(a)$. Then P has 4 interpretations $\emptyset$, $\{p(a)\}$, $\{q(a)\}$, and $\{p(a), q(a)\}$. The GL-reduct of P w.r.t. $\emptyset$ is $\{p(a) \leftarrow; q(a) \leftarrow\}$ which has $\{p(a), q(a)\}$ as its minimal model, and thus $\emptyset$ is not an answer set. The GL-reduct of P w.r.t. $\{p(a), q(a)\}$ is $\emptyset$ which has $\emptyset$ as its minimal model, and thus $\{p(a), q(a)\}$ is not an answer set. The GL-reduct of P w.r.t. $\{p(a)\}$ is $\{p(a) \leftarrow\}$ which has $\{p(a)\}$ as its minimal model, making $\{p(a)\}$ an answer set. Similarly, one can deduce that $\{q(a)\}$ is an answer set.

A *universe* U for an EP P is a non-empty countable superset of the constants in P: $cts(P) \subseteq U$. Let $\mathcal{B}_P^U$ be the set of regular ground atoms that can be formed from an EP P and the terms in a universe U for P. An *open interpretation* of an EP P is a pair (U, I) where U is a universe for P and I is any subset of $\mathcal{B}_P^U$.

For ground EPs P the *GLi-reduct* $P^{\mathbf{X}(U,I)}$ w.r.t. an open interpretation (U, I) removes the generalized literals from the program: $P^{\mathbf{X}(U,I)}$ contains the rules[8]

$$\alpha \leftarrow \beta \backslash \beta^{\mathbf{X}}, \bigcup_{\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi \in \beta^{\mathbf{X}}} \{\psi[\boldsymbol{Y}|\boldsymbol{y}] \mid \boldsymbol{y} \subseteq U, I \models \phi[\boldsymbol{Y}|\boldsymbol{y}]\}\ , \tag{1}$$

for $\alpha \leftarrow \beta$ in P. Intuitively, a generalized literal $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$ is replaced by those $\psi[\boldsymbol{Y}|\boldsymbol{y}]$ for which $\phi[\boldsymbol{Y}|\boldsymbol{y}]$ is true, such that[9], e.g., $p(a) \leftarrow [\forall \boldsymbol{X} \cdot q(X) \Rightarrow r(X)]$ means that in order to deduce $p(a)$ one needs to deduce $r(x)$ for all x where $q(x)$ holds. If only $q(x_1)$ and $q(x_2)$ hold, then the GLi-reduct contains $p(a) \leftarrow r(x_1), r(x_2)$. With an infinite universe and a condition ϕ that holds for an infinite number of elements in the universe, one can thus have a rule with an infinite body in the GLi-reduct. An open interpretation (U, I) is an *open answer set* of a ground P if I is an answer set of $P^{\mathbf{X}(U,I)}$.

We call P_U the ground EP obtained from an EP P by substituting every (free) variable in a rule in P by every element in U. In the following, an EP is assumed to be a finite set of rules; infinite EPs only appear as byproducts of grounding a finite program with an infinite universe, or, by taking the GLi-reduct w.r.t. an infinite universe. An *open answer set* of P is an open interpretation (U, M) of P with (U, M) an open 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, basically for technical reasons (see Example 4), 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 linearly reduced to satisfiability checking of a new non-free n-ary predicate p' w.r.t. $P \cup \{p'(\boldsymbol{X}) \leftarrow p(\boldsymbol{X})\}$.

[8] We denote the substitution of $\boldsymbol{Y} = Y_1, \ldots, Y_d$ with $\boldsymbol{y} = y_1, \ldots, y_d$ in an expression (be it an atom, set of atoms, boolean formula, or rule) X as $X[\boldsymbol{Y}|\boldsymbol{y}]$. If the substitution is clear from the context we write $X[]$.

[9] We put square brackets around generalized literals for clarity.

Example 2. Take an EP P

$$p(X) \leftarrow [\forall Y \cdot q(Y) \Rightarrow r(Y)] \qquad r(X) \leftarrow q(X)$$
$$q(X) \vee not\ q(X) \leftarrow$$

and an open interpretation $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$. Intuitively, the first rule says that $p(X)$ holds if for every Y where $q(Y)$ holds, $r(Y)$ holds (thus $p(X)$ also holds if $q(Y)$ does not hold for any Y). The GLi-reduct of $P_{\{x,y\}}$ is

$$\begin{array}{ll} p(x) \leftarrow r(x) & p(y) \leftarrow r(x) \\ r(x) \leftarrow q(x) & r(y) \leftarrow q(y) \\ q(x) \leftarrow & \end{array}$$

which has exactly $\{p(x), r(x), q(x), p(y)\}$ as its minimal model such that the open interpretation $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$ is indeed an open answer set.

There are EPs, not containing (in)equality atoms, for which predicates are only satisfiable by infinite open answer sets.

Example 3. Take the program P, the open answer set variant of the classical infinity axiom in guarded fixed point logic from [11]:

$$\begin{array}{ll} r_1: & q(X) \leftarrow f(X, Y) \\ r_2: & \leftarrow f(X, Y), not\ q(Y) \\ r_3: & \leftarrow f(X, Y), not\ well(Y) \\ r_4: & well(Y) \leftarrow q(Y), [\forall X \cdot f(X, Y) \Rightarrow well(X)] \\ r_5: & f(X, Y) \vee not\ f(X, Y) \leftarrow \end{array}$$

In order to satisfy q with some x, one needs to apply r_1, which enforces an f-successor y. The second rule ensures that also for this y an f-successor must exist, etc. The third rule makes sure that every f-successor is on a well-founded f-chain. The well-foundedness itself is defined by r_4 which says that y is on a well-founded chain of elements where q holds if all f-predecessors of y satisfy the same property.

For example, take an infinite open answer set (U, M) with $U = \{x_0, x_1, \ldots\}$ and $M = \{q(x_0), well(x_0), f(x_0, x_1), q(x_1), well(x_1), f(x_1, x_2), \ldots\}$. P_U contains the following grounding of r_4:

$$\begin{array}{l} r_4^0 : well(x_0) \leftarrow q(x_0), [\forall X \cdot f(X, x_0) \Rightarrow well(X)] \\ r_4^1 : well(x_1) \leftarrow q(x_1), [\forall X \cdot f(X, x_1) \Rightarrow well(X)] \\ \quad \vdots \end{array}$$

Since, for r_4^0, there is no $f(y, x_0)$ in M, the body of the corresponding rule in the GLi-reduct w.r.t. (U, M) contains only $q(x_0)$. For r_4^1, we have that $f(x_0, x_1) \in M$ such that we include $well(x_0)$ in the body:

$$\begin{array}{l} well(x_0) \leftarrow q(x_0) \\ well(x_1) \leftarrow q(x_1), well(x_0) \\ \quad \vdots \end{array}$$

Thus, (U, M) is an open answer set of the EP, satisfying q.

Moreover, no finite open answer set can satisfy q. First, note that an open answer set (U, M) of P cannot contain loops, i.e. $\{f(x_0, x_1), \ldots, f(x_n, x_0)\} \subseteq M$ is not possible. Assume the contrary. By rule r_3, we need $well(x_0) \in M$. However, the GLi-reduct of P_U contains rules:

$$\begin{aligned} well(x_0) &\leftarrow q(x_0), well(x_n), \ldots \\ well(x_n) &\leftarrow q(x_n), well(x_{n-1}), \ldots \\ &\vdots \\ well(x_1) &\leftarrow q(x_1), well(x_0), \ldots \end{aligned}$$

such that $well(x_0)$ cannot be in any open answer set: we have a circular dependency and cannot use these rules to motivate $well(x_0)$, i.e. $well(x_0)$ is unfounded. Thus an open answer set cannot contain loops.

Assume that q is satisfied in an open answer set (U, M) with $q(x_0) \in M$. Then, by rule r_1, we need some X such that $f(x_0, X) \in M$. Since M cannot contain loops X must be different from x_0 and we need some new x_1. By rule r_2, $q(x_1) \in M$, such that by rule r_1, we again need an X such that $f(x_1, X)$. Using x_0 or x_1 for X results in a loop, such that we need a new x_2. This process continues infinitely, such that there are only infinite open answer sets that make q satisfiable w.r.t. P.

3 Open Answer Set Programming with EPs Via Fixed Point Logic

We assume first-order logic (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.

We define *Fixed Point Logic (FPL)* along the lines of [11], i.e. as an extension of first-order logic, where formulas may additionally be *fixed point formulas* of the form

$$[\text{LFP } W\boldsymbol{X}.\psi(W, \boldsymbol{X})](\boldsymbol{X}) \quad \text{or} \quad [\text{GFP } W\boldsymbol{X}.\psi(W, \boldsymbol{X})](\boldsymbol{X}) \ , \tag{2}$$

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

For an interpretation (U, M) and a valuation χ of the free predicate variables, except W, in ψ, we define the operator $\psi^{(U,M),\chi} : 2^{U^n} \to 2^{U^n}$ on sets S of n-ary tuples

$$\psi^{(U,M),\chi}(S) \equiv \{\boldsymbol{x} \in U^n \mid (U, M), \chi \cup \{W \to S\} \models \psi(W, \boldsymbol{x})\} \ , \tag{3}$$

where $\chi \cup \{W \to S\}$ is the valuation χ extended such that W is assigned to S. If $\psi(W, \boldsymbol{X})$ contains only the predicate variable W, we often omit the valuation χ and write just $\psi^{(U,M)}$. By definition, W appears only positively in ψ such that $\psi^{(U,M),\chi}$ is monotonic on sets of n-ary U-tuples and thus has a least and greatest fixed point, which we denote by $\text{LFP}(\psi^{(U,M),\chi})$ and $\text{GFP}(\psi^{(U,M),\chi})$ respectively. Finally, we have that

$$(U, M), \chi \models [\text{LFP } W\boldsymbol{X}.\psi(W, \boldsymbol{X})](\boldsymbol{x}) \iff \boldsymbol{x} \in \text{LFP}(\psi^{(U,M),\chi}) \ , \tag{4}$$

and similarly for greatest fixed point formulas. As in [8], we call an FPL *sentence* (i.e. an FPL formula without free variables) *alternation-free* if it does not contain subformulas $\psi \equiv [\text{LFP } T\boldsymbol{X}.\varphi](\boldsymbol{X})$ and $\theta \equiv [\text{GFP } S\boldsymbol{Y}.\eta](\boldsymbol{Y})$ such that T occurs in η and θ is a subformula of φ, or S occurs in φ and ψ is a subformula of η. We can eliminate greatest fixed point formulas from a formula, by the equivalence: $[\text{GFP } W\boldsymbol{X}.\psi] \equiv \neg[\text{LFP } W\boldsymbol{X}.\neg\psi[W|\neg W]]$, where $\neg\psi[W|\neg W]$ is $\neg\psi$ with W replaced by $\neg W$. If we thus remove greatest fixed point predicates, and if negations appear only in front of atoms or least fixed point formulas, then a formula is alternation-free iff no fixed point variable W appears in the scope of a negation.

First, we rewrite an arbitrary EP as an EP 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. We assume without loss of generality that the set of constants and the set of predicates in an EP are disjoint and that each predicate q has one associated arity, e.g. $q(x)$ and $q(x, y)$ are not allowed. An EP P is a *p-EP* if p is the only predicate in P different from the (in)equality predicate. In [14], we showed how to rewrite any program P (without generalized literals) as an equivalent p-program P_p. We adapt that transformation to cope with generalized literals as well. For an EP P, let $in(Y) \equiv \cup\{Y \neq a \mid a \in preds(P) \cup \{0\}\}$, i.e. a set of inequalities between the variable Y and the predicates in P as well as a new constant 0. For a sequence of variables $\boldsymbol{Y}$, we have $in(\boldsymbol{Y}) \equiv \cup_{Y \in \boldsymbol{Y}} in(Y)$. For a set of extended literals α, we construct α_p in two stages:

1. replace every regular m-ary atom $q(\boldsymbol{t})$ appearing in α (either in atoms, naf-atoms, or generalized literals) by $p(\boldsymbol{t}, \boldsymbol{0}, q)$ where p has arity n, with n the maximum of the arities of predicates in P augmented by 1, $\boldsymbol{0}$ a sequence of new constants 0 of length $n-m-1$, and q a new constant with the same name as the original predicate,
2. in the set thus obtained, replace every generalized literal $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$ by $\forall \boldsymbol{Y} \cdot \phi \wedge \bigwedge in(\boldsymbol{Y}) \Rightarrow \psi$, where $Y \neq t$ in $in(\boldsymbol{Y})$ stands for $\neg(Y = t)$ (we defined generalized literals in function of boolean formulas of atoms).

The p-EP P_p is the program P with all non-free rules $r : \alpha \leftarrow \beta$ replaced by $r_p : \alpha_p \leftarrow \beta_p, in(\boldsymbol{X})$ where $vars(r) = \boldsymbol{X}$. Note that P and P_p have the same free rules.

Example 4. Let P be the EP:

$$\begin{aligned} q(X) &\leftarrow [\forall Y \cdot r(Y) \Rightarrow f(X, Y)] \\ r(a) &\leftarrow \\ f(X, Y) \vee not\ f(X, Y) &\leftarrow \end{aligned}$$

Then q is satisfiable $(\{a, x\}, \{f(x, a), r(a), q(x)\})$. The p-EP P_p is

$$\begin{aligned} p(X, 0, q) &\leftarrow [\forall Y \cdot p(Y, 0, r) \wedge \bigwedge in(Y) \Rightarrow p(X, Y, f)], in(X) \\ p(a, 0, r) &\leftarrow \\ p(X, Y, f) \vee not\ p(X, Y, f) &\leftarrow \end{aligned}$$

where $in(X) = \{X \neq f, X \neq q, X \neq r, X \neq 0\}$. The corresponding open answer set for this program is $(\{a, x, f, r, q, 0\}, \{p(x, a, f), p(a, 0, r), p(x, 0, q)\})$. Note that the free rule in P_p may introduce unwanted literals $p(q, x, f)$, i.e. where X is grounded

with a predicate q from P. Those unwanted literals will, however, never make non-free rules applicable since the latter have $X \neq q$ in the body, and hence the assumption that we only check satisfiability of non-free predicates.

Proposition 1. *Let P be an EP 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-EP P_p with $p(\boldsymbol{x}, \mathbf{0}, q) \in M'$.*

Note that the size of P_p is polynomial in the size of P.

In [3], 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.

As was shown in [14], we can reduce a p-program P (without generalized literals) to an equivalent FPL formula. We extend this translation for EPs, i.e. we take into account generalized literals. The *completion* $\texttt{comp}(P)$ of an EP P consists of formulas that demand that different constants in P are interpreted as different elements:

$$a \neq b \tag{5}$$

for every pair of different constants a and b in P, and where $a \neq b \equiv \neg(a = b)$. $\texttt{comp}(P)$ contains formulas ensuring the existence of at least one element in the domain of an interpretation:

$$\exists X \cdot \mathbf{true} \tag{6}$$

Besides these technical requirements matching FOL interpretations with open interpretations, $\texttt{comp}(P)$ contains the formulas in $\texttt{fix}(P) = \texttt{sat}(P) \cup \texttt{gl}(P) \cup \texttt{gli}(P) \cup \texttt{fpf}(P)$, 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, $\texttt{gli}(P)$ indicates when the antecedent of generalized literals are true, and finally $\texttt{fpf}(P)$ ensures that every model of $\texttt{fix}(P)$ is a minimal model of the GL-reduct of the GLi-reduct of 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 the GLi-reduct of P; the atoms defined in $\texttt{gli}(P)$ ensure that the generalized literals are interpreted correctly.

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, $\bigwedge X = \mathbf{true}$ and $\bigvee X = \mathbf{false}$. In the following, we assume that the arity of p, the only predicate in a p-EP is n.

Definition 1. *Let P be a p-EP. The fixed point translation of P is $\texttt{fix}(P) \equiv \texttt{sat}(P) \cup \texttt{gli}(P) \cup \texttt{gl}(P) \cup \texttt{fpf}(P)$, where*

1. $\texttt{sat}(P)$ *contains formulas*
$$\forall \boldsymbol{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{7}$$
for rules $r : \alpha \leftarrow \beta \in P$ with $vars(r) = \boldsymbol{Y}$,
2. $\texttt{gl}(P)$ *contains the formulas*
$$\forall \boldsymbol{Y} \cdot r(\boldsymbol{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{8}$$
for rules $r : \alpha \leftarrow \beta \in P$[10] with $vars(r) = \boldsymbol{Y}$,

[10] We assume that rules are uniquely named.

3. $\texttt{gli}(P)$ *contains the formulas*

$$\forall \boldsymbol{Z} \cdot g(\boldsymbol{Z}) \Leftrightarrow \phi \tag{9}$$

for generalized literals $g : \forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi \in P$[11] *where* ϕ *contains the variables* $\boldsymbol{Z}$*,*

4. $\texttt{fpf}(P)$ *contains the formula*

$$\forall \boldsymbol{X} \cdot p(\boldsymbol{X}) \Rightarrow [\text{LFP } W\boldsymbol{X}.\phi(W, \boldsymbol{X})](\boldsymbol{X}) \tag{10}$$

with

$$\phi(W, \boldsymbol{X}) \equiv W(\boldsymbol{X}) \vee \bigvee_{r:p(\boldsymbol{t}) \vee \alpha \leftarrow \beta \in P} E(r) \tag{11}$$

and

$$E(r) \equiv \exists \boldsymbol{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p \mid W] \wedge \bigwedge \gamma \wedge r(\boldsymbol{Y}) \tag{12}$$

where $\boldsymbol{X} = X_1, \ldots, X_n$ *are* n *new variables,* $vars(r) = \boldsymbol{Y}$*,* W *is a new (second-order) variable,* $\beta^+[p \mid W]$ *is* β^+ *with* p *replaced by* W*, and* γ *is* β^x *with*

- *every generalized literal* $g : \forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$ *replaced by* $\forall \boldsymbol{Y} \cdot g(\boldsymbol{Z}) \Rightarrow \psi$*,* $\boldsymbol{Z}$ *the variables of* ϕ*, and, subsequently,*
- *every* p *replaced by* W*.*

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

The predicate W appears only positively in $\phi(W, \boldsymbol{X})$ such that the fixed point formula in (10) is well-defined. Note that the predicate p is replaced by the fixed point variable W in $E(r)$ except in the antecedents of generalized literals, which were replaced by g-atoms, and the negative part of r, which were replaced by r-atoms, thus respectively encoding the GLi-reduct and the GL-reduct.

Example 5. We rewrite the program from Example 3 as the p-EP P.

$$\begin{array}{lrl}
r_1: & p(X,0,q) & \leftarrow p(X,Y,f), in(X), in(Y) \\
r_2: & & \leftarrow p(X,Y,f), not\ p(Y,0,q), in(X), in(Y) \\
r_3: & & \leftarrow p(X,Y,f), not\ p(Y,0,well), in(X), in(Y) \\
r_4: & p(Y,0,well) & \leftarrow p(Y,0,q), in(Y), \\
 & & \quad [\forall X \cdot p(X,Y,f) \wedge \textstyle\bigwedge in(X) \Rightarrow p(X,0,well)] \\
r_5: & p(X,Y,f) \vee not\ p(X,Y,f) & \leftarrow
\end{array}$$

where $in(X)$ and $in(Y)$ are shorthand for the inequalities with the new constants. $\texttt{sat}(P)$ consists of the sentences

- $\forall X, Y \cdot p(X,Y,f) \wedge \bigwedge in(X) \wedge \bigwedge in(Y) \Rightarrow p(X,0,q)$,
- $\forall X, Y \cdot p(X,Y,f) \wedge \neg p(Y,0,q) \wedge \bigwedge in(X) \wedge \bigwedge in(Y) \Rightarrow$ **false**,
- $\forall X, Y \cdot p(X,Y,f) \wedge \neg p(Y,0,well) \wedge \bigwedge in(X) \wedge \bigwedge in(Y) \Rightarrow$ **false**, and
- $\forall Y \cdot p(Y,0,q) \wedge \bigwedge in(Y) \wedge (\forall X \cdot p(X,Y,f) \wedge \bigwedge in(X) \Rightarrow p(X,0,well)) \Rightarrow p(Y,0,well)$,
- $\forall X, Y \cdot \textbf{true} \Rightarrow p(X,Y,f) \vee \neg p(X,Y,f)$.

[11] We assume that generalized literals are named.

$\mathtt{gl}(P)$ contains the sentences

- $\forall X, Y \cdot r_1(X, Y) \Leftrightarrow in(X) \wedge in(Y)$,
- $\forall X, Y \cdot r_2(X, Y) \Leftrightarrow \neg p(Y, 0, q) \wedge in(X) \wedge in(Y)$,
- $\forall X, Y \cdot r_3(X, Y) \Leftrightarrow \neg p(Y, 0, well) \wedge in(X) \wedge in(Y)$,
- $\forall Y \cdot r_4(Y) \Leftrightarrow in(Y)$, and
- $\forall X, Y \cdot r_5(X, Y) \Leftrightarrow p(X, Y, f)$.

$\mathtt{gli}(P)$ contains the sentence $\forall X, Y \cdot g(X, Y) \Leftrightarrow p(X, Y, f) \wedge \bigwedge in(X)$, and $\mathtt{fpf}(P)$ is constructed with

- $E(r_1) \equiv \exists X, Y \cdot X_1 = X \wedge X_2 = 0 \wedge X_3 = q \wedge W(X, Y, f) \wedge r_1(X, Y)$,
- $E(r_4) \equiv \exists Y \cdot X_1 = Y \wedge X_2 = 0 \wedge X_3 = well \wedge W(Y, 0, q) \wedge$ $(\forall X \cdot g(X, Y) \Rightarrow W(X, 0, well)) \wedge r_4(Y)$.
- $E(r_5) \equiv \exists X, Y \cdot X_1 = X \wedge X_2 = Y \wedge X_3 = f \wedge r_5(X, Y)$.

Take an infinite FOL interpretation (U, M) with $U = \{q, f, well, 0, x_0, x_1, \ldots\}$ and[12]

$$\begin{aligned} M = \{&p(x_0, 0, q), p(x_0, 0, well), p(x_0, x_1, f), \\ &p(x_1, 0, q), p(x_1, 0, well), p(x_1, x_2, f), \ldots \\ &r_1(x_0, x_0), r_1(x_0, x_1), \ldots, r_1(x_1, x_0), \ldots, r_4(x_0), r_4(x_1), \ldots \\ &r_5(x_0, x_1), r_5(x_1, x_2), \ldots, g(x_0, x_1), g(x_1, x_2), \ldots\}) \ . \end{aligned}$$

$\mathtt{sat}(P)$, $\mathtt{gl}(P)$, and $\mathtt{gli}(P)$ are satisfied. We check that $\mathtt{fpf}(P)$ is satisfied by M. We first construct the fixed point of $\phi^{(U,M)}$ where $\phi(W, X_1, X_2, X_3) \equiv W(X_1, X_2, X_3) \vee E(r_1) \vee E(r_4) \vee E(r_5)$ as in [9], i.e. in stages starting from $W^0 = \emptyset$. We have that

- $W^1 = \phi^{(U,M)}(W^0) = \{(x_0, x_1, f), (x_1, x_2, f), \ldots\}$, where the (x_i, x_{i+1}, f) are introduced by $E(r_5)$,
- $W^2 = \phi^{(U,M)}(W^1) = W_1 \cup \{(x_0, 0, q), (x_1, 0, q), \ldots\}$, where the $(x_i, 0, q)$ are introduced by $E(r_1)$,
- $W^3 = \phi^{(U,M)}(W^2) = W_2 \cup \{(x_0, 0, well)\}$, where $(x_0, 0, well)$ is introduced by $E(r_4)$,
- $W^4 = \phi^{(U,M)}(W^3) = W_3 \cup \{(x_1, 0, well)\}$,
- $\ldots$

The least fixed point $\mathrm{LFP}(\phi^{(U,M)})$ is then $\cup_{\alpha<\omega} W^\alpha$ [9]. The sentence $\mathtt{fpf}(P)$ is then satisfied since every p-literal in M is also in this least fixed point. (U, M) is thus a model of $\mathtt{comp}(P)$, and it corresponds to an open answer set of P.

Proposition 2. *Let P be a p-EP. Then, (U, M) is an open answer set of P iff $(U, M \cup R \cup G)$ is a model of $\mathtt{comp}(P)$, where $R \equiv \{r(\boldsymbol{y}) \mid r[\boldsymbol{Y} \mid \boldsymbol{y}] : \alpha[] \leftarrow \beta[] \in P_U, M \models \alpha[]^- \cup not\ \beta[]^-, vars(r) = \boldsymbol{Y}\}$, i.e. the atoms corresponding to rules for which the GLi-reduct version will be in the GL-reduct, and $G \equiv \{g(\boldsymbol{z}) \mid g : \forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi \in P, vars(\phi) = \boldsymbol{Z}, M \models \phi[\boldsymbol{Z} \mid \boldsymbol{z}]\}$, i.e. the atoms corresponding to true antecedents of generalized literals in P.*

[12] We interpret the constants in $\mathtt{comp}(P)$ by universe elements of the same name.

Using Propositions 1 and 2, we can reduce satisfiability checking in OASP to satisfiability checking in FPL. Moreover, since $\texttt{comp}(P)$ contains only one fixed point predicate, the translation falls in the alternation-free fragment of FPL. If the number of constants in a program P is c, then the number of formulas $a \neq b$ is $\frac{1}{2}c(c-1)$; 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 an EP and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \texttt{comp}(P_p)$ is satisfiable. Moreover, this reduction is polynomial.*

4 Open Answer Set Programming with Guarded Extended Programs

We repeat the definitions of the *guarded fragment* [2] of first-order logic as in [11]: *The guarded fragment GF of first-order logic is defined inductively as follows:*

(1) *Every relational atomic formula belongs to GF.*
(2) *GF is closed under propositional connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$.*
(3) *If $\boldsymbol{X}$, $\boldsymbol{Y}$ are tuples of variables, $\alpha(\boldsymbol{X}, \boldsymbol{Y})$ is an atomic formula, and $\psi(\boldsymbol{X}, \boldsymbol{Y})$ is a formula in GF such that free$(\psi) \subseteq$ free$(\alpha) = \boldsymbol{X} \cup \boldsymbol{Y}$, 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 GF, (where free(ψ) are the free variables of ψ). $\alpha(\boldsymbol{X}, \boldsymbol{Y})$ is the guard *of the formula.*

The *guarded fixed point logic* μGF is GF extended with fixed point formulas (2) where $\psi(W, \boldsymbol{X})$ is a formula such that W does not appear in guards.

Definition 2. *A generalized literal $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$ is* guarded *if ϕ is of the form $\gamma \wedge \phi'$ with γ an atom, and $vars(\boldsymbol{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$; we call γ the* guard *of the generalized literal. A rule $r : \alpha \leftarrow \beta$ is* guarded *if every generalized literal in r is guarded, and there is an atom $\gamma_b \in \beta^+$ such that $vars(r) \subseteq vars(\gamma_b)$; we call γ_b a* body guard *of r. It is* fully guarded *if it is guarded and there is a $\gamma_h \subseteq \alpha^-$ such that $vars(r) \subseteq vars(\gamma_h)$; γ_h is called a* head guard *of r.*

An EP P is a (fully) guarded EP ((F)GEP) *if every non-free rule in P is (fully) guarded.*

Example 6. Reconsider the EP from Example 3. r_1, r_2, and r_3 are guarded with guard $f(X, Y)$. The generalized literal in r_4 is guarded by $f(X, Y)$, and r_4 itself is guarded by $q(Y)$. Note that r_5 does not influence the guardedness as it is a free rule.

Every fully guarded EP is guarded. Vice versa, we can transform every guarded EP into an equivalent fully guarded one.

Example 7. Take the guarded EP consisting of the rules r_1 and r_5 from Example 3. We rewrite r_1 as the fully guarded rule $q(X) \vee not\ f(X, Y) \leftarrow f(X, Y)$, i.e. take the body guard and write it negated in the head, where it serves as head guard. Intuitively, rules in the original EP where the body guard cannot be satisfied are removed in the GL-reduct of the new EP; if the body guard is true then the GL-reduct removes the head guard from the head. The effect is in both cases the same.

For a GEP P, P^{f} is P with the non-free rules $\alpha \leftarrow \beta$ replaced by $\alpha \cup not\ \gamma_b \leftarrow \beta$ for the body guard γ_b of $\alpha \leftarrow \beta$. For a GEP P, we have that P^{f} is a FGEP, where the head guard of each non-free rule is equal to the body guard. Moreover, the size of P^{f} is linear in the size of P.

Proposition 3. *Let P be a GEP. 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}.*

We have that the construction of a p-EP retains the guardedness properties.

Proposition 4. *Let P be an EP. Then, P is a (F)GEP iff P_p is a (F)GEP.*

For a fully guarded p-EP P, we can rewrite $\mathtt{comp}(P)$ as the equivalent μGF formulas $\mathtt{gcomp}(P)$. For a guarded generalized literal $\xi \equiv \forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$, define $\xi^g = \forall \boldsymbol{Y} \cdot \gamma \Rightarrow \psi \vee \neg\phi'$, where, since the generalized literal is guarded, $\phi = \gamma \wedge \phi'$, and $vars(\boldsymbol{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$, making formula ξ^g a guarded formula. The extension of this operator for sets (or boolean formulas) of generalized literals is as usual.

$\mathtt{gcomp}(P)$ is $\mathtt{comp}(P)$ with the following modifications.

- Formula $\exists X \cdot \mathbf{true}$ is replaced by
$$\exists X \cdot X = X\ , \tag{13}$$
such that it is guarded by $X = X$.
- Formula (7) is removed if $r : \alpha \leftarrow \beta$ is free or otherwise replaced by
$$\forall \boldsymbol{Y} \cdot \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \{\gamma_b\}) \vee \bigvee \beta^- \vee \bigvee \neg(\beta^{\mathrm{x}})^g\ , \tag{14}$$
where γ_b is a body guard of r, thus we have logically rewritten the formula such that it is guarded. If r is a free rule of the form $q(\boldsymbol{t}) \vee not\ q(\boldsymbol{t}) \leftarrow$ we have $\forall Y \cdot \mathbf{true} \Rightarrow q(\boldsymbol{t}) \vee \neg q(\boldsymbol{t})$ which is always true and can thus be removed from $\mathtt{comp}(P)$.
- Formula (8) is replaced by the formulas
$$\forall \boldsymbol{Y} \cdot r(\boldsymbol{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{15}$$
and
$$\forall \boldsymbol{Y} \cdot \gamma_h \Rightarrow r(\boldsymbol{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \setminus \{\gamma_h\})\ , \tag{16}$$
where γ_h is a head guard of $\alpha \leftarrow \beta$. We thus rewrite an equivalence as two implications where the first implication is guarded by $r(\boldsymbol{Y})$ and the second one is guarded by the head guard of the rule - hence the need for a fully guarded program, instead of just a guarded one.
- Formula (9) is replaced by the formulas
$$\forall \boldsymbol{Z} \cdot g(\boldsymbol{Z}) \Rightarrow \phi \tag{17}$$
and
$$\forall \boldsymbol{Z} \cdot \gamma \Rightarrow g(\boldsymbol{Z}) \vee \neg\phi' \tag{18}$$
where $\phi = \gamma \wedge \psi$ by the guardedness of the generalized literal $\forall \boldsymbol{Y} \cdot \phi \Rightarrow \psi$. We thus rewrite an equivalence as two implications where the first one is guarded by $g(\boldsymbol{Z})$ ($vars(\phi) = \boldsymbol{Z}$ by definition of g), and the second one is guarded by γ ($vars(g(\boldsymbol{Z}) \vee \neg\phi') = vars(\boldsymbol{Z}) = vars(\gamma)$).

– For every $E(r)$ in (10), replace $E(r)$ by

$$E'(r) \equiv \bigwedge_{t_i \notin \boldsymbol{Y}} X_i = t_i \wedge \exists \boldsymbol{Z} \cdot (\bigwedge \beta^+[p|W] \wedge \bigwedge \gamma \wedge r(\boldsymbol{Y}))[t_i \in \boldsymbol{Y}|X_i] \ , \quad (19)$$

with $\boldsymbol{Z} = \boldsymbol{Y} \setminus \{t_i \mid t_i \in \boldsymbol{Y}\}$, i.e. move all $X_i = t_i$ where t_i is constant out of the scope of the quantifier, and remove the others by substituting each t_i in $\bigwedge \beta^+[p|W] \wedge \bigwedge \gamma \wedge r(\boldsymbol{Y})$ by X_i. This rewriting makes sure that every (free) variable in the quantified part of $E'(R)$ is guarded by $r(\boldsymbol{Y})[t_i \in \boldsymbol{Y}|X_i]$.

Example 8. The rule $r : p(X) \vee not\ p(X) \leftarrow p(X), [\forall Y \cdot p(Y) \wedge p(b) \Rightarrow p(a)]$ constitutes a fully guarded p-EP P. The generalized literal is guarded by $p(Y)$ and the rule by head and body guard $p(X)$. $\mathtt{sat}(P)$ contains the formula $\forall X \cdot p(X) \wedge (\forall Y \cdot p(Y) \wedge p(b) \Rightarrow p(a)) \Rightarrow p(X) \vee \neg p(X)$, $\mathtt{gl}(P)$ consists of $\forall X \cdot r(X) \Leftrightarrow p(X)$, $\mathtt{gli}(P)$ is the formula $\forall Y \cdot g(Y) \Leftrightarrow p(Y) \wedge p(b)$ and $E(r) \equiv \exists X \cdot X_1 = X \wedge W(X) \wedge (\forall Y \cdot g(Y) \Rightarrow W(a)) \wedge r(X)$.

$\mathtt{gcomp}(P)$ consists then of the corresponding guarded formulas:

– $\forall X \cdot p(X) \Rightarrow p(X) \vee \neg p(X) \vee \neg(\forall Y \cdot p(Y) \Rightarrow p(a) \vee \neg p(b))$,
– $\forall X \cdot r(X) \Rightarrow p(X)$,
– $\forall X \cdot p(X) \Rightarrow r(X)$,
– $\forall Y \cdot g(Y) \Rightarrow p(Y) \wedge p(b)$,
– $\forall Y \cdot p(Y) \Rightarrow g(Y) \vee \neg p(b)$, and
– $E'(r) \equiv W(X_1) \wedge (\forall Y \cdot g(Y) \Rightarrow W(a)) \wedge r(X_1)$.

As $\mathtt{gcomp}(P)$ is basically a linear logical rewriting of $\mathtt{comp}(P)$, they are equivalent. Moreover, $\bigwedge \mathtt{gcomp}(P)$ is an alternation-free μGF formula.

Proposition 5. *Let P be a fully guarded p-EP. (U, M) is a model of $\mathtt{comp}(P)$ iff (U, M) is a model of $\mathtt{gcomp}(P)$.*

Proposition 6. *Let P be a fully guarded p-EP. Then, $\bigwedge \mathtt{gcomp}(P)$ is an alternation-free μGF formula.*

For a GEP P, we have that P^{f} is a FGEP. By Proposition 4, we have that $(P^{\mathrm{f}})_p$ is a fully guarded p-EP, thus the formula $\mathtt{gcomp}((P^{\mathrm{f}})_p)$ is defined. By Proposition 3, q is satisfiable w.r.t. P iff q is satisfiable w.r.t. P^{f}. By Theorem 1, we have that q is satisfiable w.r.t. P^{f} iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{comp}((P^{\mathrm{f}})_p)$ is satisfiable. Finally, Proposition 5 yields that q is satisfiable w.r.t. P iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable.

The polynomial reduction in Theorem 1 is the worst reduction used, thus yielding the upper bound for the overall reduction.

Theorem 2. *Let P be a GEP and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable. Moreover, this reduction is polynomial.*

For a GEP P, we have, by Proposition 6, that $\bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is an alternation-free μGF formula such that $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is a μGF sentence.

Corollary 1. *Satisfiability checking w.r.t. GEPs can be polynomially reduced to satisfiability checking of alternation-free μGF-formulas.*

Since satisfiability checking of μGF formulas is 2-EXPTIME-complete (Proposition [1.1] in [11]), satisfiability checking w.r.t. GEPs is, by Corollary 1, in 2-EXPTIME.

Corollary 2. *Satisfiability checking w.r.t. GEPs is in 2-*EXPTIME.

Thus, adding generalized literals to guarded programs does not come at the cost of increased complexity of reasoning, as also for guarded programs without generalized literals, reasoning is in 2-EXPTIME [14]. In [14], we established 2-EXPTIME-completeness for satisfiability checking w.r.t. guarded programs (without generalized literals). Since every guarded program is a GEP, 2-EXPTIME-hardness w.r.t. GEPs follows.

Theorem 3. *Satisfiability checking w.r.t. GEPs is 2-*EXPTIME*-complete.*

To conclude this section, we illustrate the use of open answer set programming with GEPs as a general purpose knowledge representation formalism by simulating satisfiability checking of *computation tree logic (CTL)* [4, 5] formulas. Let AP be the finite set of available proposition symbols. Computation tree logic (CTL) formulas are defined as follows[13]: every proposition symbol $P \in AP$ is a formula, if p and q are formulas, so are $p \wedge q$ and $\neg p$, if p and q are formulas, then $\mathsf{EG}p$, $\mathsf{E}(p\ \mathsf{U}\ q)$, and $\mathsf{EX}p$ are formulas. The semantics of a CTL formula is given by *(temporal) structures*. A structure K is a tuple (S, R, L) with S a countable set of states, $R \subseteq S \times S$ a total relation on S, i.e. $\forall s \in S \cdot \exists t \in S \cdot (s,t) \in R$, and $L : S \to 2^{AP}$ a function labeling states with propositions. Intuitively, R indicates the permitted transitions between states and L indicates which propositions are true at certain states.

A path π in K is an infinite sequence of states $(s_0, s_1, \ldots)$ such that $(s_{i-1}, s_i) \in R$ for each $i > 0$. For a path $\pi = (s_0, s_1, \ldots)$, we denote the element s_i with π_i. For a structure $K = (S, R, L)$, a state $s \in S$, and a formula p, we inductively define when K is a *model* of p at s, denoted $K, s \models p$:

- $K, s \models P$ iff $P \in L(s)$ for $P \in AP$,
- $K, s \models \neg p$ iff not $K, s \models p$.
- $K, s \models p \wedge q$ iff $K, s \models p$ and $K, s \models q$,
- $K, s \models \mathsf{EG}p$ iff there exists a path π in K with $\pi_0 = s$ and $\forall k \geq 0 \cdot K, \pi_k \models p$,
- $K, s \models \mathsf{E}(p\ \mathsf{U}\ q)$ iff there exists a path π in K with $\pi_0 = s$ and $\exists k \geq 0 \cdot (K, \pi_k \models q \wedge \forall j < k \cdot K, \pi_j \models p)$,
- $K, s \models \mathsf{EX}p$ iff there is a $(s,t) \in R$ and $K, t \models p$.

The expression $K, s \models \mathsf{EG}p$ can be read as "there is some path from s along which p holds Globally (everywhere)", $K, s \models \mathsf{EX}p$ as "there is some neXt state where p holds", and $K, s \models \mathsf{E}(p\ \mathsf{U}\ q)$ as "there is some path from s along which p holds Until q holds (and q eventually holds)". A structure $K = (S, R, L)$ *satisfies* a CTL formula p if there is a state $s \in S$ such that $K, s \models p$; we also call K a *model* of p. A CTL formula p is *satisfiable* iff there is a model of p.

For a CTL formula p, let $clos(p)$ be the *closure* of p: the set of subformulas of p. We construct a GEP $G \cup D_p$ consisting of a generating part G and a defining part D_p. The

[13] In order to make the treatment as simple as possible, we do not include formulas involving the path quantifier A. However, as indicated in [15], the defined constructs are adequate, i.e. every CTL formula can be rewritten using only those, while preserving satisfiability.

guarded program G contains free rules (g_1) for every proposition $P \in AP$, free rules (g_2) allowing for state transitions, and rules (g_3) that ensure that the transition relation is total:

$$[P](S) \vee not\ [P](S) \leftarrow \tag{g_1}$$
$$next(S,N) \vee not\ next(S,N) \leftarrow \tag{g_2}$$
$$succ(S) \leftarrow next(S,N) \qquad \leftarrow S = S, not\ succ(S) \tag{g_3}$$

where $[P]$ is the predicate corresponding to the proposition P. The $S = S$ is necessary merely for having guarded rules; note that any rule containing only one (free) variable can be made guarded by adding such an equality.

The GEP D_p introduces for every non-propositional CTL formula in $clos(p)$ the following rules (we write $[q]$ for the predicate corresponding to the CTL formula $q \in clos(p)$); as noted before we tacitly assume that rules containing only one (free) variable S are guarded by $S = S$:

$$[\neg q](S) \leftarrow not\ [q](S) \tag{d_1}$$
$$[q \wedge r](S) \leftarrow [q](S), [r](S) \tag{d_2}$$
$$[\mathsf{EG}q](S) \leftarrow not\ [\mathsf{AF}\neg q](S) \tag{d_3^1}$$
$$[\mathsf{AF}\neg q](S) \leftarrow not\ [q](S) \tag{d_3^2}$$
$$[\mathsf{AF}\neg q](S) \leftarrow \forall N \cdot next(S,N) \Rightarrow [\mathsf{AF}\neg q](N) \tag{d_3^3}$$
$$[\mathsf{E}(q\ \mathsf{U}\ r)](S) \leftarrow [r](S) \tag{d_4}$$
$$[\mathsf{E}(q\ \mathsf{U}\ r)](S) \leftarrow [q](S), next(S,N), [\mathsf{E}(q\ \mathsf{U}\ r)](N) \tag{d_5}$$
$$[\mathsf{EX}q](S) \leftarrow next(S,N), [q](N) \tag{d_6}$$

The rules ($d_{\{1,2,6\}}$) are direct translations of the CTL semantics.

Rules (d_3^2) and (d_3^3) ensure that if $[\mathsf{AF}\neg q](s)$ holds, then on all paths from s we eventually reach a state where q does not hold. In particular this is true if q does not hold in the current state (d_3^2), or if it holds for all successors (d_3^3); minimality of open answer sets ensures that after a finite time a state where q does not hold is reached. Rule (d_3^1) then defines $[\mathsf{EG}q]$ as the negation of $[\mathsf{AF}\neg q]$.

Rules (d_4) and (d_5) are in accordance with the characterization $\mathsf{E}(q\ \mathsf{U}\ r) \equiv r \vee (q \wedge \mathsf{EXE}(q\ \mathsf{U}\ r))$ [4], and make implicit use of the minimality of answer sets to eventually ensure realization of r.

Theorem 4. *Let p be a CTL formula. p is satisfiable iff $[p]$ is satisfiable w.r.t. the GEP $G \cup D_p$.*

Since CTL satisfiability checking is EXPTIME-complete [4] and satisfiability checking w.r.t. GEPs is 2-EXPTIME-complete (Theorem 3), the reduction from CTL to GEPs does not seem to be optimal. However, we can show that the particular GEP $G \cup D_p$ is a *bound* GEP for which reasoning is indeed EXPTIME-complete and thus optimal.

Define the *width* of a formula ψ as the maximal number of free variables in its subformulas [10]. We define *bound* programs by looking at their first order form and the arity of its predicates.

Definition 3. *Let P be an EP. Then, P is* bound *if every formula in $\mathtt{sat}(P)$ is of bounded width and the predicates in P have a bounded arity.*

For a CTL formula p, one has that $G \cup D_p$ is a bound GEP. Indeed, every subformula of formulas in $\mathtt{sat}(G \cup D_p)$ contains at most 2 free variables and the maximum arity of the predicates is 2 as well.

Let P be a bound GEP. We have that $(P^{\mathrm{f}})_p$ is bound and one can check that $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is of bounded width.

Using Theorem 2, one can reduce satisfiability checking of a bound GEP to satisfiability of a μGF-formula with bounded width. The latter can be done in EXPTIME by Theorem 1.2 in [11], such that satisfiability checking w.r.t. bound GEPs is in EXPTIME.

The EXPTIME-hardness follows from Theorem 4 and the EXPTIME-hardness of CTL satisfiability checking [4].

Theorem 5. *Satisfiability checking w.r.t. bound GEPs is* EXPTIME*-complete.*

5 Equivalence with Datalog LITE

We define *Datalog* LITE as in [8]. A *Datalog rule* is a rule $\alpha \leftarrow \beta$ where $\alpha = \{a\}$ for some atom a and β does not contain generalized literals. 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[14], is given by the unique (subset) minimal model whose restriction to the extensional predicates yields $\mathcal{U}$. We refer to [1] for more details.

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 the minimal model M of Σ_P with input $\mathcal{U}$ contains $q(\boldsymbol{a})$, where Σ_P are the first-order clauses corresponding to P, see [1]. We call (P, q) satisfiable if there exists a $\mathcal{U}$ and an $\boldsymbol{a}$ such that $\boldsymbol{a} \in (P, q)(\mathcal{U})$.

A program P is a *stratified Datalog program* if it can be written as a union of basic Datalog programs $(P_1, \ldots, P_n)$, 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. For an input structure $\mathcal{U}$ and a stratified program $P = (P_1, \ldots, P_n)$, define as in [1]:

$$\begin{aligned} \mathcal{U}_0 &\equiv \mathcal{U} \\ \mathcal{U}_i &\equiv \mathcal{U}_{i-1} \cup P_i(\mathcal{U}_{i-1}|edb(P_i)) \end{aligned}$$

[14] We assume that an input structure always defines equality, and that it does so as the identity relation.

where $S_i \equiv P_i(\mathcal{U}_{i-1}|edb(P_i))$ is the minimal model of Σ_{P_i} among those models of Σ_{P_i} whose restriction to the extensional predicates of P_i is equal to $\mathcal{U}_{i-1}|edb(P_i)$. The least fixed point model of P is per definition $\mathcal{U}_n$.

A *Datalog* LITE *generalized literal* is a generalized literal $\forall \boldsymbol{Y} \cdot a \Rightarrow b$ where a and b are atoms and $vars(b) \subseteq vars(a)$. Note that Datalog LITE generalized literals $\forall \boldsymbol{Y} \cdot a \Rightarrow b$ can be replaced by the equivalent $\forall \boldsymbol{Z} \cdot a \Rightarrow b$ where $\boldsymbol{Z} \equiv \boldsymbol{Y} \setminus \{Y \mid Y \notin vars(a)\}$, i.e. with the variables that are not present in the formula $a \Rightarrow b$ removed from the quantifier. After such a rewriting, Datalog LITE generalized literals are guarded according to Definition 2.

A *Datalog* LITE program is a stratified Datalog program, possibly containing Datalog LITE 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 $\forall \boldsymbol{Y} \cdot a \Rightarrow b$ 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, as in [8]: every generalized literal $\forall \boldsymbol{Y} \cdot a \Rightarrow b$ is instantiated (for any $\boldsymbol{x}$ grounding the free variables $\boldsymbol{X}$ in the generalized literal) by $\bigwedge\{b[\boldsymbol{X} \mid \boldsymbol{x}][\boldsymbol{Y} \mid \boldsymbol{y}] \mid a[\boldsymbol{X} \mid \boldsymbol{x}][\boldsymbol{Y} \mid \boldsymbol{y}] \text{ is true}\}$, which is well-defined since a is defined in a lower stratum than the rule where the generalized literal appears.

For stratified Datalog programs, least fixed point models with as input the identity relation on a universe U coincide with open answer sets with universe U.

Proposition 7. *Let $P = (P_1, \ldots, P_n)$ be a stratified Datalog program, possibly with generalized literals, U a universe for P, and l a literal. For the least fixed point model $\mathcal{U}_n$ of P with input $\mathcal{U} = \{id(U)\}$, we have $\mathcal{U}_n \models l$ iff there exists an open answer set (U, M) of P such that $M \models l$.*

Moreover, for any open answer set (U, M) of P, we have that $M = \mathcal{U}_n \setminus id(U)$.

From Proposition 7, we obtain a generalization of Corollary 2 in [6] (*If Π is stratified, then its unique stable model is identical to its fixed point model.*) for stratified programs with generalized literals and an open answer set semantics.

Corollary 3. *Let P be a stratified Datalog program, possibly with generalized literals, and U a universe for P. The unique open answer set (U, M) of P is identical to its least fixed point model (minus the equality atoms) with input structure $id(U)$.*

We generalize Proposition 7, to take into account arbitrary input structures $\mathcal{U}$. For a stratified Datalog program P, possibly with generalized literals, define $F_P \equiv \{q(\boldsymbol{X}) \vee not\ q(\boldsymbol{X}) \leftarrow \mid q \text{ extensional (but not } =) \text{ in } P\}$.

Proposition 8. *Let $P = (P_1, \ldots, P_n)$ be a stratified Datalog program, possibly with generalized literals, and l a literal. There exists an input structure $\mathcal{U}$ for P with least fixed point model $\mathcal{U}_n$ such that $\mathcal{U}_n \models l$ iff there exists an open answer set (U, M) of $P \cup F_P$ such that $M \models l$.*

The set of free rules F_P ensures a free choice for extensional predicates, a behavior that corresponds to the free choice of an input structure for a Datalog program P. Note that $P \cup F_P$ is not a Datalog program anymore, due to the presence of naf in the heads of F_P.

Define a Datalog LITEM program as a Datalog LITE program where all rules are guarded. As we will see below this is not a restriction. As F_P contains only free rules, $P \cup F_P$ is a GEP if P is a Datalog LITEM program. Furthermore, the size of the GEP $P \cup F_P$ is linear in the size of P.

Proposition 9. *Let P be a Datalog* LITEM *program. Then, $P \cup F_P$ is a GEP whose size is linear in the size of P.*

By Propositions 8 and 9, satisfiability checking of Datalog LITEM queries can be reduced to satisfiability checking w.r.t. GEPs.

Theorem 6. *Let (P, q) be a Datalog* LITEM *query. Then, (P, q) is satisfiable iff q is satisfiable w.r.t. $P \cup F_P$. Moreover, this reduction is linear.*

With a similar reasoning as in [14], one can show that the opposite direction holds as well. In [8], Theorem 8.5., a Datalog LITEM query (π_φ, q_φ) was defined for an alternation-free μGF sentence 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 least fixed point model of $(\pi_\varphi, q_\varphi)(M \cup id(U))$. For the formal details of this reduction, we refer to [8]. Satisfiability checking w.r.t. GEPs can then be polynomially reduced to satisfiability checking in Datalog LITEM. Indeed, by Theorem 2, we have that q is satisfiable w.r.t. a GEP P iff $\varphi \equiv \exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable. Since φ is an alternation-free 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 polynomial in the size of P and the query (π_φ, q_φ) is quadratic in φ [8], resulting in a polynomial reduction.

Theorem 7. *Let P be a GEP, q an n-ary predicate in P and φ the μGF sentence $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \mathbf{0}, q) \wedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$. q is satisfiable w.r.t. P iff (π_φ, q_φ) is satisfiable. Moreover, this reduction is polynomial.*

Theorems 6 and 7 lead to the conclusion that Datalog LITEM and open ASP with GEPs are equivalent (i.e. satisfiability checking in either one of the formalisms can be polynomially reduced to satisfiability checking in the other). Furthermore, since Datalog LITEM, Datalog LITE, and alternation-free μGF are equivalent as well [8], we have the following concluding result.

Theorem 8. *Datalog* LITE*, alternation-free μGF, and open ASP with GEPs are equivalent.*

6 ω-Restricted Logic Programs

A class of logic programs with function symbols are the *ω-restricted programs* from [19]. The Herbrand Universe of ω-restricted programs is possibly infinite (in the presence of function symbols), however, answer sets are guaranteed to be finite, exactly by

the structure of ω-restricted programs. Informally, an ω-restricted program consists of a stratified part and a part that cannot be stratified (the *ω-stratum*), where every rule is such that every variable in a rule is "guarded" by an atom of which the predicate is defined in a strictly lower stratum. The answer sets of ω-restricted programs can then be computed by instantiating the strata from the bottom up. We refer to [19] for precise definitions.

We extend the definition of universe for programs that contain function symbols. A *universe* U for a program P is a non-empty countable superset of the Herbrand Universe $\mathcal{H}_P$ of P. Thus, a universe U is equal to $\mathcal{H}_P \cup X$ for some countable X; as usual, we call the elements from $U \setminus \mathcal{H}_P$ *anonymous*.

For ω-restricted programs, the open answer set semantics coincides with the normal answer set semantics.

Theorem 9. *Let P be an ω-restricted program and U a universe for P. (U, M) is an open answer set of P iff M is an answer set of P.*

Since checking whether there exists an answer set of an ω-restricted program is in general 2-NEXPTIME-complete [19], we have, with Theorem 9, 2-NEXPTIME-completeness for consistency checking under the open answer set semantics for ω-restricted programs, where consistency checking involves checking whether there exists an open answer set of a program.

Theorem 10. *Consistency checking w.r.t. ω-restricted programs is* 2-NEXPTIME-*complete.*

Furthermore, since reasoning with ω-restricted programs is implemented in the SMODELS reasoner [18], Theorem 9 implies an implementation of the open answer set semantics for ω-restricted programs as well.

In [20], ω-restricted programs allow for *cardinality constraints* and *conditional literals*. Conditional literals have the form $X.L : A$ where X is a set of variables, A is an atom (the condition) and L is an atom or a naf-atom. Intuitively, conditional literals correspond to generalized literals $\forall X \cdot A \Rightarrow L$, i.e., the defined reducts add instantiations of L to the body if the corresponding instantiation of A is true. However, conditional literals appear only in cardinality constraints $Card(b, S)$[15] where S is a set of literals (possibly conditional), such that a *for all* effect such as with generalized literals cannot be obtained with conditional literals.

Take, for example, the rule $q \leftarrow [\forall X \cdot b(X) \Rightarrow a(X)]$ and a universe $U = \{x_1, x_2\}$ with an interpretation containing $b(x_1)$ and $b(x_2)$. The reduct will contain a rule $q \leftarrow a(x_1), a(x_2)$ such that, effectively, q holds only if a holds everywhere where b holds. The equivalent rule rewritten with a conditional literal would be something like $q \leftarrow Card(n, \{X.a(X) : b(X)\})$, resulting in a rule $q \leftarrow Card(n, \{a(x_1), a(x_2)\})$. In order to have the *for all* effect, we have that n must be 2. However, we cannot know this n in advance, making it impossible to express a *for all* restriction.

Further note that consistent ω-restricted programs (with cardinality constraints and conditional literals) always have finite answer sets, which makes a reduction from GEPs (in which infinity axioms can be expressed) to ω-restricted programs non-trivial.

[15] $Card(b, S)$ is true if at least b elements from S are true.

7 Conclusions and Directions for Further Research

We defined GEPs, guarded programs with generalized literals, under an open answer set semantics, and showed 2-EXPTIME-completeness of satisfiability checking by a reduction to μGF. Furthermore, we translated Datalog LITEM programs to GEPs, and generalized the result that the unique answer set of a stratified program is identical to its least fixed point.

We plan to extend GEPs to loosely guarded EPs, where a guard may be a set of atoms; a reduction to the loosely guarded fixed point logic should then provide for decidability. More liberal generalized literals, with the consequent a conjunction of atoms and naf-atoms instead of just an atom, does not affect the definition of the GLi-reduct, but the FPL translation requires modification to ensure no fixed point variable appears negatively.

We plan to look into the correspondence with Datalog and use decidability results for Datalog satisfiability checking, as, e.g., in [12], to search for decidable fragments under an open answer set semantics.

Although adding generalized literals to guarded programs does not increase the complexity of reasoning, it does seem to increase expressivity: one can, for example, express infinity axioms. Given the close relation with Datalog LITE and the fact that Datalog LITE without generalized literals cannot express well-founded statements, it seems unlikely that guarded programs without generalized literals can express infinity axioms; this is subject to further research.

We only considered generalized literals in the positive body. If the antecedents in generalized literals are atoms, it seems intuitive to allow also generalized literals in the negative body. E.g., take a rule $\alpha \leftarrow \beta, not\ [\forall X \cdot b(X) \Rightarrow a(X)]$; it seems natural to treat $not\ [\forall X \cdot b(X) \Rightarrow a(X)]$ as $\exists X \cdot b(X) \wedge \neg a(X)$ such that the rule becomes $\alpha \leftarrow \beta, b(X), not\ a(X)$. A rule like $[\forall X \cdot b(X) \Rightarrow a(X)] \vee \alpha \leftarrow \beta$ is more involved and it seems that the generalized literal can only be intuitively removed by a modified GLi-reduct.

We established the equivalence of open ASP with GEPs, alternation-free μGF, and Datalog LITE. Intuitively, Datalog LITE is not expressive enough to simulate normal μGF since such μGF formulas could contain negated fixed point variables, which would result in a non-stratified program when translating to Datalog LITE [8]. Open ASP with GEPs does not seem to be sufficiently expressive either: fixed point predicates would need to appear under negation as failure, however, the GL-reduct removes naf-literals, such that, intuitively, there is no real recursion through naf-literals. Note that it is unlikely (but still open) whether alternation-free μGF and normal μGF are equivalent, i.e., whether the alternation hierarchy can always be collapsed.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. J. Van Benthem. Dynamic Bits and Pieces. In *ILLC research report*. University of Amsterdam, 1997.
3. A. K. Chandra and D. Harel. Horn Clauses and the Fixpoint Query Hierarchy. In *Proc. of PODS '82*, pages 158–163. ACM Press, 1982.

4. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers B.V., 1990.
5. 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.
6. 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.
7. M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993.
8. 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.
9. E. Grädel. Guarded Fixed Point Logic and the Monadic Theory of Trees. *Theoretical Computer Science*, 288:129–152, 2002.
10. E. Grädel. Model Checking Games. In *Proceedings of WOLLIC 02*, volume 67 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.
11. E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *Proc. of LICS '99*, pages 45–54. IEEE Computer Society, 1999.
12. A. Halevy, I. Mumick, Y. Sagiv, and O. Shmueli. Static Analysis in Datalog Extensions. *Journal of the ACM*, 48(5):971–1012, 2001.
13. 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.
14. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Guarded Open Answer Set Programming. In *8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, number 3662 in LNAI, pages 92–104. Springer, 2005.
15. M. R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.
16. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
17. J. Lloyd and R. Topor. Making Prolog More Expressive. *J. Log. Program.*, 1(3):225–240, 1984.
18. P. Simons. SMODELS homepage. http://www.tcs.hut.fi/Software/smodels/.
19. T. Syrjänen. Omega-restricted Logic Programs. In *Proc. of LPNMR*, volume 2173 of *LNAI*, pages 267–279. Springer, 2001.
20. T. Syrjänen. Cardinality Constraint Programs. In *Proc. of JELIA'04*, pages 187–200. Springer, 2004.