Open Answer Set Programming with Guarded Programs

STIJN HEYMANS
Digital Enterprise Research Institute (DERI)
Leopold-Franzens-Universität
Innsbruck, Austria
stijn.heymans@deri.org
and
DAVY VAN NIEUWENBORGH, and DIRK VERMEIR
Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{dvnieuwe,dvermeir}@vub.ac.be

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 for the first time, a characterization of an answer set semantics by μLGF formulas.

We further extend the open answer set semantics for programs with generalized literals. Such *generalized programs (gPs)* have interesting properties, e.g., the ability to express infinity axioms. We restrict the syntax of gPs such that both rules and generalized literals are *guarded*. Via a translation to guarded fixed point logic, we deduce 2-EXPTIME-completeness of satisfiability checking in such *guarded gPs* (GgPs). *Bound GgPs* are restricted GgPs with EXPTIME-complete satisfiability checking, but still sufficiently expressive to optimally simulate *computation tree logic* (CTL). We translate Datalog LITE programs to GgPs, establishing equivalence of GgPs under an open answer set semantics, alternation-free μGF, and Datalog LITE.

Categories and Subject Descriptors: I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Logic Programming*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms: Theory

Additional Key Words and Phrases: Answer Set Programming, Open Domains, Fixed Point Logic

---

This is a revised and extended version of [Heymans et al. 2005a] and [Heymans et al. 2006a].
Stijn Heymans is supported by the European Commission under the projects Knowledge Web and SUPER; by the FFG (Österreichische Forschungsförderungsgeselleschaft mbH) under the projects RW^2, SemNetMan, and SEnSE. Davy Van Nieuwenborgh is supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

1. INTRODUCTION

In closed answer set programming (ASP) [Gelfond and Lifschitz 1988], 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 is infeasible. One wants to be able to test satisfiability of a predicate p in a schema independent of any associated data, see, e.g., conceptual modeling as in the *Object-role Modeling* paradigm [Halpin 2001].

For answer set programming, this problem was solved in [Gelfond and Przymusinska 1993], where k-belief sets are the answer sets of a program that is extended with k extra constants (reasoning with k-belief sets was shown to be undecidable in [Schlipf 1993]). We extended this idea, e.g., in [Heymans et al. 2005b], by allowing for arbitrary, thus possibly infinite, universes[1]. *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. Open Answer Set Programming solves the above conceptual modeling problem, confirmed by the ability of Open Answer Set Programming to simulate several expressive Description Logics [Heymans et al. 2005b]. Moreover, as it is a rule-based formalism Open Answer Set Programming is thus very suitable to function as an integrating formalism of Description Logics and Logic Programming.

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 [Gelfond and Lifschitz 1988]), 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) [Grädel and Walukiewicz 1999] formulas. 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 $p(\mathbf{x})$ 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 [Chandra and Harel 1982], Horn clauses were translated to FPL formulas and in [Gottlob et al. 2002] reasoning with an extension of stratified Datalog is reduced to FPL, but, to the best of our knowledge, this is the first encoding of an answer

[1]Note that answer sets for programs with (infinite) universes were also considered in [Schlipf 1995].

set semantics in FPL.

In [Lin and Zhao 2002; Lee and Lifschitz 2003], ASP with (finite) propositional programs is reduced to propositional satisfiability checking. The translation makes the loops in a program explicit and ensures that atoms $p(\mathbf{x})$ 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 $p(\mathbf{x})$, is undecidable. It is well-known that satisfiability checking in FOL is undecidable, and thus the extension to FPL is too[Moschovakis 1974]. However, expressive decidable fragments of FPL have been identified [Grädel and Walukiewicz 1999]: *(loosely) guarded fixed point logic* (μ(L)GF) extends the *(loosely) guarded fragment* (L)GF of FOL with fixed point predicates.

GF is identified in [Andréka et al. 1998] 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 [Van Benthem 1997], GF is 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 [Grädel 1999], as are their extensions with fixed point predicates μGF and μLGF [Grädel and Walukiewicz 1999].

We identify a syntactically restricted class of programs, *(loosely) guarded programs ((L)GPs)*, for which the FPL translation falls in (alternation-free[2]) 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 introduce 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 [Gottlob et al. 2002] 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 $\forall \mathbf{Y} \cdot a \Rightarrow b$ for atoms a and b. It has an appropriately adapted bottom-up fixed point semantics. Datalog LITE is

[2] μ(L)GF without nested fixed point variables in alternating least and greatest fixed point formulas.

devised to ensure linear time model checking while being expressive enough to capture *computational tree logic* [Emerson and Clarke 1982] and alternation-free μ-calculus [Kozen 1983]. Moreover, it is shown to be equivalent to alternation-free μGF. Our reduction of GPs to alternation-free 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 [Gottlob et al. 2002], and, since satisfiability checking of GF formulas is 2-EXPTIME-hard [Grädel 1999], we obtain 2-EXPTIME-completeness for satisfiability checking w.r.t. (L)GPs.

We next extend programs with generalized literals, resulting in *generalized programs (gPs)*. A generalized literal is a first-order formula of the form $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ where $\mathbf{Y}$ is a sequence of variables, ϕ is a finite boolean combination of atomic formula and ψ is an atom. Intuitively, such a generalized literal is true in an open interpretation (U, M) if for all substitutions[3] $[\mathbf{Y} \mid \mathbf{y}]$, $\mathbf{y}$ in U, such that $\phi[\mathbf{Y} \mid \mathbf{y}]$ is true in M, $\psi[\mathbf{Y} \mid \mathbf{y}]$ is true in M.

Generalized literals $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$, with ϕ an atom instead of a finite boolean combination of atomic formula, were introduced in Datalog[4] with the language Datalog LITE. 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 *GeLi-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 GeLi-reduct does not contain generalized literals and one can apply the normal answer set semantics, modified to take into account the infinite body.

Just as 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 well as r (and in fact s is the GeLi-reduct of r), but adding new critical devices, implies revisiting s and replacing

[3] As usual, for a finite boolean combination of atomic formula ξ, we use $\xi[\mathbf{Y} \mid \mathbf{y}]$ to denote the formula ξ where all occurrences of Y are replaced by y.

[4] The extension of logic programming syntax with first-order formulas dates back to [Lloyd and Topor 1984].

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.

One can modify the aforementioned FPL translation of programs without generalized literals to take into account generalized literals. With this FPL translation, we then have again a mapping from one undecidable framework into another undecidable framework. We restrict gPs, resulting in *guarded gPs (GgPs)*, 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 if it can be written as a guarded formula in μGF. The FPL translation of GgPs 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 this establishes 2-EXPTIME-completeness for satisfiability checking w.r.t. GgPs. As a consequence, adding generalized literals to a guarded program does not increase the complexity of reasoning.

We further illustrate the expressiveness of (bound) GgPs by simulating reasoning in *computational tree logic (CTL)* [Emerson 1990], a temporal logic. *Temporal logics* [Emerson 1990] are widely used for expressing properties of nonterminating programs. Transformation semantics, such as *Hoare's logic* are not appropriate here since they depend on the program having a final state that can be verified to satisfy certain properties. Temporal logics on the other hand have a notion of (infinite) time and may express properties of a program along a time line, without the need for that program to terminate. E.g., formulas may express that from each state a program should be able to reach its initial state: $\mathsf{AGEF}\mathit{initial}$.

Two well-known temporal logics are *linear temporal logic (LTL)* [Emerson 1990; Sistla and Clarke 1985] and *computation tree logic (CTL)* [Emerson 1990; Emerson and Halpern 1982; Clarke et al. 1986], which differ in their interpretation of time: the former assumes that time is linear, i.e., for every state of the program there is only one successor state, while time is branching for the latter, i.e., every state may have different successor states, corresponding to nondeterministic choices for the program.

Since CTL satisfiability checking is EXPTIME-complete and satisfiability checking w.r.t. GgPs is 2-EXPTIME-complete, a reduction from CTL to GgPs does not seem to be optimal. However, we can show that the particular translation has a special form, i.e., it is *bound*, for which reasoning is EXPTIME-complete and thus optimal.

Finally, we can reduce general Datalog LITE reasoning, i.e., with recursion, to reasoning with GgPs. In particular, we prove a generalization of the well-known result from [Gelfond and Lifschitz 1988] 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[5] 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 GgPs to alternation-free μGF, as well as the equivalence of alternation-free μGF and Datalog LITE [Gottlob et al. 2002], lead to the conclusion that alternation-free μGF, Datalog LITE, and OASP with GgPs, are equivalent, i.e., their satisfiability

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

checking problems can be effectively polynomially reduced to one another.

GgPs are thus just as expressive as Datalog LITE, however, from a knowledge representation viewpoint, GgPs 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, but can be regarded as convenient semantic sugar.

The remainder of the paper starts with an introduction of the open answer set semantics, fixed point logic, and computation tree logic. In Section 3, we reduce satisfiability checking w.r.t. arbitrary logic programs to satisfiability checking of alternation-free fixed point logic formulas. We identify in Section 4 syntactical classes of programs for which this FPL translation falls into the decidable logic μGF or μLGF, i.e., guarded or loosely guarded fixed point logic.

In Section 5, we introduce so-called generalized literals and modify the translation to FPL in Section 6. Section 7 mirrors Section 4 and identifies classes of programs with generalized literals that can be mapped to guarded FPL. In Section 8, we relate the obtained languages under the open answer set semantics to Datalog LITE which has a least fixed point model semantics. Section 9 discusses a translation from CTL to bound guarded programs. Finally, Section 10 contains conclusions and directions for further research.

2. PRELIMINARIES

2.1 Open Answer Set Programming

We introduce open answer set programming (OASP) as in [Heymans et al. 2006b]. *Constants*, *variables*, *terms*, and *atoms* are defined as usual[6]. A *literal* is an atom $p(\mathbf{t})$ or a *naf-atom* $not\ p(\mathbf{t})$.[7] The *positive part* of a set of literals α is $\alpha^+ = \{p(\mathbf{t}) \mid p(\mathbf{t}) \in \alpha\}$ and the *negative part* of α is $\alpha^- = \{p(\mathbf{t}) \mid not\ p(\mathbf{t}) \in \alpha\}$. 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.[8] 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 $q(\mathbf{t}) \vee not\ q(\mathbf{t}) \leftarrow$ for a tuple $\mathbf{t}$ of terms; they enable a choice for the inclusion of atoms. We call a predicate p free if there is a free rule $p(\mathbf{t}) \vee not\ p(\mathbf{t}) \leftarrow$. 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

[6]Note that we do not allow function symbols.

[7]We have no classical negation $\neg$, however, programs with $\neg$ can be reduced to programs without it, see e.g. [Lifschitz et al. 2001a].

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

preds(P) its predicates. A *universe* U for P is a non-empty countable[9] 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 $\mathcal{B}_P^U$ be the set of ground regular atoms that can be formed from a ground program P and the elements in U.

Let I be a subset of some $\mathcal{B}_P^U$. For a ground regular atom $p(\mathbf{t})$, we write $I \models p(\mathbf{t})$ if $p(\mathbf{t}) \in I$; For an equality atom $p(\mathbf{t}) \equiv t = s$, we have $I \models p(\mathbf{t})$ if s and t are equal constants. We have $I \models \mathit{not}\ p(\mathbf{t})$ if $I \not\models p(\mathbf{t})$. 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 $I \models \beta$. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. For a ground program P without *not*, I 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* [Gelfond and Lifschitz 1988] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $I \models \mathit{not}\ \beta^-$ 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 a subset of $\mathcal{B}_P^U$. 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 $\mathbf{x} \in U^n$ such that $p(\mathbf{x}) \in M$. 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. $P \cup \{p'(\mathbf{X}) \leftarrow p(\mathbf{X})\}$. Note that this is a linear reduction.

Example 2.1. Take the program

$$
\begin{array}{lrcl}
r_1: & \mathit{restore}(X) & \leftarrow & \mathit{crash}(X), y(X, Y), \mathit{backSucc}(Y) \\
r_2: & \mathit{backSucc}(X) & \leftarrow & \neg\mathit{crash}(X), y(X, Y), \mathit{not}\ \mathit{backFail}(Y) \\
r_3: & \mathit{backFail}(X) & \leftarrow & \mathit{not}\ \mathit{backSucc}(X) \\
r_4: & & \leftarrow & y(Y_1, X), y(Y_2, X), Y_1 \neq Y_2 \\
r_5: & y(X, Y) \vee \mathit{not}\ y(X, Y) & \leftarrow & \\
r_6: & \mathit{crash}(X) \vee \mathit{not}\ \mathit{crash}(X) & \leftarrow & \\
r_7: & \neg\mathit{crash}(X) \vee \mathit{not}\ \neg\mathit{crash}(X) & \leftarrow &
\end{array}
$$

Rule r_1 represents the knowledge that a system that has crashed on a particular day X (*crash*(X)), can be restored on that day (*restore*(X)) if a backup of the system on the day Y before ($y(X, Y)$ – y stands for *yesterday*) succeeded (*backSucc*(Y)). Backups succeed, if the system does not crash and it cannot be established that the backups at previous dates failed (r_2) and a backup fails if it does not succeed (r_3). Rule r_4 ensures that for a particular today there can be only one tomorrow. Rules r_5, r_6, and r_7 allow to freely introduce y, *crash*, and $\neg$*crash* literals. Indeed, take, e.g., *crash*(x) in an interpretation; the GL-reduct w.r.t. that interpretation

[9]Note that U is countable, as later on, this is needed to be able to use a result from [Grädel 1999] that indicates that the fixed point can be reached at the first ordinal ω.

contains then the rule $crash(x) \leftarrow$ which motivates the presence of the *crash* literal in an (open) answer set. If there is no $crash(x)$ in an interpretation then the GL-reduct removes the rule r_5 (more correctly, its grounded version with x). Below, we formally define rules of such a form as *free rules* in correspondence with the intuition that they allow for a free introduction of literals.

Every open answer set (U, M) of this program that makes *restore* satisfiable, i.e., such that there is a $restore(x) \in M$ for some $x \in U$, must be infinite. An example of such an open answer set M is (we omit U if it is clear from M)

$$\{restore(x), crash(x), backFail(x), y(x, x_1), \\ backSucc(x_1), \neg crash(x_1), y(x_1, x_2) \\ backSucc(x_2), \neg crash(x_2), y(x_2, x_3), \ldots\}$$

One sees that every *backSucc* literal with element x_i enforces a new y-successor x_{i+1} since none of the previously introduced universe elements can be used without violating rule r_4, thus enforcing an infinite open answer set.

Indeed, assume *restore* is satisfiable w.r.t. P. Then, there must be a x_0 in the universe U of some open answer set (U, M) such that $restore(x_0) \in M$. With r_1, we must have that $crash(x_0) \in M$, and there must be some $x_1 \in U$ such that $y(x_0, x_1) \in M$ and $backSucc(x_1) \in M$, and thus, with rule r_2, $\neg crash(x_1) \in M$, $y(x_1, x_2) \in M$ and $backFail(x_2) \notin M$. With $crash(x_0) \in M$ and $\neg crash(x_1) \in M$, we are sure that $x_1 \neq x_0$. With r_3, one must have that $backSucc(x_2) \in M$ such that $x_2 \neq x_0$ for the same reason. Furthermore, $x_2 \neq x_1$, since otherwise $y(x_0, x_1) \in M$ and $y(x_1, x_1) \in M$: with $x_0 \neq x_1$ this is a contradiction with r_4. Thus, summarizing, $x_2 \neq x_1$ and $x_2 \neq x_0$. One can continue this way, and one will be obliged to introduce new x_i's ad infinitum.

Rules $\alpha \leftarrow \beta$ are such that $|\alpha^+| \leq 1$. This restriction ensures that the GL-reduct contains no disjunction in the head anymore, i.e., the head will be an atom or it will be empty. This property of the GL-reduct allows us to define an *immediate consequence operator* [van Emden and Kowalski 1976] T that computes the closure of a set of literals w.r.t. a GL-reduct.

For a program P and an open interpretation (U, M) of P, $T_P^{(U,M)} : \mathcal{B}_{P_U} \rightarrow \mathcal{B}_{P_U}$ is defined as $T(B) = B \cup \{a \mid a \leftarrow \beta \in P_U^M \wedge B \models \beta\}$. Additionally, we define $T^0(B) = B$, and $T^{n+1}(B) = T(T^n(B))$.[10]

Although we allow for infinite universes, we can motivate the presence of atoms in open answer sets in a finite way, where the motivation of an atom is formally expressed by the immediate consequence operator.

THEOREM 2.2. *Let P be a program and (U, M) an open answer set of P. Then, $\forall a \in M \cdot \exists n < \infty \cdot a \in T^n$.*

For the relation of OASP with other logic programming paradigms that allow for (some form of) opennes, we refer to [Heymans et al. 2006b].

[10] We omit the sub- and superscripts (U, M) and P from $T_P^{(U,M)}$ if they are clear from the context and, furthermore, we will usually write T instead of $T(\emptyset)$.

2.2 Fixed Point Logic

Extensions of *first-order logic (FOL)* that allow for the expression of recursive procedures are well-investigated in *finite model theory*, see e.g., [Moschovakis 1974; Immerman 1986]. Also in the presence of infinite models, so-called *fixed point logic (FPL)* proves to be an interesting logic [Flum 1999]. E.g., a decidable subclass of FPL is the *guarded fixed point logic* [Grädel and Walukiewicz 1999], which lifts propositional μ-calculus [Kozen 1983] to a first-order setting.

We assume FOL interpretations are represented as pairs (U, M) where M is an interpretation 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 [Grädel and Walukiewicz 1999], i.e., as an extension of first-order logic, where formulas may additionally be *fixed point formulas* of the form

$$[\text{LFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{X}) \quad \text{or} \quad [\text{GFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{X}) \;, \tag{1}$$

where W is an n-ary predicate variable, $\mathbf{X}$ is an n-ary sequence of distinct variables, $\psi(W,\mathbf{X})$ is a (FPL) formula with all free variables contained in $\mathbf{X}$ and W appears only positively in $\psi(W,\mathbf{X})$.[11]

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

$$\psi^{(U,M),\chi}(S) \equiv \{\mathbf{x} \in U^n \mid (U, M), \chi \cup \{W \to S\} \models \psi(W,\mathbf{x})\} \;, \tag{2}$$

where $\chi \cup \{W \to S\}$ is the valuation χ extended such that the extension of W is assigned to S. If $\psi(W,\mathbf{X})$ 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 [Tarski 1955], which we denote by $\text{LFP}(\psi^{(U,M),\chi})$ and $\text{GFP}(\psi^{(U,M),\chi})$ respectively. Finally, we have that

$$(U, M), \chi \models [\text{LFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{x}) \iff \mathbf{x} \in \text{LFP}(\psi^{(U,M),\chi}) \;, \tag{3}$$

and similarly for greatest fixed point formulas. We call an FPL *sentence* (i.e., an FPL formula without free variables) *alternation-free* if it does not contain subformulas $\psi \equiv [\text{LFP } T\mathbf{X}.\varphi](\mathbf{X})$ and $\theta \equiv [\text{GFP } S\mathbf{Y}.\eta](\mathbf{Y})$ 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:

$$[\text{GFP } W\mathbf{X}.\psi] \equiv \neg[\text{LFP } W\mathbf{X}.\neg\psi[W/\neg W]] \;, \tag{4}$$

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.

[11] A formula ψ is in *negation-normal form* if the only used connectives are $\wedge$, $\vee$, and $\neg$, and $\neg$ only appears in front of atoms. Let ψ be a formula in negation-normal form. A predicate p appears then only positively in ψ if there is no $\neg p$ in ψ.

As in [Grädel 2002a], we define

$$\begin{aligned}\psi^{(U,M)} \uparrow 0 &\equiv \emptyset \\ \psi^{(U,M)} \uparrow \alpha + 1 &\equiv \psi^{(U,M)}(\psi^{(U,M)} \uparrow \alpha) \text{ for ordinals } \alpha \\ \psi^{(U,M)} \uparrow \beta &\equiv \bigcup_{\alpha<\beta} (\psi^{(U,M)} \uparrow \alpha) \text{ for limit ordinals } \beta\end{aligned}$$

Furthermore, since $\psi^{(U,M)}$ is monotone, we have that $\psi^{(U,M)} \uparrow 0 \subseteq \psi^{(U,M)} \uparrow 1 \subseteq \ldots$ and there exists a (limit) ordinal α such that $\psi^{(U,M)} \uparrow \alpha = \mathrm{LFP}(\psi^{(U,M)})$.

Example 2.3. Take the conjunction of the following formulas, i.e., the infinity axiom[12] from [Grädel and Walukiewicz 1999]:

$$\exists X, Y \cdot F(X, Y) \tag{5}$$

$$\forall X, Y \cdot (F(X, Y) \Rightarrow (\exists Z \cdot F(Y, Z))) \tag{6}$$

$$\forall X, Y \cdot F(X, Y) \Rightarrow [\mathrm{LFP}\ WX.\forall Y \cdot F(Y, X) \Rightarrow W(Y)](X) \tag{7}$$

A model of these formulas contains at least one $F(x, y)$ (by formula (5)), which then leads to a F-chain by formula (6). Formula (7) ensures that each element x is on a well-founded chain (and thus formula (6) actually generates an infinite chain).

2.3 Computation Tree Logic

We introduce in this subsection the temporal logic *computation tree logic (CTL)* [Emerson 1990; Emerson and Halpern 1982; Clarke et al. 1986]. Let AP be the finite set of available proposition symbols. CTL formulas are defined as follows:

—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 $\mathsf{EX}p$, $\mathsf{E}(p\ \mathsf{U}\ q)$, $\mathsf{AX}p$, and $\mathsf{A}(p\ \mathsf{U}\ q)$ 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 in 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, S is a set of states, 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 $K, s \models q$,

—$K, s \models \mathsf{EX}p$ iff there is a $(s, t) \in R$ and $K, t \models p$,

—$K, s \models \mathsf{AX}p$ iff for all $(s, t) \in R$, $K, t \models p$,

[12] An *infinity axiom* is a formula that has only infinite models (if it has models).

—$K, s \models \mathsf{E}(p\ \mathsf{U}\ q)$ iff there exists a path π in K with $\pi_0 = s$ and $\exists k \geq 0 \cdot (K, \pi_k \models q \wedge \forall j < k \cdot K, \pi_j \models p)$,

—$K, s \models \mathsf{A}(p\ \mathsf{U}\ q)$ iff for all paths π in K with $\pi_0 = s$ we have $\exists k \geq 0 \cdot (K, \pi_k \models q \wedge \forall j < k \cdot K, \pi_j \models p)$.

Intuitively, $K, s \models \mathsf{EX}p$ ($K, s \models \mathsf{AX}p$) can be read as "there is some neXt state where p holds" ("p holds in all next states"), and $K, s \models \mathsf{E}(p\ \mathsf{U}\ q)$ ($K, s \models \mathsf{A}(p\ \mathsf{U}\ q)$) as "there is some path from s along which p holds Until q holds (and q eventually holds)" ("for all paths from s, p holds until q holds (and q eventually holds)").

Some common abbreviations for CTL formulas are $\mathsf{EF}p = \mathsf{E}(\mathit{true}\ \mathsf{U}\ p)$ (there is some path on which p will eventually hold), $\mathsf{AF}p = \mathsf{A}(\mathit{true}\ \mathsf{U}\ p)$ (p will eventually hold on all paths), $\mathsf{EG}p = \neg\mathsf{AF}\neg p$ (there is some path on which p holds globally), and $\mathsf{AG}p = \neg\mathsf{EF}\neg p$ (p holds everywhere on all paths). Furthermore, we have the standard propositional abbreviations $p \vee q = \neg(\neg p \wedge \neg q)$, $p \Rightarrow q = \neg p \vee q$, and $p \Leftrightarrow q = (p \Rightarrow q) \wedge (q \Rightarrow p)$.

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.

Example 2.4. Consider the expression of *absence of starvation* $t \Rightarrow \mathsf{AF}c$ [Clarke et al. 1986] for a process in a mutual exclusion problem[13] . The formula demands that if a process tries (t) to enter a critical region, it will eventually succeed in doing so (c) for all possible future execution paths.

We will usually represent structures by diagrams as in Figure 1, where states are nodes, transitions between nodes define R, and the labels of the nodes contain the propositions true at the corresponding states. E.g., take the structure $K = (S, R, L)$ with

—$S = \{s_0, s_1, s_2\}$,

—$R = \{(s_0, s_0), (s_0, s_1), (s_1, s_2), (s_2, s_0)\}$, and

—$L(s_0) = L(s_1) = t$, $L(s_2) = c$,

which is represented by Figure 1. This structure does not satisfy $t \Rightarrow \mathsf{AF}c$ at s_0 since on the path $(s_0, s_0, \ldots)$ the proposition c never holds. We have, however, $K, s_1 \models t \Rightarrow \mathsf{AF}c$: t holds at s_1 such that we must have that on all paths from s_1 the proposition c must eventually hold; since the only path from s_1 leads to s_2 where c holds, $t \Rightarrow \mathsf{AF}c$ holds at s_1. We also have $K, s_2 \models t \Rightarrow \mathsf{AF}c$, since $t \notin L(s_2)$.

THEOREM 2.5 [EMERSON 1990]. *The problem of testing satisfiability for CTL is complete for deterministic exponential time.*

[13]In the mutual exclusion problem, we have two or more processes that want to access a critical section of code, but cannot do this at the same time. The problem is then how to model the behavior of the processes (or the concurrent program in general), such that this *mutual exclusion* is never violated. For more details, we refer to, e.g., [Emerson and Clarke 1982; Emerson 1990; Clarke et al. 1986; Attie and Emerson 2001; Huth and Ryan 2000; Manna and Wolper 1984].

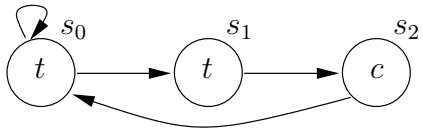

Fig. 1. Example Structure $t \Rightarrow \mathsf{AF}c$

3. OPEN ANSWER SET PROGRAMMING VIA FIXED POINT LOGIC

In this section, we will show how the external manipulations to define the (open) answer set semantics can be compiled into fixed point logic, so allowing in the next sections to analyse the decidability of OASP via the various decidability results of fragments of FPL.

We assume, without loss of generality, that the predicates in a program P are differently named than the constants in P and that each predicate q in P has one associated arity, e.g., $q(x)$ and $q(x, y)$ are not allowed.

Definition 3.1. A program P is a *p-program* if the only predicate in P different from the (in)equality predicate is p.

For a program P, let $in(Y) \equiv \{Y \neq a \mid a \in preds(P) \cup \{0\}\}$, 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 $\mathbf{Y}$, we have $in(\mathbf{Y}) \equiv \cup_{Y \in \mathbf{Y}} in(Y)$.

For a predicate name p not appearing in an arbitrary program P, we can rewrite P as an equivalent p-program P_p by replacing every regular m-ary atom $q(\mathbf{t})$ in P by $p(\mathbf{t}, \mathbf{0}, q)$ where p has arity n, with n the maximum of the arities of predicates in P augmented by 1, $\mathbf{0}$ is a sequence of new constants 0 of length $n-m-1$, and q is a new constant with the same name as the original predicate. Furthermore, in order to avoid grounding with 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. The rule in P_p corresponding to $r : \alpha \leftarrow \beta \in P$ is denoted as $r_p : \alpha_p \leftarrow \beta_p, in(\mathbf{X}) \in P_p$ for $vars(r) = \mathbf{X}$.

Example 3.2. Take a program P:

$$\begin{aligned} h(a,b) &\leftarrow q(X) \\ q(X) \vee not\ q(X) &\leftarrow \\ &\leftarrow q(a) \\ &\leftarrow q(b) \end{aligned}$$

For a universe $U = \{x, a, b\}$ of P, we have the open answer sets $M_1 = (U, \emptyset)$ and $M_2 = (U, \{q(x), h(a,b)\})$. The translation P_p is

$$\begin{aligned} p(a,b,h) &\leftarrow p(X,0,q), X \neq 0, X \neq h, X \neq q \\ p(X,0,q) \vee not\ p(X,0,q) &\leftarrow \\ &\leftarrow p(a,0,q) \\ &\leftarrow p(b,0,q) \end{aligned}$$

The open answer sets of this program can then be rewritten as open answer sets of the original program (by leaving out all "wrong" literals $p(q,0,q), p(0,0,q), p(h,0,q)$ that can be generated by the free rule).

THEOREM 3.3. *Let P be a program, p a predicate not in P, 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 $p(\boldsymbol{x}, \boldsymbol{0}, q) \in M'$.*

PROOF. For the "only if" direction, assume (U, M) is an open answer set of P that satisfies q, i.e., there is a $q(\mathbf{x}) \in M$. Let $U' = U \cup \mathit{preds}(P) \cup \{0\}$ and $M' = \{p(\mathbf{x}, \mathbf{0}, q) \mid q(\mathbf{x}) \in M\})$. Then (U', M') is an open interpretation of P_p and $p(\mathbf{x}, \mathbf{0}, q) \in M'$. One can show that (U', M') is an open answer set of P_p.

For the "if" direction, assume (U', M') is an open answer set of P_p with $p(\mathbf{x}, \mathbf{0}, q) \in M'$. Define $U \equiv U' \setminus (\mathit{preds}(P) \cup \{0\})$ and $M \equiv \{q(\mathbf{x}) \mid p(\mathbf{x}, \mathbf{0}, q) \in M' \wedge \mathbf{x} \cap (\mathit{preds}(P) \cup \{0\}) = \emptyset\}$.

We can assume that q is a non-free predicate (and we assume this throughout the rest of this paper). Then there are no free rules with a $q(\mathbf{t})$ in the head such that there are no free rules with a $p(\mathbf{t}, \mathbf{0}, q)$ in the head in P_p. Since there is a $p(\mathbf{x}, \mathbf{0}, q) \in M'$, and (U', M') is an open answer set, there must be a rule $r[]$ in $(P_p)^{M'}_{U'}$[14] such that $M' \models \mathit{in}(\mathbf{Y})[]$ for $\mathbf{Y}$ the variables in the corresponding ungrounded rule r. Thus $\mathbf{x} \cap (\mathit{preds}(P) \cup \{0\}) = \emptyset$, such that $q(\mathbf{x}) \in M$, by definition of M.

One can show that (U, M) is an open answer set of P. □

The translation of a program to a p-program does not influence the complexity of reasoning.

THEOREM 3.4. *Let P be a program and p a predicate not in P. The size[15] of P_p is polynomial in the size of P.*

PROOF. The size of a rule $r \in P$ is of the order $v + k$, with v the number of variables and k the number of predicate names in r. The corresponding r_p then contains an extra $v \times n$ inequality atoms for $n \equiv |\mathit{preds}(P) \cup \{0\}|$, and the size of r_p is thus in general quadratic in the size of r. □

By Theorems 3.3 and 3.4, we can focus, without loss of generality, 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 program as we will show in Theorem 3.8.

In [Chandra and Harel 1982], a similar motivation drives the reduction of Horn clauses[16] 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, [Chandra and Harel 1982] needs an additional transformation that unfolds bodies of clauses.

[14] For objects o (rules, (sets of) literals, ...), we denote with $o[Y_1|y_1, \ldots, Y_d|y_d]$, the grounding of o where each variable Y_i is substituted with y_i. Equivalently, we may write $o[\mathbf{Y}|\mathbf{y}]$ for $\mathbf{Y} = Y_1, \ldots, Y_d$ and $\mathbf{y} = y_1, \ldots, y_d$, or $o[]$ if the grounding substitution is clear from the context, or if it does not matter what the substitution exactly looks like.

[15] In the rest of the paper we use $n \times s$ for the size of a program P, where n is the number of rules in P and s is the maximum size of the rules in P.

[16] Horn clauses are rules of the form $a \leftarrow \beta$ where β is a finite set of atoms (i.e., negation as failure is not allowed).

We can reduce a p-program P to equivalent formulas $\texttt{comp}(P)$ in fixed point logic. The *completion* $\texttt{comp}(P)$ of a program P consists of formulas that demand that different constants in P are interpreted as different elements:

$$a \neq b \tag{8}$$

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:

$$\exists X \cdot \mathbf{true}\ . \tag{9}$$

Besides these technical requirements matching 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

—$\texttt{fpf}(P)$ 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, $\bigwedge X = \mathbf{true}$ and $\bigvee X = \mathbf{false}$. In the following, we assume that the arity of p, the only predicate in a p-program is n.

Definition 3.5. Let P be a p-program. The *fixed point translation* of P is $\texttt{fix}(P) \equiv \texttt{sat}(P) \cup \texttt{gl}(P) \cup \texttt{fpf}(P)$, where

(1) $\texttt{sat}(P)$ contains formulas

$$\forall \mathbf{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{10}$$

for rules $\alpha \leftarrow \beta \in P$ with variables $\mathbf{Y}$,

(2) $\texttt{gl}(P)$ contains the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{11}$$

for rules $r : \alpha \leftarrow \beta \in P$[17] with variables $\mathbf{Y}$,

(3) $\texttt{fpf}(P)$ contains the formula

$$\forall \mathbf{X} \cdot p(\mathbf{X}) \Rightarrow [\text{LFP}\ \ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{X}) \tag{12}$$

with

$$\phi(W, \mathbf{X}) \equiv W(\mathbf{X}) \vee \bigvee_{r:p(\mathbf{t})\vee\alpha\leftarrow\beta\in P} E(r) \tag{13}$$

and

$$E(r) \equiv \exists \mathbf{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p|W] \wedge r(\mathbf{Y}) \tag{14}$$

where $\mathbf{X} = X_1, \ldots, X_n$ are n new variables, $\mathbf{Y}$ are the variables in r, W is a new (second-order) variable and $\beta^+[p|W]$ is β^+ with p replaced by W.

[17] We assume that rules are uniquely named.

The *completion* of P is $\texttt{comp}(P) \equiv \texttt{fix}(P) \cup \{(8), (9)\}$.

The predicate W appears only positively in $\phi(W, \mathbf{X})$ such that the fixed point formula in (12) is well-defined. By the first disjunct in (13), we have that applying the operator $\phi^{(U,M)}$ (see pp. 9) to an arbitrary set $S \subseteq U^n$ does not lose information from S.

THEOREM 3.6. *Let P be a p-program and (U, M) an interpretation with $S \subseteq U^n$. Then*

$$S \subseteq \phi^{(U,M)}(S) \ .$$

PROOF. Take $\mathbf{x} \in S$, then $(U, M), W \to S \models W(\mathbf{x})$, such that, by (13), we have $(U, M), W \to S \models \phi(W, \mathbf{x})$. Thus, by (2), we have that $\mathbf{x} \in \phi^{(U,M)}(S)$. □

Example 3.7. Take a p-program P

$$r: \ p(X) \ \leftarrow \ p(X)$$

The completion $\texttt{comp}(P)$ contains the formulas $\exists X \cdot \mathbf{true}$, together with $\texttt{fix}(P) \equiv \texttt{sat}(P) \cup \texttt{gl}(P) \cup \texttt{fpf}(P)$, where

$$\texttt{sat}(P) = \{\forall X \cdot p(X) \Rightarrow p(X)\} \ ,$$

ensuring that r is satisfied, and

$$\texttt{gl}(P) = \{\forall X \cdot r(X) \Leftrightarrow \mathbf{true}\} \ ,$$

saying that r belongs to every GL-reduct since there are no naf-atoms. Finally,

$$\texttt{fpf}(P) = \{\forall X_1 \cdot p(X_1) \Rightarrow [\text{LFP} \ \ W X_1 . \phi(W, X_1)](X_1)\} \ ,$$

with

$$\phi(W, X_1) \equiv W(X_1) \vee \exists X \cdot X_1 = X \wedge W(X) \wedge r(X) \ .$$

The formula $\texttt{fpf}(P)$ ensures that every atom in a FOL interpretation is motivated by a fixed point construction, using the available rule $p(X) \leftarrow p(X)$.

THEOREM 3.8. *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 $\bigwedge \texttt{comp}(P)$, where*

$$R \equiv \{r(\boldsymbol{y}) \mid r[\boldsymbol{Y} \mid \boldsymbol{y}] : \alpha[] \leftarrow \beta[] \in P_U, M \models \alpha[]^- \cup \mathit{not}\ \beta[]^-, \mathit{vars}(r) = \boldsymbol{Y}\} \ .$$

PROOF. Denote $M \cup R$ as M'.
$\boxed{\Rightarrow}$ For the "only if" direction, assume (U, M) is an open answer set of P. We show that (U, M') is a model of $\texttt{comp}(P)$. It is not too difficult to show that (U, M') is a model of (8), (9), $\texttt{sat}(P)$, and $\texttt{gl}(P)$. We also have that (U, M') is a model of $\texttt{fpf}(P)$. Indeed, take $\mathbf{x}$ for $\mathbf{X}$ and assume $p(\mathbf{x}) \in M'$. Thus, $p(\mathbf{x}) \in M$. Since (U, M) is an open answer set we have that $p(\mathbf{x}) \in T^n$ for some $n < \infty$.

Claim 3.9. $\mathbf{x} \in \phi^{(U,M')} \uparrow n$, $n < \infty$.

We prove the claim by induction on n.

$n = 1$ *(Base step).* If $p(\mathbf{x}) \in T^1$ there is some $r' : p(\mathbf{x}) \leftarrow \beta^+[] \in P_U^M$ originating from $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ with variables $\mathbf{Y} = Y_1, \ldots, Y_d$ such that for $[\mathbf{Y}|\mathbf{y}]$, $r[] = r'$ (and thus $t_i[] = x_i$ for $1 \leq i \leq n$). Furthermore, we have

—$\emptyset \models \beta^+[]$[18],
—$M \models \alpha^-[]$, and
—$M \models \mathit{not}\ \beta^-[]$.
Thus $\bigwedge \alpha^-[]$ and $\bigwedge \neg\beta^-[]$ are true in M', such that, by definition of M', $r(\mathbf{y}) \in M'$. It follows immediately that $E(r)$ is true in M'. Since $\emptyset \models \beta^+[]$ we do not use W to deduce the latter, such that $(U, M'), W \to \emptyset \models \phi(W, \mathbf{x})$, and thus $\mathbf{x} \in \phi^{(U,M')}(\emptyset) = \phi^{(U,M')} \uparrow 1$.

(Induction). Assume for every $p(\mathbf{u}) \in T^{n-1}$ that $\mathbf{u} \in \phi^{(U,M')} \uparrow n-1$, $n-1 < \infty$. From $p(\mathbf{x}) \in T^n$, we have some $r' : p(\mathbf{x}) \leftarrow \beta^+[\mathbf{Y}|\mathbf{y}] \in P_U^M$ originating from $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ with variables $\mathbf{Y} = Y_1, \ldots, Y_d$ and such that for $[\mathbf{Y}|\mathbf{y}]$, $r[] = r'$ (and thus $t_i[] = x_i$ for $1 \leq i \leq n$). Furthermore, we have
—$T^{n-1} \models \beta^+[]$,
—$M \models \alpha^-[]$, and
—$M \models \mathit{not}\ \beta^-[]$.
Thus $\bigwedge \alpha^-[]$ and $\bigwedge \neg\beta^-[]$ are true in M', such that, by definition of M', $r(\mathbf{y}) \in M'$. Since P is a p-program β contains only p-literals and (in)equalities. Furthermore, the equalities in $\beta^+[]$ are true in M'. For every regular $p(\mathbf{u}) \in \beta^+[]$, we have that $p(\mathbf{u}) \in T^{n-1}$, and thus, by induction, that $\mathbf{u} \in \phi^{(U,M')} \uparrow n-1$. We have that $(U, M'), W \to \phi^{(U,M')} \uparrow n-1 \models E(r)[\mathbf{X}|\mathbf{x}]$, such that $(U, M'), W \to \phi^{(U,M')} \uparrow n-1 \models \phi(W, \mathbf{x})$. Thus $\mathbf{x} \in \phi^{(U,M')} \uparrow n$.

From $\mathbf{x} \in \phi^{(U,M')} \uparrow n$, $n < \infty$, we have that $\mathbf{x} \in \phi^{(U,M')} \uparrow n \subseteq \phi^{(U,M')} \uparrow \alpha$, for a limit ordinal α such that $\phi^{(U,M')} \uparrow \alpha = \mathrm{LFP}(\phi^{(U,M')})$. Then, we have that $\mathbf{x} \in \mathrm{LFP}(\phi^{(U,M')})$, and consequently, $[\mathrm{LFP}\ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{x})$ is true in (U, M') such that (12) is satisfied.

$\boxed{\Leftarrow}$ For the "if" direction, assume (U, M') is a model of $\mathtt{comp}(P)$. We show that (U, M) is an open answer set of P. Denote $\{\mathbf{x} \mid p(\mathbf{x}) \in M\}$ as $\overline{M}$.

(1) From (8) and (9), we have that U is non-empty and interprets different constants as different elements. We assume that the elements that interpret the constants in U have the same name as those constants.
(2) $\overline{M} = \mathrm{LFP}(\phi^{(U,M')})$.
—$\overline{M} = \phi^{(U,M')}(\overline{M})$.
—$\overline{M} \subseteq \phi^{(U,M')}(\overline{M})$. Immediate, with Theorem 3.6.
—$\overline{M} \supseteq \phi^{(U,M')}(\overline{M})$. Assume $\mathbf{x} \in \phi^{(U,M')}(\overline{M})$. Then by (2), we have that $(U, M'), W \to \overline{M} \models \phi(W, \mathbf{x})$. Thus, by (13), we have either that $\mathbf{x} \in \overline{M}$, which means we are done, or there is a $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ such that $(U, M'), W \to \overline{M} \models E(r)[\mathbf{X}|\mathbf{x}]$.
Then, there exist $[\mathbf{Y}|\mathbf{y}]$ with
—$\mathbf{x} = \mathbf{t}[]$,
—$(U, M'), W \to \overline{M} \models \beta^+[p|W][]$, such that $M' \models \beta^+[]$, and
—$r(\mathbf{y}) \in M'$, from which, since M' is a model of $\mathtt{gl}(P)$, we have that $M' \models \bigwedge \alpha^-[]$ and $M' \models \bigwedge \neg\beta^-[]$.
Since M' is a model of $\mathtt{sat}(P)$ we then have that $p(\mathbf{t})[] \in M'$ and thus $p(\mathbf{x}) \in M$, such that $\mathbf{x} \in \overline{M}$.

[18] β^+ may contain equalities but no regular atoms.

—$\overline{M}$ is a least fixed point. Assume there is a $Y \subseteq U^n$ such that $Y = \phi^{(U,M')}(Y)$. We prove that $\overline{M} \subseteq Y$. Take $\mathbf{x} \in \overline{M}$, then $p(\mathbf{x}) \in M'$. Since M' is a model of $\mathtt{fpf}(P)$, we have that $\mathbf{x} \in \mathrm{LFP}(\phi^{(U,M')})$. And since $\mathrm{LFP}(\phi^{(U,M')}) \subseteq Y$, we have that $\mathbf{x} \in Y$.

(3) M is a model of P_U^M. Take a rule $r' : p(\mathbf{x}) \leftarrow \beta^+[\mathbf{Y}|y] \in P_U^M$ originating from $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ with variables $\mathbf{Y} = Y_1, \ldots, Y_d$ and such that for $[\mathbf{Y}|\mathbf{y}]$, $r[] = r'$ (and thus $t_i[] = x_i$ for $1 \leq i \leq n$). Furthermore, we have
—$M \models \alpha^-[]$,
—$M \models \mathit{not}\ \beta^-[]$.
Assume $M \models \beta^+[]$, we then have that
—$M' \models \alpha^-[]$,
—$M' \models \mathit{not}\ \beta^-[]$,
—$M' \models \beta^+[]$.
Since M' is a model of $\mathtt{sat}(P)$, we then have that $p(\mathbf{x}) \in M'$, and thus $p(\mathbf{x}) \in M$.

(4) M is a minimal model of P_U^M. Assume not, then there is a $N \subset M$, N a model of P_U^M. Take $\overline{N} = \{\mathbf{x} \mid p(\mathbf{x}) \in N\}$, one can then show that $\overline{N}$ is a fixed point of $\phi^{(U,M')}$, i.e., $\overline{N} = \phi^{(U,M')}(\overline{N})$. Since $\overline{M} = \mathrm{LFP}(\phi^{(U,M')})$, we have that $\overline{M} \subseteq \overline{N}$, which is a contradiction with $N \subset M$, and M is indeed a minimal model of P_U^M.

□

Example 3.10. For a universe $U = \{x\}$ we have the unique open answer set $(U, \emptyset)$ of P in Example 3.7. Since U is non-empty, every open answer set with a universe U satisfies $\exists X \cdot \mathbf{true}$. Both $(U, M_1 = \{p(x), r(x)\})$ and $(U, M_2 = \{r(x)\})$ satisfy $\mathtt{sat}(P) \cup \mathtt{gl}(P)$. However, $\mathrm{LFP}(\phi^{(U,M_1)}) = \mathrm{LFP}(\phi^{(U,M_2)}) = \emptyset$, such that only (U, M_2) satisfies $\mathtt{fpf}(P)$; (U, M_2) corresponds exactly to the open answer set $(U, \emptyset)$ of P.

The completion in Definition 3.5 differs from Clark's completion [Clark 1987] both in the presence of the fixed point construct in (12) and atoms representing membership of the GL-reduct. For p-programs P Clark's Completion $\mathtt{ccomp}(P)$ does not contain $\mathtt{gl}(P)$ and $\mathtt{fpf}(P)$ is replaced by a formula that ensures support for every atom by an applied rule

$$\forall \mathbf{X} \cdot p(\mathbf{X}) \Rightarrow \bigvee_{r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P} D(r)$$

with

$$D(r) \equiv \exists \mathbf{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta \wedge \bigwedge \alpha^- \ .$$

Program P in Example 3.7 is the open ASP version of the classical example $p \leftarrow p$ [Lee and Lifschitz 2003]. There are FOL models of $\mathtt{ccomp}(P)$ that do not correspond to any open answer sets: both $(\{x\}, \{p(x)\})$ and $(\{x\}, \emptyset)$ are FOL models while only the latter is an open answer set of P. The next example shows the translation to FPL in detail.

Example 3.11. Take the p-program P corresponding to the program consisting of the rules $a \leftarrow not\ b$ and $b \leftarrow not\ a$, i.e.

$$\begin{aligned} r_1 &: p(X,a) \leftarrow not\ p(X,b), X \neq a, X \neq b \\ r_2 &: p(X,b) \leftarrow not\ p(X,a), X \neq a, X \neq b \end{aligned}$$

which has, for a universe $U = \{x, a, b\}$, two open answer sets $M_1 = \{p(x,a)\}$ and $M_2 = \{p(x,b)\}$. $\mathtt{sat}(P)$ contains the formulas

$$\forall X \cdot \neg p(X,b) \wedge X \neq a \wedge X \neq b \Rightarrow p(X,a)\ ,$$

and

$$\forall X \cdot \neg p(X,a) \wedge X \neq a \wedge X \neq b \Rightarrow p(X,b)\ .$$

$\mathtt{gl}(P)$ is defined by the formulas $\forall X \cdot r_1(X) \Leftrightarrow \neg p(X,b) \wedge X \neq a \wedge X \neq b$ and $\forall X \cdot r_2(X) \Leftrightarrow \neg p(X,a) \wedge X \neq a \wedge X \neq b$. Finally, $\mathtt{fpf}(P)$ is

$$\forall X_1, X_2 \cdot p(X_1, X_2) \Rightarrow [\text{LFP}\ \ W X_1, X_2 . \phi(W, X_1, X_2)](X_1, X_2)$$

with

$$\begin{aligned} \phi(W, X_1, X_2) \equiv\ & W(X_1, X_2) \\ & \vee \exists X \cdot X_1 = X \wedge X_2 = a \wedge r_1(X) \\ & \vee \exists X \cdot X_1 = X \wedge X_2 = b \wedge r_2(X)\ . \end{aligned}$$

To satisfy $\mathtt{sat}(P)$ a model must contain $p(x,a)$ or $p(x,b)$. Taking into account $\mathtt{gl}(P)$, we then distinguish three different classes of models, represented by

$$\begin{aligned} M_1' &\models \{p(x,a), \neg p(x,b), r_1(x), \neg r_2(x)\}\ , \\ M_2' &\models \{\neg p(x,a), p(x,b), \neg r_1(x), r_2(x)\}\ , \\ M_3' &\models \{p(x,a), p(x,b), \neg r_1(x), \neg r_2(x)\}\ . \end{aligned}$$

Now, we have that $\text{LFP}(\phi^{(U,M_3')}) = \emptyset$, such that $\mathtt{fpf}(P)$ is not satisfied by M_3'. Furthermore, $\text{LFP}(\phi^{(U,M_1')}) = \{(x,a)\}$ and $\text{LFP}(\phi^{(U,M_2')}) = \{(x,b)\}$. Thus, in order to satisfy $\mathtt{fpf}(P)$, we have that $M_1' = \{p(x,a), r_1(x)\}$ and $M_2' = \{p(x,b), r_2(x)\}$, which correspond to the open answer sets of P.

Note that this example also shows that writing knowledge down in Logic Programming style is easier and more intuitive than the corresponding FPL translation.

THEOREM 3.12. *Let P be a p-program. The size of $\bigwedge \mathtt{comp}(P)$ is quadratic in the size of P.*

PROOF. If the number of constants in a program P is c, then the number of formulas (8) is $\frac{1}{2}c(c-1)$, which yields the quadratic bound. The size of $\mathtt{sat}(P)$ is linear in the size of P, as is the size of $\mathtt{gl}(P)$ (with $|P|$ new predicates). Finally, each $E(r)$ in $\mathtt{fpf}(P)$ is linear in the size of r, such that $\mathtt{fpf}(P)$ is linear in the size of P. □

THEOREM 3.13. *Let P be a program, p a predicate not appearing in P, and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \bigwedge \mathtt{comp}(P_p)$ is satisfiable. Moreover, this reduction is polynomial in the size of P.*

PROOF. Assume q is satisfiable w.r.t. P. By Theorem 3.3, we have that $p(\mathbf{x}, \mathbf{0}, q)$ is in an open answer set of P_p, such that, with Theorem 3.8, $p(\mathbf{x}, \mathbf{0}, q)$ is in a model of $\mathtt{comp}(P_p)$.

For the opposite direction, assume $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{comp}(P_p)$ is satisfiable. Then there is a model (U, M') of $\bigwedge \mathtt{comp}(P)$ with $p(\mathbf{x}, \mathbf{0}, q) \in M'$. We have that $M' = M \cup R$ as in Theorem 3.8, such that (U, M) is an open answer set of P_p and $p(\mathbf{x}, \mathbf{0}, q) \in M$. From Theorem 3.3, we then have that q is satisfiable w.r.t. P.

By Theorem 3.12, the size of $\bigwedge \mathtt{comp}(P_p)$ is quadratic in the size of P_p. Since the size of the latter is polynomial in the size of P by Theorem 3.4, the size of $\bigwedge \mathtt{comp}(P_p)$ is polynomial in the size of P. □

4. GUARDED OPEN ANSWER SET PROGRAMMING

In this section, we will identify a syntactically restricted class of programs such that the translation to FPL falls within a decidable fragment of FPL and which enables us to devise some complexity result for satisfiability checking. Intuitively, rules will be equipped with a guard, i.e. a set of atoms, in the positive body, such that every pair of variables in the rule appears together in an atom in that guard.

We repeat the definitions of the *loosely guarded fragment* [Van Benthem 1997] of first-order logic as in [Grädel and Walukiewicz 1999]: *The* loosely guarded fragment LGF *of first-order logic 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* $\psi(\boldsymbol{X}, \boldsymbol{Y})$[19] *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* $\mathit{free}(\psi) \subseteq \mathit{free}(\alpha) = \boldsymbol{X} \cup \boldsymbol{Y}$ *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* $\mathit{free}(\psi)$ *are the free variables of* ψ*).*

The *loosely guarded fixed point logic* μLGF is LGF extended with fixed point formulas (1) where $\psi(W, \mathbf{X})$ is a μLGF formula such that W does not appear in guards. The *guarded fragment* GF is defined as LGF but with the guards $\alpha(\mathbf{X}, \mathbf{Y})$ atoms instead of a conjunction of atoms. The *guarded fixed point logic* μGF is GF extended with fixed point formulas where $\psi(W, \mathbf{X})$ is a μGF formula such that W does not appear in guards.

Example 4.1. The infinity axiom in Example 2.3 (pp. 10) is a μGF formula where all the formulas are guarded by $F(X, Y)$.

Example 4.2 *[Grädel and Walukiewicz 1999].* Take the formula

$$\exists Y \cdot X \leq Y \wedge \varphi(Y) \wedge (\forall Z \cdot (X \leq Z \wedge Z < Y) \Rightarrow \psi(Z)) \ .$$

[19] Recall that $\psi(\mathbf{X}, \mathbf{Y})$ denotes a formula whose free variables are all among $\mathbf{X} \cup \mathbf{Y}$ ([Andréka et al. 1998], pp. 236).

This formula is not guarded as the formula $\forall Z \cdot (X \leq Z \wedge Z < Y) \Rightarrow \psi(Z)$ has no atom as guard. It is however loosely guarded.

Definition 4.3. 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 4.4. The rule in Example 3.7 is loosely guarded but not fully loosely guarded. The program in Example 3.11 is neither fully loosely guarded nor loosely guarded. A rule

$$a(X) \vee \mathit{not}\ g(X, Y, Z) \leftarrow \mathit{not}\ b(X, Y), f(X, Y), f(X, Z), h(Y, Z), \mathit{not}\ c(Y)$$

has a body guard $\{f(X,Y), f(X,Z), h(Y,Z)\}$ and a head guard $\{g(X,Y,Z)\}$.

Definition 4.5. A rule $r : \alpha \leftarrow \beta$ is *guarded* if it is loosely guarded with a singleton body guard. It is *fully guarded* if it is fully loosely guarded with body and head guards singleton sets.

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

In [Grädel et al. 2002] it is noted that a singleton set $\{b\} \subseteq U$ for a universe U is always guarded by an atom $b = b$. With a similar reasoning one sees that rules with only one variable X can be made guarded by adding the guard $X = X$ to the body. E.g., $a(X) \leftarrow \mathit{not}\ b(X)$ is equivalent to $a(X) \leftarrow X = X, \mathit{not}\ b(X)$.

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

Example 4.6. The rule $p(X) \leftarrow p(X)$ can be rewritten as $p(X) \vee \mathit{not}\ p(X) \leftarrow p(X)$ 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 $\alpha \cup \mathit{not}\ \beta^+ \leftarrow \beta$.

One can consider the body guard of a rule in a loosely guarded program P as the head guard such that P^{f} is indeed a fully (loosely) guarded program.

THEOREM 4.7. *Let P a (L)GP. Then, P^{f} is a F(L)GP.*

PROOF. Let P be a (L)GP. We show that every non-free rule $r : \alpha \cup \mathit{not}\ \beta^+ \leftarrow \beta \in P^{\mathrm{f}}$ is fully (loosely) guarded. Since $\alpha \leftarrow \beta$ is a non-free rule of P, we have that there is a body guard $\gamma_b \subseteq \beta^+$, and thus r is (loosely) guarded. Furthermore, $\gamma_b \subseteq (\alpha \cup \mathit{not}\ \beta^+)^-$ such that γ_b is a head guard of r and r is fully (loosely) guarded. □

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.

THEOREM 4.8. *Let P be any program (not necessarily guarded). 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 only copy (a part of) the bodies to the heads, the size of P^{f} only increases linearly in the size of P.

THEOREM 4.9. *Let P be any program (not necessarily guarded). The size of P^{f} is linear in the size of P.*

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

THEOREM 4.10. *Let P be any program (not necessarily guarded). Then, P is a (F)LGP iff P_p is a (F)LGP. And similarly for (F)GPs.*

PROOF. We only prove the LGP case, the cases for FLGPs and (F)GPs are similar.

For the "only if" direction, take a non-free rule $r_p : \alpha_p \leftarrow \beta_p, in(\mathbf{X}) \in P_p$ and two variables X and Y in r_p. We have that $r : \alpha \leftarrow \beta$ is a non-free rule in P by the construction of P_p and X and Y are two variables in r, such that there is a $\gamma \subseteq \beta^+$ with either a regular atom $q(\mathbf{t})$ that contains X and Y or an equality atom $X = Y$ in γ. In the former case, we have that $p(\mathbf{t}, \mathbf{0}, q) \in \gamma_p \subseteq {\beta_p}^+$ such that r_p is loosely guarded. In the latter case, $X = Y \in \gamma_p$ such that again r_p is loosely guarded.

For the "if" direction, take a non-free $r : \alpha \leftarrow \beta \in P$ and two variables X and Y in r. Then $r_p : \alpha_p \leftarrow \beta_p, in(\mathbf{X})$ is non-free in P_p and X and Y are variables in r_p. Thus, there is a $\gamma_p \subseteq (\beta_p \cup in(\mathbf{X}))^+ = {\beta_p}^+$ with an atom containing the two variables X and Y. Then $\gamma \subseteq \beta^+$ with an atom in γ containing X and Y. □

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

—Formula (9) is replaced by

$$\exists X \cdot X = X \ , \tag{15}$$

such that it is guarded by $X = X$.

—Formula (10) is removed if $r : \alpha \leftarrow \beta$ is free or otherwise replaced by

$$\forall \mathbf{Y} \cdot \bigwedge \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \gamma_b) \vee \bigvee \beta^- \ , \tag{16}$$

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

—Formula (11) is replaced by the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{17}$$

and

$$\forall \mathbf{Y} \cdot \bigwedge \gamma_h \Rightarrow r(\mathbf{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \setminus \gamma_h) \tag{18}$$

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 $r(\mathbf{Y})$ 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 (12), replace $E(r)$ by

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

with $\mathbf{Z} = \mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\}$, 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 $\bigwedge \beta^+[p|W] \wedge r(\mathbf{Y})$ by X_i. This rewriting makes sure that every variable in the quantified part of $E'(R)$ is guarded by $r(\mathbf{Y})[t_i \in \mathbf{Y}|X_i]$.

Example 4.11. 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 $\{p(X)\}$, $\mathtt{sat}(P) = \{\forall X \cdot p(X) \Rightarrow p(X) \vee \neg p(X)\}$, $\mathtt{gl}(P) = \{\forall X \cdot r(X) \Leftrightarrow p(X)\}$ and the formula $\phi(W, X_1)$ in $\mathtt{fpf}(P)$ is $\phi(W, X_1) \equiv W(X_1) \vee \exists X \cdot X_1 = X \wedge W(X) \wedge r(X)$. $\mathtt{gcomp}(P)$ translates $\mathtt{sat}(P)$ identically and rewrites the equivalence of $\mathtt{gl}(P)$ as two implications resulting in guarded rules. The rewritten $\phi(W, X_1)$ is $W(X_1) \vee (W(X_1) \wedge r(X_1))$. There is no quantification anymore in this formula since X was substituted by X_1. Clearly, for a universe $\{x\}$, we have that the open answer set of the program is $(\{x\}, \emptyset)$, which corresponds with the unique model of $\mathtt{gcomp}(P)$ for a universe $\{x\}$.

The translation $\mathtt{gcomp}(P)$ is logically equivalent to $\mathtt{comp}(P)$ and, moreover, it contains only formulas in (loosely) guarded fixed point logic.

THEOREM 4.12. *Let P be a fully (loosely) guarded p-program. (U, M) is a model of $\bigwedge \mathtt{comp}(P)$ iff (U, M) is a model of $\bigwedge \mathtt{gcomp}(P)$.*

PROOF. This can be shown by using standard logical equivalences. □

THEOREM 4.13. *Let P be a fully (loosely) guarded p-program. Then, the formula $\bigwedge \mathtt{gcomp}(P)$ is a μ(L)GF formula.*

PROOF. We first show that $[\mathrm{LFP}\ W\mathbf{X}.\phi'(W, \mathbf{X})](\mathbf{X})$ is a valid fixed point formula, with $\phi'(W, \mathbf{X})$ equal to $\phi(W, \mathbf{X})$ with $E'(r)$ instead of $E(r)$. We have that all free variables are still in $\mathbf{X}$, since only $X_i = t_i$ where t_i is constant is moved out of the scope of the quantifier in $E(r)$ and all other t_i where substituted by X_i such that $\mathbf{Z}$ in $E(r)$ bounds all other variables than $\mathbf{X}$. Furthermore, p appears only positively in ϕ'.

We next show that $\bigwedge \mathtt{gcomp}(P)$ is a μLGF formula if P is fully loosely guarded; the treatment for μGF formulas if P is fully guarded is similar.

—Formula (15) is guarded with guard $X = X$.

—Formula (16) corresponds with a non-free rule $\alpha \leftarrow \beta$ with a body guard γ_b; thus $\mathit{vars}(\alpha \leftarrow \beta) \subseteq \mathit{vars}(\gamma_b)$.
  —$\mathrm{free}(\bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \gamma_b) \vee \bigvee \beta^-) \subseteq \mathbf{Y} = \mathit{vars}(\alpha \leftarrow \beta) = \mathit{vars}(\gamma_b) = \mathrm{free}(\bigwedge \gamma_b)$.
  —Take two variables Y_i and Y_j from $\mathbf{Y}$, then $Y_i \in \mathit{vars}(\alpha \leftarrow \beta)$ and $Y_j \in \mathit{vars}(\alpha \leftarrow \beta)$, such that Y_i and Y_j are in an atom from γ_b.

—Formula (17) is guarded with guard $r(\mathbf{Y})$.

—Formula (18):
  —For a non-free rule $\alpha \leftarrow \beta$ with a head guard γ_h. Can be done similarly as formula (16).
  —If $\alpha \leftarrow \beta$ is free, i.e., of the form $q(\mathbf{t}) \vee \mathit{not}\ q(\mathbf{t}) \leftarrow$, we have that $\gamma_h = \{q(\mathbf{t})\}$, and formula (18) is of the form $\forall \mathbf{Y} \cdot q(\mathbf{t}) \Rightarrow r(\mathbf{Y})$.
    —$\text{free}(r(\mathbf{Y})) = \mathbf{Y} = \mathit{vars}(\alpha \leftarrow \beta) = \mathit{vars}(q(\mathbf{t})) = \text{free}(\bigwedge \gamma_h)$.
    —Take two variables Y_i and Y_j from $\mathbf{Y}$, then $Y_i \in \mathit{vars}(\alpha \leftarrow \beta)$ and $Y_j \in \mathit{vars}(\alpha \leftarrow \beta)$, such that Y_i and Y_j are in $\mathit{vars}(q(\mathbf{t})) = \text{free}(\gamma_h)$.
—For the last case, we need to show that $\phi'(\mathbf{X})$ is a μLGF formula where W does not appear as a guard. We show that for each $r : \alpha \leftarrow \beta$, $\exists \mathbf{Z} \cdot (\bigwedge \beta^+[p|W] \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y}|X_i]$ is a guarded formula with guard $r(\mathbf{Y})[]$. Thus W does not appear as a guard.
  —$\text{free}((\bigwedge \beta^+[p|W] \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y}|X_i]) = \mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\} \cup \{X_i \mid t_i \in \mathbf{Y}\} = \text{free}(r(\mathbf{Y})[])$.
  —Take a quantified variable $Z \in \mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\}$ and U from $\mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\} \cup \{X_i \mid t_i \in \mathbf{Y}\}$, then Z and U appear in $r(\mathbf{Y})[]$.

□

Since $\texttt{gcomp}(P)$ is just a logical rewriting of $\texttt{comp}(P)$ its size is linear in the size of $\texttt{comp}(P)$.

THEOREM 4.14. *Let P be a fully (loosely) guarded p-program. The size of the formula $\texttt{gcomp}(P)$ is linear in the size of $\texttt{comp}(P)$.*

PROOF. The size of formula (15) is linear in the size of (9). Formula (16) is just a shuffling of (10). Every formula (11) is replaced by two shuffled formulas. Finally, $E'(r)$ is $E(r)$ with the movement of some atoms and applying a substitution, thus the size of $E'(r)$ is linear in the size of $E(r)$. □

THEOREM 4.15. *Let P be a (L)GP and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \bigwedge \texttt{gcomp}((P^{\text{f}})_p)$ is satisfiable. Moreover, this reduction is polynomial in the size of P.*

PROOF. By Theorem 4.7 and 4.10, we have that $(P^{\text{f}})_p$ is a fully (loosely) guarded p-program, thus the formula $\bigwedge \texttt{gcomp}((P^{\text{f}})_p)$ is defined. By Theorem 4.8, we have that q is satisfiable w.r.t. P iff q is satisfiable w.r.t. P^{f}. By Theorem 3.13, we have that q is satisfiable w.r.t. P^{f} iff $p(\mathbf{X}, \mathbf{0}, q) \wedge \texttt{comp}((P^{\text{f}})_p)$ is satisfiable. Finally, Theorem 4.12 yields that q is satisfiable w.r.t. P iff $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \texttt{gcomp}((P^{\text{f}})_p)$ is satisfiable.

Theorem 4.9, Theorem 3.13, and Theorem 4.14 yield that this reduction is polynomial. □

For a (L)GP P, we have, by Theorem 4.13, that $\bigwedge \texttt{gcomp}((P^{\text{f}})_p)$ is a μ(L)GF formula such that the formula $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \texttt{gcomp}((P^{\text{f}})_p)$ is as well. Since satisfiability checking for μ(L)GF is 2-EXPTIME-complete (Theorem [1.1] in [Grädel and Walukiewicz 1999]), satisfiability checking w.r.t. P is in 2-EXPTIME.

THEOREM 4.16. *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 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, such that 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, add $g(a, b) \leftarrow$ for every $a, b \in cts(P)$. Note that we assume, without loss of generality, that P does not contain a predicate g.

Example 4.17. Take a program P

$$\begin{aligned} q(X) &\leftarrow f(X, Y) \\ f(a, Y) \vee \mathit{not}\ f(a, Y) &\leftarrow \end{aligned}$$

such that $cts(P) = \{a\}$, and P has answer sets $\{f(a, a), q(a)\}$ and $\emptyset$. The loosely guarded program P^{g} is

$$\begin{aligned} q(X) &\leftarrow g(X, X), g(Y, Y), g(X, Y), f(X, Y) \\ f(a, Y) \vee \mathit{not}\ f(a, Y) &\leftarrow g(Y, Y) \\ g(a, a) &\leftarrow \end{aligned}$$

For a universe U, we have the open answer sets $(U, \{f(a, a), q(a), g(a, a)\}$ and $(U, \{g(a, a)\})$.

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

THEOREM 4.18. *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}.*

Note that one can use Theorem 4.18 as a definition of *answer set* of programs with generalized literals, in case one is only interested in answer sets and not in the open answer sets.

THEOREM 4.19. *Let P be a program. The size of P^{g} is quadratic in the size of P.*

PROOF. If there are c constants in P, we add c^2 rules $g(a, b) \leftarrow$ to P^{g}. Furthermore, the size of each rule grows also grows quadratically, since for a rule with n variables we add n^2 atoms $g(X, Y)$ to the body of r. □

By construction, P^{g} is loosely guarded.

THEOREM 4.20. *Let P be a program. P^{g} is a LGP.*

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.

THEOREM 4.21. *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.*

THEOREM 4.22. *Satisfiability checking w.r.t. LGPs is* NEXPTIME*-hard.*

PROOF. By [Dantsin et al. 2001; Baral 2003] 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 $q(\mathbf{a})$ is NEXPTIME-complete. Thus, by Theorem 4.21, satisfiability checking w.r.t. a LGP is NEXPTIME-hard. □

A similar approach to show NEXPTIME-hardness of GPs instead of LGPs does not seem to be directly applicable. E.g., a naive approach is to add to the body of every rule r in a program P, an n-ary guarding atom $g(X_1, \ldots, X_k, \ldots X_k)$, $k \leq n$, with n the maximum number of different variables in rules of P and $X_1, \ldots, X_k$ the pairwise different variables in r. Furthermore, one need to enforce that for an open answer set and n constants $a_1, \ldots, a_n$, $g(a_1, \ldots, a_n)$ is in the answer set, and vice versa, if $g(x_1, \ldots, x_n)$ is in the open answer set then $x_1, \ldots, x_n \in cts(P)$. This amounts to adding c^n rules $g(a_1, \ldots, a_n) \leftarrow$ for constants $a_1, \ldots, a_n \in cts(P)$ where c is the number of constants in P. Since n is not bounded, this transformation is, however, not polynomial.

In Section 8, we improve[20] on Theorem 4.22 and show that both satisfiability checking w.r.t. GPs and w.r.t. LGPs is 2-EXPTIME-hard.

5. OPEN ANSWER SET PROGRAMMING WITH GENERALIZED LITERALS

In this section, we extend the language of logic programs with *generalized literals* and modify the open answer set semantics to accommodate for those generalized literals. As already argued in the introduction, generalized literals allow for a more robust representation of knowledge than is possible without them.

E.g., in [Balduccini and Gelfond 2003] a mapping is given from an action description into an answer set program. In this mapping, a predicate $prec\_h(D, T)$ needs to be computed, intuitively meaning that all preconditions of D hold at time T. As the authors did not have generalized literals at their disposal, they needed a ternary relation $pred(D, N, C)$ which encodes that C is the N-th precondition of D and a special predicate denoting the number of preconditions for D, i.e. an explicit linear order among the preconditions has to be established. Next, using this linear order, they had to introduce some additional ternary predicate all_h that checks if all preconditions hold and than use this predicate to compute $prec\_h(D, T)$. However, with generalized literals no linear order needs to be established to compute $prec\_h(D, T)$, i.e. it suffices to have predicates $prec(D, C)$ encoding that D is a precondition of C. Than, we can use, with $h(C, T)$ meaning that condition C holds at time T, the rule

$$prec\_h(D, T) \leftarrow [\forall C \cdot prec(D, C) \Rightarrow h(C, T)]$$

to compute the correct meaning of $prec\_h$.

[20] Note that P ⊆ NP ⊆ EXPTIME ⊆ NEXPTIME ⊆ 2-EXPTIME ⊆ ... where P ⊂ EXPTIME, EXPTIME ⊂ 2-EXPTIME, ..., and NP ⊂ NEXPTIME, NEXPTIME ⊂ 2-NEXPTIME, ..., see, e.g., [Papadimitriou 1994; Tobies 2001].

Formally, a *generalized literal* is a first-order formula of the form

$$\forall \mathbf{Y} \cdot \phi \Rightarrow \psi \ ,$$

where ϕ is a finite boolean combination 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 (i.e., atoms and naf-atoms since we assume the absence of $\neg$) and generalized literals as *g-literals*. For a set of g-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 g-literals as follows: $\alpha^+ = (\alpha \backslash \alpha^{\mathrm{x}})^+$ and $\alpha^- = (\alpha \backslash \alpha^{\mathrm{x}})^-$; thus $\alpha = \alpha^+ \cup \mathit{not}\ \alpha^- \cup \alpha^{\mathrm{x}}$.

A *generalized program (gP)* is a countable set of *rules* $\alpha \leftarrow \beta$, where α is a finite set of literals, $|\alpha^+| \leq 1$, β is a countable[21] set of g-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. Furthermore, generalized literals are ground if they do not contain free variables, and rules and gPs are ground if all g-literals in it are ground.

For a g-literal l, we define $\mathit{vars}(l)$ as the (free) variables in l. For a rule r, we define $\mathit{vars}(r) \equiv \cup\{\mathit{vars}(l) \mid l \text{ g-literal in } r\}$. For a set of atoms I, we extend the $\models$ relation for interpretations I, by induction, for any boolean formula of ground atoms. For such ground boolean formulas ϕ and ψ, we have

(1) $I \models \phi \wedge \psi$ iff $I \models \phi$ and $I \models \psi$,

(2) $I \models \phi \vee \psi$ iff $I \models \phi$ or $I \models \psi$, and

(3) $I \models \neg\phi$ iff $I \not\models \phi$.

Similarly as for programs without generalized literals, call a pair (U, I) where U is a universe for P and I a subset of $\mathcal{B}_P^U$ an *open interpretation* of P. For a ground gP P and an open interpretation (U, I) of P, we define the *GeLi-reduct* $P^{\mathrm{x}(U,I)}$ which removes the generalized literals from the program: $P^{\mathrm{x}(U,I)}$ contains the rules

$$\alpha \leftarrow \beta \backslash \beta^{\mathrm{x}}, (\beta^{\mathrm{x}})^{\mathrm{x}(U,I)} \ , \tag{20}$$

for $\alpha \leftarrow \beta$ in P, where

$$(\beta^{\mathrm{x}})^{\mathrm{x}(U,I)} \equiv \bigcup_{\forall \mathbf{Y} \cdot \phi \Rightarrow \psi \in \beta^{\mathrm{x}}} \{\psi[\mathbf{Y}|\mathbf{y}] \mid \mathbf{y} \subseteq U, I \models \phi[\mathbf{Y}|\mathbf{y}]\} \ .$$

Intuitively, a generalized literal $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ is replaced by those $\psi[\mathbf{Y}|\mathbf{y}]$ for which $\phi[\mathbf{Y}|\mathbf{y}]$ is true, such that[22], e.g., $p(a) \leftarrow [\forall \mathbf{X} \cdot q(X) \Rightarrow r(X)]$ 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 GeLi-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 GeLi-reduct. Note that $((\beta^{\mathrm{x}})^{\mathrm{x}(U,I)})^-$ is always empty by definition of generalized literals: the consequent is always an atom.

Also note that $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ cannot be seen as $\forall \mathbf{Y} \cdot \neg\phi \vee \psi$, where the forall is an abbreviation of the conjunction with respect to a given domain and where we use

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

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

an extended reduction for nested programs [Lifschitz et al. 1999]. Consider e.g. the rules

$$\begin{aligned} p(X) &\leftarrow [\forall Y \cdot \neg b(Y) \wedge \neg c(Y) \Rightarrow d(Y)] \\ b(a) &\leftarrow \end{aligned}$$

and consider the open interpretation $I = (\{a\}, \{b(a)\})$. The GeLi reduct of P w.r.t. I is (note that $I \models \neg b(a) \wedge \neg c(a)$)

$$\begin{aligned} p(a) &\leftarrow d(a) \\ b(a) &\leftarrow \end{aligned}$$

which will have I as an open answer set according to Definition 5.1 below. However, if we apply the suggested transformation to P, we get

$$\begin{aligned} p(X) &\leftarrow [\forall Y \cdot b(Y) \vee c(Y) \vee d(Y)] \\ b(a) &\leftarrow \end{aligned}$$

which would have the following "GL reduct for nested programs" wrt I:

$$\begin{aligned} p(a) &\leftarrow b(a) \vee c(a) \vee d(a) \\ b(a) &\leftarrow \end{aligned}$$

But, since $I \not\models b(a)$, the first rule becomes applicable and thus any answer set containing $b(a)$ must also contain $p(a)$. Hence I is not an answer set using this transformation.

Definition 5.1. An *open answer set* of P is an open interpretation (U, M) of P where M is an answer set of $(P_U)^{\mathrm{x}(U,M)}$.

In the following, a gP is assumed to be a finite set of finite rules; infinite gPs only appear as byproducts of grounding a finite program with an infinite universe, or, by taking the GeLi-reduct w.r.t. an infinite universe. Satisfiability checking remains defined as before.

Example 5.2. Take a gP P

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

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). Take an open interpretation $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$. Then, the GeLi-reduct of $P_{\{x,y\}}$ is

$$\begin{aligned} p(x) &\leftarrow r(x) \\ p(y) &\leftarrow r(x) \\ r(x) &\leftarrow q(x) \\ r(y) &\leftarrow q(y) \\ q(x) \vee \mathit{not}\ q(x) &\leftarrow \\ q(y) \vee \mathit{not}\ q(y) &\leftarrow \end{aligned}$$

$\{p(x), r(x), q(x), p(y)\}$ is an answer set such that the open interpretation is an open answer set.

Example 5.3. Take the following program P, i.e., the open answer set variant of the classical infinity axiom in guarded fixed point logic from [Grädel and Walukiewicz 1999] (see also Example 2.3, pp. 10), where we use *well* to denote *well_founded*:

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

Intuitively, in order to satisfy q with some x, one needs to apply r_1, which enforces an f-successor y. Moreover, 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.

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

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

Since, for r_4^0, there is no $f(y, x_0)$ in M, the body of the corresponding rule in the GeLi-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 $\mathit{well}(x_0)$ in the body:

$$\begin{array}{rcl}
\mathit{well}(x_0) & \leftarrow & q(x_0) \\
\mathit{well}(x_1) & \leftarrow & q(x_1), \mathit{well}(x_0) \\
& \vdots &
\end{array}$$

One can check that (U, M) is indeed an open answer set of the gP, 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 otherwise. By rule r_3, we need $\mathit{well}(x_0) \in M$. However, the GeLi-reduct of P_U contains rules:

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

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

We defined the open answer set semantics for gPs in function of the answer set semantics for programs without generalized literals. We can, however, also define a GL-reduct P^M directly for a ground gP P by treating generalized literals as positive, such that $\alpha^+ \leftarrow \beta^+, \beta^{\mathrm{x}} \in P^M$ iff $\alpha \leftarrow \beta \in P$ and $M \models \alpha^-$ and $M \models not\ \beta^-$ for a ground gP P. Applying the GL-reduct transformation after the GeLi-reduct transformation (like we defined it), is then equivalent to first applying the GL-reduct transformation to a gP and subsequently computing the GeLi-reduct.

Example 5.4. Take a program $F \cup \{r\}$ with $F \equiv \{q(x) \leftarrow, b(x) \leftarrow, b(y) \leftarrow, c(x) \leftarrow\}$ and $r : a(X) \leftarrow [\forall X \cdot \neg q(X) \Rightarrow b(X)], not\ c(X)$. For a universe $U = \{x, y\}$, $(F \cup \{r\})_U$ is $F \cup \{r_x, r_y\}$ where

$$r_x : a(x) \leftarrow [\forall X \cdot \neg q(X) \Rightarrow b(X)], not\ c(x)$$

and

$$r_y : a(y) \leftarrow [\forall X \cdot \neg q(X) \Rightarrow b(X)], not\ c(y)$$

Applying the GeLi-reduct transformation w.r.t.

$$(U, M = \{q(x), b(x), b(y), c(x), a(y)\})$$

yields

$$(F \cup \{r_x, r_y\})^{\mathrm{x}(U,M)} \equiv F \cup \{a(x) \leftarrow b(y), not\ c(x); a(y) \leftarrow b(y), not\ c(y)\}\ .$$

The GL-reduct of the latter is $F \cup \{a(y) \leftarrow b(y)\}$, such that (U, M) is a (unique) open answer set of $F \cup \{r\}$ for $U = \{x, y\}$.

First applying the GL-reduct transformation to $F \cup \{r_x, r_y\}$ yields $F \cup \{r_y\}$, and, subsequently, the GeLi-reduct again gives $F \cup \{a(y) \leftarrow b(y)\}$. Thus

$$((F \cup \{r_x, r_y\})^{\mathrm{x}(U,M)})^M = ((F \cup \{r_x, r_y\})^M)^{\mathrm{x}(U,M)}\ .$$

Since the GeLi-reduct transformation never removes rules or naf-atoms from rules, while the GL-reduct transformation may remove rules (and thus generalized literals), calculating the GL-reduct before the GeLi-reduct is likely to be more efficient in practice. We opted, however, for the "GeLi-reduct before GL-reduct" transformation as the standard definition, as it is theoretically more robust against changes in the definition of generalized literals. E.g., if naf were allowed in the consequent of generalized literals, the "GL-reduct before GeLi-reduct" approach does not work since the GeLi-reduct (as currently defined) could introduce naf again in the program, making another application of the GL-reduct transformation necessary.

THEOREM 5.5. *Let P be a ground gP with an open interpretation (U, M). Then,*

$$(P^{x(U,M)})^M = (P^M)^{x(U,M)}\ .$$

We have a similar result as in Theorem 2.2 regarding the finite motivation of literals in possibly infinite open answer sets. We again express the motivation

of a literal more formally by means of the *immediate consequence operator* [van Emden and Kowalski 1976] T that computes the closure of a set of literals w.r.t. a GL-reduct of a GeLi-reduct.

For a gP P and an open interpretation (U, M) of P, $T_P^{(U,M)} : \mathcal{B}_P^U \to \mathcal{B}_P^U$ is defined as $T(B) = B \cup \{a | a \leftarrow \beta \in \left(P_U^{\mathrm{x}(U,M)}\right)^M \wedge B \models \beta\}$. Additionally, we have $T^0(B) = B$[23], and $T^{n+1}(B) = T(T^n(B))$.

THEOREM 5.6. *Let P be a gP and (U, M) an open answer set of P. Then, $\forall a \in M \cdot \exists n < \infty \cdot a \in T^n$.*

Finally, the next example illustrates that their is a difference between our answer set semantics for generalized literals and the answer set semantics introduced in [Lifschitz et al. 2001b; Osorio et al. 2004; Osorio and Ortiz 2004] for propositional theories, which are based on intuistionistic logic.

Example 5.7. Consider the program

$$\begin{array}{rcl} a(X) & \leftarrow & [\forall X \cdot c(X) \Rightarrow b(X)] \\ a(X) & \leftarrow & b(X) \\ b(X) & \leftarrow & c(X) \\ c(X) & \leftarrow & a(X) \end{array}$$

and consider the open interpretation $I = (\{a\}, \{a(a), b(a), c(a)\})$. Applying the GeLi-reduct on this program w.r.t. I yields the program

$$\begin{array}{rcl} a(a) & \leftarrow & b(a) \\ a(a) & \leftarrow & b(a) \\ b(a) & \leftarrow & c(a) \\ c(a) & \leftarrow & a(a) \end{array}$$

which only has $\emptyset$ as an answer set, implying that I is not an open answer set for this program.

However, one could expect the programs

$$\begin{array}{rcl} a(a) & \leftarrow & [\forall X \cdot c(a) \Rightarrow b(a)] \\ a(a) & \leftarrow & b(a) \\ b(a) & \leftarrow & c(a) \\ c(a) & \leftarrow & a(a) \end{array}$$

and

$$\begin{array}{rcl} a(a) & \leftarrow & c(a) \Rightarrow b(a) \\ a(a) & \leftarrow & b(a) \\ b(a) & \leftarrow & c(a) \\ c(a) & \leftarrow & a(a) \end{array}$$

to be equivalent, but, in the context of [Lifschitz et al. 2001b; Osorio et al. 2004; Osorio and Ortiz 2004], we have $\{a(a), b(a), c(a)\}$ as the unique answer set for the last program, as $c(a) \Rightarrow b(a)$ is true because of the third rule in that program.

[23] We omit the sub- and superscripts (U, M) and P from $T_P^{(U,M)}$ if they are clear from the context and, furthermore, we will usually write T instead of $T(\emptyset)$.

Thus, this example illustrates that there is a difference between our semantics and the one in [Lifschitz et al. 2001b; Osorio et al. 2004; Osorio and Ortiz 2004].

In [Leone and Perri 2003], so-called *parametric connectives* are introduced in the context of disjunctive logic programs. The semantics of parametric connectives, e.g, $\bigwedge\{p(X) : a(X,Y), b(X)\}$, is essentially the same as the semantics of generalized literals $\forall X \cdot a(X,Y) \wedge b(X) \Rightarrow p(X)$. Note that [Leone and Perri 2003] also allows for a disjunction in the body (indicated by a $\bigvee$ instead of $\bigwedge$), however, since we allow for arbitrary boolean formulas in the antecedent of our generalized literals, the latter are more flexible.

6. OPEN ANSWER SET PROGRAMMING WITH GPS VIA FIXED POINT LOGIC

We reduce satisfiability checking w.r.t. gPs to satisfiability checking of FPL formulas. Note that the exposition in this section is along the lines of Section 3, such that we will skip the details of some of the proofs.

First, we rewrite an arbitrary gP as a gP containing only one designated predicate p and (in)equality. A gP P is a *p-gP* if p is the only predicate in P different from the (in)equality predicate. For a set of g-literals α, we construct α_p in two stages:

(1) replace every regular m-ary atom $q(\mathbf{t})$ appearing in α (either in atoms, naf-atoms, or generalized literals) by $p(\mathbf{t}, \mathbf{0}, q)$ where p has arity n, with n the maximum of the arities of predicates in P augmented by 1, $\mathbf{0}$ 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 $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ by $\forall \mathbf{Y} \cdot \phi \wedge \bigwedge in(\mathbf{Y}) \Rightarrow \psi$, where $Y \neq t$ in $in(\mathbf{Y})$ stands for $\neg(Y = t)$ (we defined generalized literals in function of boolean formulas of atoms).

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

Example 6.1. Let P be the gP:

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

Then q is satisfiable by an open answer set $(\{a, x\}, \{s(x), r(a), q(x)\})$. The p-gP P_p is

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

where $in(X) = \{X \neq s, X \neq q, X \neq r, X \neq 0\}$. The corresponding open answer set for this program is $(\{a, x, s, r, q\}, \{p(x,s), p(a,r), p(x,q)\})$.

THEOREM 6.2. *Let P be a gP, p a predicate not in P, 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-gP P_p with $p(\boldsymbol{x}, \boldsymbol{0}, q) \in M'$. Furthermore, the size of P_p is polynomial in the size of P.*

PROOF. The proof is analogous to the proof of Theorem 3.3. □

The *completion* $\texttt{compgl}(P)$ of a gP P consists of formulas that demand that different constants in P are interpreted as different elements:

$$a \neq b \ . \tag{21}$$

For every pair of different constants a and b in P, $\texttt{compgl}(P)$ contains formulas ensuring the existence of at least one element in the domain of an interpretation:

$$\exists X \cdot \mathbf{true} \ . \tag{22}$$

Besides these technical requirements matching FOL interpretations with open interpretations, $\texttt{compgl}(P)$ 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 antecedents of generalized literals are true, and

—$\texttt{fpf}(P)$ ensures that every model of $\texttt{fix}(P)$ is a minimal model of the GL-reduct of the GeLi-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 GeLi-reduct of P; the atoms defined in $\texttt{gli}(P)$ ensure that the generalized literals are interpreted correctly.

In the following, we assume that the arity of p, the only predicate in a p-gP is n.

Definition 6.3. Let P be a p-gP. 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

$$\forall \mathbf{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{23}$$

for rules $r : \alpha \leftarrow \beta \in P$ with $vars(r) = \mathbf{Y}$,

(2) $\texttt{gl}(P)$ contains the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg \beta^- \tag{24}$$

for rules $r : \alpha \leftarrow \beta \in P$ with $vars(r) = \mathbf{Y}$,

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

$$\forall \mathbf{Z} \cdot g(\mathbf{Z}) \Leftrightarrow \phi \tag{25}$$

for generalized literals $g : \forall \mathbf{Y} \cdot \phi \Rightarrow \psi \in P$[24] where ϕ contains the variables $\mathbf{Z}$,

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

$$\forall \mathbf{X} \cdot p(\mathbf{X}) \Rightarrow [\text{LFP } \ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{X}) \tag{26}$$

[24]We assume that generalized literals are named.

with

$$\phi(W, \mathbf{X}) \equiv W(\mathbf{X}) \vee \bigvee_{r:p(\mathbf{t})\vee\alpha\leftarrow\beta\in P} E(r) \tag{27}$$

and

$$E(r) \equiv \exists\mathbf{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p \mid W] \wedge \bigwedge \gamma \wedge r(\mathbf{Y}) \tag{28}$$

where $\mathbf{X} = X_1, \ldots, X_n$ are n new variables, $vars(r) = \mathbf{Y}$, 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 $g : \forall\mathbf{Y} \cdot \phi \Rightarrow \psi$ replaced by $\forall\mathbf{Y} \cdot g(\mathbf{Z}) \Rightarrow \psi$, $\mathbf{Z}$ the variables of ϕ, and, subsequently,
—every p replaced by W.

The *completion* of P is $\texttt{compgl}(P) \equiv \texttt{fix}(P) \cup \{(21), (22)\}$.

The predicate W appears only positively in $\phi(W, \mathbf{X})$ such that the fixed point formula in (26) 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 atoms $g(\mathbf{Z})$, and the negative part of r, which were replaced by atoms $r(\mathbf{Y})$, thus respectively encoding the GeLi-reduct and the GL-reduct.[25]

By the first disjunct in (27), we have that applying $\phi^{(U,M)}$ to a set $S \subseteq U^n$ does not lose information from S.

THEOREM 6.4. *Let P be a p-gP and (U, M) an open interpretation with $S \subseteq U^n$. Then*

$$S \subseteq \phi^{(U,M)}(S) \,.$$

PROOF. Similar to the proof of Theorem 3.6. □

Example 6.5. We rewrite the program from Example 5.3 as the p-gP P:

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

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

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

[25]Note that we apply the GeLi-reduct and the GL-reduct "at the same time", while the open answer set semantics is defined such that first the GeLi-reduct is constructed and then the GL-reduct. However, as indicated by Theorem 5.5, the order of applying the reducts does not matter.

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

—$\forall X, Y \cdot r_1(X, Y) \Leftrightarrow \bigwedge in(X) \wedge \bigwedge in(Y)$,
—$\forall X, Y \cdot r_2(X, Y) \Leftrightarrow \neg p(Y, 0, q) \wedge \bigwedge in(X) \wedge \bigwedge in(Y)$,
—$\forall X, Y \cdot r_3(X, Y) \Leftrightarrow \neg p(Y, 0, well) \wedge \bigwedge in(X) \wedge \bigwedge in(Y)$,
—$\forall Y \cdot r_4(Y) \Leftrightarrow \bigwedge in(Y)$, 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

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

Take an infinite FOL interpretation (U, M) with $U = \{q, f, well, 0, x_0, x_1, \ldots\}$ and[26]

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

$\mathtt{sat}(P), \mathtt{gl}(P)$, and $\mathtt{gli}(P)$ are satisfied. We check that $\mathtt{fpf}(P)$ is satisfied by M. We 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 [Grädel 2002a], 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 $E(r_4)$,
—$W^4 = \phi^{(U,M)}(W^3) = W^3 \cup \{(x_1, 0, well)\}$,
—...

The least fixed point $\mathrm{LFP}(\phi^{(U,M)})$ is then $\cup_{\alpha<\infty} W^\alpha$ [Grädel 2002a]. 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 $\mathtt{compgl}(P)$, and it corresponds to an open answer set of P.

THEOREM 6.6. *Let P be a p-gP. Then, (U, M) is an open answer set of P iff $(U, M \cup R \cup G)$ is a model of $\bigwedge \mathtt{compgl}(P)$, where*

$$R \equiv \{r(\boldsymbol{y}) \mid r[\boldsymbol{Y} \mid \boldsymbol{y}] : \alpha[] \leftarrow \beta[] \in P_U, M \models \alpha[]^- \cup not\ \beta[]^-, vars(r) = \boldsymbol{Y}\} \ ,$$

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

i.e., the atoms corresponding to rules for which the GeLi-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.

PROOF. Similar to the proof of Theorem 3.8. □

Using Theorems 6.2 and 6.6, we can reduce satisfiability checking w.r.t. gPs to satisfiability checking in FPL. Moreover, since $\bigwedge \texttt{compgl}(P)$ contains only one fixed point predicate, the translation falls in the alternation-free fragment of FPL.

THEOREM 6.7. *Let P be a gP, p a predicate not appearing in P, and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \bigwedge \texttt{compgl}(P_p)$ is satisfiable. Moreover, this reduction is polynomial.*

PROOF. Assume q is satisfiable w.r.t. P. By Theorem 6.2, we have that $p(\mathbf{x}, \mathbf{0}, q)$ is in an open answer set of P_p, such that with Theorem 6.6, $p(\mathbf{x}, \mathbf{0}, q)$ is in a model of $\bigwedge \texttt{compgl}(P_p)$.

For the opposite direction, assume $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \texttt{compgl}(P_p)$ is satisfiable. Then there is a model (U, M') of $\bigwedge \texttt{compgl}(P)$ with $p(\mathbf{x}, \mathbf{0}, q) \in M'$. We have that $M' = M \cup R \cup G$ as in Theorem 6.6, such that (U, M) is an open answer set of P_p and $p(\mathbf{x}, \mathbf{0}, q) \in M$. From Theorem 6.2, we then have an open answer set of P satisfying q.

The size of $\bigwedge \texttt{compgl}(P_p)$ is polynomial in the size of P_p. Since the size of the latter is also polynomial in the size of P, the size of $\bigwedge \texttt{compgl}(P_p)$ is polynomial in the size of P. □

7. OPEN ANSWER SET PROGRAMMING WITH GUARDED GENERALIZED PROGRAMS

As we did in Section 4 for programs, we introduce in this section a notion of guardedness such that the FPL translation of *guarded gPs* falls in μGF. We do not, however, consider their *loosely guarded* counterpart like we did in Section 4, but leave this as an exercise to the reader.

Definition 7.1. A generalized literal $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ is *guarded* if ϕ is of the form $\gamma \wedge \phi'$ with γ an atom, and $vars(\mathbf{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$; 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.

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

Example 7.2. Reconsider the gP from Example 5.3. 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 gP is guarded. Vice versa, we can transform every guarded gP into an equivalent fully guarded one. For a GgP P, P^{f} is defined as in Section 4

(pp. 35), i.e., as P with the rules $\alpha \leftarrow \beta$ replaced by $\alpha \cup \mathit{not}\ \beta^+ \leftarrow \beta$ for the body guard γ_b of $\alpha \leftarrow \beta$. For a GgP P, we have that P^{f} is a FGgP, 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.

THEOREM 7.3. *Let P be a GgP. 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}.*

PROOF. The proof is analogous to the proof of Theorem 4.8 (pp. 21). □

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

THEOREM 7.4. *Let P be a gP. Then, P is a (F)GgP iff P_p is a (F)GgP.*

PROOF. The proof is analogous to the proof of Theorem 4.10 (pp. 21). □

For a fully guarded p-gP P, we can rewrite $\mathtt{compgl}(P)$ as the equivalent μGF formulas $\mathtt{gcompgl}(P)$. For a guarded generalized literal $\xi \equiv \forall \mathbf{Y} \cdot \phi \Rightarrow \psi$, define

$$\xi^g \equiv \forall \mathbf{Y} \cdot \gamma \Rightarrow \psi \vee \neg\phi' \ ,$$

where, since the generalized literal is guarded, $\phi = \gamma \wedge \phi'$, and $\mathit{vars}(\mathbf{Y}) \cup \mathit{vars}(\phi') \cup \mathit{vars}(\psi) \subseteq \mathit{vars}(\gamma)$, making formula ξ^g a guarded formula. The extension of this operator $\cdot^g$ for sets (or boolean formulas) of generalized literals is as usual.

$\mathtt{gcompgl}(P)$ is $\mathtt{compgl}(P)$ with the following modifications.

—Formula $\exists X \cdot \mathbf{true}$ is replaced by

$$\exists X \cdot X = X \ , \tag{29}$$

such that it is guarded by $X = X$.

—Formula (23) is removed if $r : \alpha \leftarrow \beta$ is free or otherwise replaced by

$$\forall \mathbf{Y} \cdot \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \{\gamma_b\}) \vee \bigvee \beta^- \vee \bigvee \neg(\beta^{\mathrm{x}})^g \ , \tag{30}$$

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 $q(\mathbf{t}) \vee \mathit{not}\ q(\mathbf{t}) \leftarrow$ we have $\forall Y \cdot \mathbf{true} \Rightarrow q(\mathbf{t}) \vee \neg q(\mathbf{t})$ which is always true and can thus be removed from $\mathtt{compgl}(P)$.

—Formula (24) is replaced by the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{31}$$

and

$$\forall \mathbf{Y} \cdot \gamma_h \Rightarrow r(\mathbf{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \setminus \{\gamma_h\}) \ , \tag{32}$$

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 $r(\mathbf{Y})$ and the second one is guarded by the head guard of the rule.

—Formula (25) is replaced by the formulas

$$\forall \mathbf{Z} \cdot g(\mathbf{Z}) \Rightarrow \phi \tag{33}$$

and

$$\forall \mathbf{Z} \cdot \gamma \Rightarrow g(\mathbf{Z}) \vee \neg\phi' \tag{34}$$

where $\phi = \gamma \wedge \psi$ by the guardedness of the generalized literal $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$. We thus rewrite an equivalence as two implications where the first one is guarded by $g(\mathbf{Z})$ ($vars(\phi) = \mathbf{Z}$ by definition of g), and the second one is guarded by γ ($vars(g(\mathbf{Z}) \vee \neg\phi') = vars(\mathbf{Z}) = vars(\gamma)$).

—For every $E(r)$ in (26), replace $E(r)$ by

$$E'(r) \equiv \bigwedge_{t_i \notin \mathbf{Y}} X_i = t_i \wedge \exists \mathbf{Z} \cdot (\bigwedge \beta^+[p|W] \wedge \bigwedge \gamma \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y}|X_i] \,, \qquad (35)$$

with $\mathbf{Z} = \mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\}$, 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 $\bigwedge \beta^+[p|W] \wedge \bigwedge \gamma \wedge r(\mathbf{Y})$ by X_i. This rewriting makes sure that every (free) variable in the quantified part of $E'(R)$ is guarded by $r(\mathbf{Y})[t_i \in \mathbf{Y}|X_i]$.

Example 7.5. The rule

$$r : p(X) \vee \mathit{not}\ p(X) \leftarrow p(X), [\forall Y \cdot p(Y) \wedge p(b) \Rightarrow p(a)]$$

constitutes a fully guarded p-gP 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 $E(r) \equiv \exists X \cdot X_1 = X \wedge W(X) \wedge (\forall Y \cdot g(Y) \Rightarrow W(a)) \wedge r(X)$.

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

THEOREM 7.6. *Let P be a fully guarded p-gP. (U, M) is a model of $\bigwedge \mathtt{compgl}(P)$ iff (U, M) is a model of $\bigwedge \mathtt{gcompgl}(P)$.*

PROOF. The only notable difference from the proof of Theorem 4.12 is the presence of generalized literals, which are handled by the observation that $(U, M) \models \xi \iff (U, M) \models \xi^g$ for a generalized literal ξ. □

THEOREM 7.7. *Let P be a fully guarded p-gP. $\bigwedge \mathtt{gcompgl}(P)$ is an alternation-free μGF formula.*

PROOF. The proof is analogous to the proof of Theorem 4.13. □

THEOREM 7.8. *Let P be a GgP and q an n-ary predicate in P. q is satisfiable w.r.t. P iff $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \bigwedge \mathtt{gcompgl}((P^{\mathrm{f}})_p)$ is satisfiable. Moreover, this reduction is polynomial.*

PROOF. We have that P^{f} is a FGgP. By Theorem 7.4, we have that $(P^{\mathrm{f}})_p$ is a fully guarded p-gP, thus the formula $\bigwedge \mathtt{gcompgl}((P^{\mathrm{f}})_p)$ is defined. By Theorem 7.3, we have that q is satisfiable w.r.t. P iff q is satisfiable w.r.t. P^{f}. By Theorem 6.7, we have that q is satisfiable w.r.t. P^{f} iff $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{compgl}((P^{\mathrm{f}})_p)$ is satisfiable. Finally, Theorem 7.6 yields that q is satisfiable w.r.t. P iff $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcompgl}((P^{\mathrm{f}})_p)$ is satisfiable. □

COROLLARY 7.9. *Satisfiability checking w.r.t. GgPs can be polynomially reduced to satisfiability checking of alternation-free μGF-formulas.*

PROOF. For a GgP P, we have, by Theorem 7.7, that $\bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is an alternation-free μGF, which yields with Theorem 7.8, the required result. □

COROLLARY 7.10. *Satisfiability checking w.r.t. GgPs is in 2-*EXPTIME.

PROOF. Since satisfiability checking of μGF formulas is 2-EXPTIME-complete (Theorem [1.1] in [Grädel and Walukiewicz 1999]), satisfiability checking w.r.t. GgPs is, by Corollary 7.9, 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, see Theorem 4.16.

In [Syrjänen 2004], *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 $\mathit{Card}(b, S)$[27] 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 $q \leftarrow \mathit{Card}(n, \{X.a(X) : b(X)\})$, resulting[28] in a rule $q \leftarrow \mathit{Card}(n, \{a(x_1), a(x_2)\})$. 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.

8. RELATIONSHIP WITH DATALOG LITE

We define *Datalog* LITE as in [Gottlob et al. 2002]. 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

[27] $\mathit{Card}(b, S)$ is true if at least b elements from S are true.
[28] Assume we again have a universe $\{x_1, x_2\}$.

Datalog program P, given a relational input structure $\mathcal{U}$ defined over extensional predicates of P[29], is given by the unique (subset) minimal model of Σ_P whose restriction to the extensional predicates yields $\mathcal{U}$ (Σ_P are the first-order clauses corresponding to P, see [Abiteboul et al. 1995]).

For a query (P, q), where P is a basic Datalog program and q is an n-ary predicate, we write $\mathbf{a} \in (P, q)(\mathcal{U})$ if the minimal model M of Σ_P with input $\mathcal{U}$ contains $q(\mathbf{a})$. We call (P, q) satisfiable if there exists a $\mathcal{U}$ and an $\mathbf{a}$ such that $\mathbf{a} \in (P, q)(\mathcal{U})$.

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 [Abiteboul et al. 1995]:

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

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 (i.e., $edb(P_i)$) is equal to $\mathcal{U}_{i-1}|edb(P_i)$. The *least fixed point model* with input $\mathcal{U}$ of P is per definition $\mathcal{U}_n$.

A *Datalog* LITE *generalized literal* is a generalized literal $\forall \mathbf{Y} \cdot a \Rightarrow b$ where a and b are atoms and $vars(b) \subseteq vars(a)$. Note that Datalog LITE generalized literals $\forall \mathbf{Y} \cdot a \Rightarrow b$ can be replaced by the equivalent $\forall \mathbf{Z} \cdot a \Rightarrow b$ where $\mathbf{Z} \equiv \mathbf{Y} \setminus \{Y \mid Y \notin vars(a)\}$, 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 4.5.

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 $\forall \mathbf{Y} \cdot a \Rightarrow b$ 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 [Gottlob et al. 2002]: every generalized literal $\forall \mathbf{Y} \cdot a \Rightarrow b$ is instantiated (for any $\mathbf{x}$ grounding the free variables $\mathbf{X}$ in the generalized literal) by $\bigwedge\{b[\mathbf{X} \mid \mathbf{x}][\mathbf{Y} \mid \mathbf{y}] \mid a[\mathbf{X} \mid \mathbf{x}][\mathbf{Y} \mid \mathbf{y}] \text{ is true}\}$, which is well-defined since a is defined in a lower stratum than the rule where the generalized literal appears.

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

8.1 Reduction from GgPs to Datalog LITE

In [Gottlob et al. 2002], Theorem 8.5., a Datalog LITE query (π_φ, q_φ) was defined for an alternation-free μGF sentence φ such that

$$(U, M) \models \varphi \iff (\pi_\varphi, q_\varphi)(M \cup id(U)) \text{ evaluates to true } ,$$

where the latter means that q_φ is in the fixed point model of π_φ with input $M \cup id(U)$ and $id(U) \equiv \{x = x \mid x \in U\}$.

Example 8.1. Take the μGF sentence $\texttt{gcomp}(P) \equiv \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ from Example 4.11, i.e., with

$$\begin{aligned}
\varphi_1 &\equiv \forall X \cdot p(X) \Rightarrow p(X) \vee \neg p(X) \\
\varphi_2 &\equiv \forall X \cdot r(X) \Rightarrow p(X) \\
\varphi_3 &\equiv \forall X \cdot p(X) \Rightarrow r(X) \\
\varphi_4 &\equiv \forall X \cdot p(X) \Rightarrow [\text{LFP } WX.\phi(W, X)](X)
\end{aligned}$$

and $\phi(W, X) \equiv W(X) \vee (W(X) \wedge r(X))$. The query $(\pi_{\texttt{gcomp}(P)}, q_{\texttt{gcomp}(P)})$ considers atoms and negated atoms as extensional predicates and introduces rules

$$\begin{aligned}
H_{p,\varphi_1}(X) &\leftarrow p(X) \\
H_{\neg p,\varphi_1}(X) &\leftarrow p(X), \neg p(X)
\end{aligned}$$

for φ_1 where both rules are guarded by the guard $p(X)$ of φ_1 (or, in general, the guard in the most closely encompassing scope). Disjunction is defined as usual:

$$\begin{aligned}
H_{p \vee \neg p,\varphi_1}(X) &\leftarrow p(X), H_{p,\varphi_1}(X) \\
H_{p \vee \neg p,\varphi_1}(X) &\leftarrow p(X), H_{\neg p,\varphi_1}(X)
\end{aligned}$$

where $p(X)$ serves again as guard.[30] The sentence φ_1 itself is translated into

$$H_{\varphi_1} \leftarrow (\forall X \cdot p(X) \Rightarrow H_{p \vee \neg p,\varphi_1}(X))$$

Formulas φ_2 and φ_3 can be translated similarly. For φ_4, we translate, as an intermediate step, $\phi(W, X)$ as

$$\begin{aligned}
H_\phi(X) &\leftarrow p(X), H_W(X) \\
H_\phi(X) &\leftarrow p(X), H_{W \wedge r}(X) \\
H_{W \wedge r}(X) &\leftarrow p(X), H_W(X), H_r(X) \\
H_W(X) &\leftarrow p(X), W(X) \\
H_r(X) &\leftarrow p(X), r(X)
\end{aligned}$$

from which the translation for $[\text{LFP } WX.\phi(W, X)](X)$ can be obtained by replacing $H_\phi(X)$ and $W(X)$ by $H_{[\text{LFP } WX.\phi(W,X)](X)}$, i.e.,

$$\begin{aligned}
H_{[\text{LFP } WX.\phi(W,X)]}(X) &\leftarrow p(X), H_W(X) \\
H_{[\text{LFP } WX.\phi(W,X)]}(X) &\leftarrow p(X), H_{W \wedge r}(X) \\
H_{W \wedge r}(X) &\leftarrow p(X), H_W(X), H_r(X) \\
H_W(X) &\leftarrow p(X), H_{[\text{LFP } WX.\phi(W,X)]}(X) \\
H_r(X) &\leftarrow p(X), r(X)
\end{aligned}$$

[30] Actually, in this particular case, the rules would already be guarded without the guard of φ_1, but we include it, as this is not true in general.

The sentence φ_4 is translated to

$$H_{\varphi_4} \leftarrow (\forall X \cdot p(X) \Rightarrow H_{[\text{LFP } WX.\phi(W,X)]}(X))$$

Finally, we compile the results in the rule $q_{\texttt{gcomp}(P)} \leftarrow H_{\varphi_1}, H_{\varphi_2}, H_{\varphi_3}, H_{\varphi_4}$.

In Example 4.11, we had, for a universe $\{x\}$, the unique model $(\{x\}, \emptyset)$ of $\texttt{gcomp}(P)$. Accordingly, we have that $\{x = x\}$ is the only relational input structure on the extensional predicates of $\pi_{\texttt{gcomp}(P)}$, r and p, that contains the term x and results in a least fixed point model of $\pi_{\texttt{gcomp}(P)}$ containing $q_{\texttt{gcomp}(P)}$.

For the formal details of this reduction, we refer to [Gottlob et al. 2002]. Satisfiability checking w.r.t. GgPs can be polynomially reduced, using the above reduction, to satisfiability checking in Datalog LITE.

THEOREM 8.2. *Let P be a GgP, q an n-ary predicate in P, and φ the μGF sentence $\exists \boldsymbol{X} \cdot p(\boldsymbol{X}, \boldsymbol{0}, q) \wedge \bigwedge \texttt{gcomp}((P^{\text{f}})_p)$. q is satisfiable w.r.t. P iff (π_φ, q_φ) is satisfiable. Moreover, this reduction is polynomial.*

PROOF. By Theorem 7.8, we have that q is satisfiable w.r.t. P iff φ is satisfiable. Since φ is a μGF sentence, we have that φ is satisfiable, i.e., there exists a (U, M) such that $(U, M) \models \varphi$, iff $(\pi_\varphi, q_\varphi)(M \cup id(U))$ evaluates to true, i.e., (π_φ, q_φ) is satisfiable.

Since, by Theorem 7.8, the translation of P to φ is polynomial in the size of P and the query (π_φ, q_φ) is polynomial in φ [Gottlob et al. 2002], we have a polynomial reduction. □

8.2 Reduction from Datalog LITE to GgPs

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

LEMMA 8.3. *Let $P = (P_1, \ldots, P_n)$ be a stratified Datalog program, possibly with generalized literals, and $\mathcal{U}$ an input structure for P. If $p(\boldsymbol{x}) \in S_j$, then p is a head predicate in P_j or $p \in \mathcal{U}_{j-1}|edb(P_j)$.*

PROOF. Either $p \in edb(P_j)$ or not. In the former case, we have that $p(\mathbf{x}) \in S_j|edb(P_j)$ such that, by the definition of S_j, $p(\mathbf{x}) \in \mathcal{U}_{j-1}|edb(P_j)$. In the latter case, we have that, since p does not appear in the body of P_j, but nevertheless $p(\mathbf{x})$ is in S_j, a minimal model of P_j, p must be a head predicate in P_j. □

LEMMA 8.4. *Let $P = (P_1, \ldots, P_n)$ be a stratified Datalog program, possibly with generalized literals, $\mathcal{U}$ an input structure for P. If p is a head predicate in some P_j, $1 \leq j \leq n$, then*

$$p(\boldsymbol{x}) \in S_j \iff p(\boldsymbol{x}) \in \mathcal{U}_n \,. \tag{36}$$

If $p \in edb(P_j)$ and $p(\boldsymbol{x}) \notin \mathcal{U}_{j-1}$, then $p(\boldsymbol{x}) \notin \mathcal{U}_n$.

PROOF. The "only if" direction of Equation (36) is immediate. For the "if" direction: assume p is a head predicate in P_j and $p(\mathbf{x}) \in \mathcal{U}_n$. Since $p(\mathbf{x}) \in \mathcal{U}_n$, there must be a k, such that $p(\mathbf{x}) \in S_k$, $1 \leq k \leq n$.

If $k = j$, we are finished, otherwise, by Lemma 8.3, $p(\mathbf{x}) \in \mathcal{U}_{k-1}|edb(P_k)$ and thus $p(\mathbf{x}) \in \mathcal{U}_{k-1}$. Again, we have that there is a $1 \leq k_1 \leq k-1$, such that $p(\mathbf{x}) \in S_{k_1}$.

If $k_1 = j$, we are finished, otherwise, we continue as before. After at most n steps, we must find a $k_n = j$, otherwise we have a contradiction ($p(\mathbf{x}) \in \mathcal{U}$ is not possible since p is a head predicate and input structures are defined on extensional predicates only).

Take p extensional in P_j, $p(\mathbf{x}) \notin \mathcal{U}_{j-1}$, and $p(\mathbf{x}) \in \mathcal{U}_n$. We show that this leads to a contradiction. From $p(\mathbf{x}) \in \mathcal{U}_n$, we have that $p(\mathbf{x}) \in \mathcal{U}_{n-1}$ or $p(\mathbf{x}) \in S_n$. For the latter, one would have, with Lemma 8.3, that $p(\mathbf{x}) \in \mathcal{U}_{n-1}|edb(P_n)$ or p is a head predicate in S_n. The latter is impossible since $p \in edb(P_j)$ and $j \leq n$. Thus, we have that $p(\mathbf{x}) \in \mathcal{U}_{n-1}$.

Continuing this way, we eventually have that $p(\mathbf{x}) \in \mathcal{U}_{j-1}$, a contradiction. □

THEOREM 8.5. *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 $M = \mathcal{U}_n \setminus id(U)$.

PROOF. For the "only if" direction, assume $\mathcal{U}_n \models l$. Define

$$M \equiv \mathcal{U}_n \setminus id(U) \; .$$

Clearly, $M \models l$, such that remains to show that (U, M) is an open answer set of P.

(1) M is a model of $R \equiv (P_U^{\mathrm{x}(U,M)})^M$.
Take a rule $r : a[\mathbf{X} \mid \mathbf{x}] \leftarrow \beta[\mathbf{X} \mid \mathbf{x}]^+, (\beta[\mathbf{X} \mid \mathbf{x}]^{\mathrm{x}})^{\mathrm{x}(U,M)} \in R$, thus $M \models not\ \beta[]^-$, originating from $a \leftarrow \beta \in P$. Assume $M \models \mathrm{body}(r)$. We have that $\forall \mathbf{X} \cdot \bigwedge \beta \Rightarrow a \in \Sigma_{P_i}$ for some stratum P_i. Take $\mathbf{x}$ as in r.
We verify that $\mathcal{U}_n \models \bigwedge \beta[]$. We have that $\mathcal{U}_n \models \bigwedge \beta[]^+ \wedge \bigwedge \neg\beta[]^-$. Take a generalized literal $\forall \mathbf{Y} \cdot c \Rightarrow b$ in $\bigwedge \beta[]$ and $\mathcal{U}_n \models c[\mathbf{Y} \mid \mathbf{y}]$. Then $M \models c[]$ such that $b[] \in (\beta[\mathbf{X} \mid \mathbf{x}]^{\mathrm{x}})^{\mathrm{x}(U,M)}$, and thus, with $M \models b[]$, that $\mathcal{U}_n \models b[]$.
With Theorem 15.2.11 in [Abiteboul et al. 1995], we have that $\mathcal{U}_n$ is a model of Σ_P, such that $a[] \in \mathcal{U}_n$, and thus $M \models a[]$.

(2) M is a minimal model of $R \equiv (P_U^{\mathrm{x}(U,M)})^M$.
Assume not, then there is a $N \subset M$, model of R. Define $N' \equiv N \cup id(U)$. Since $M \setminus N \neq \emptyset$, we have that $\mathcal{U}_n \setminus N' \neq \emptyset$. Since $\mathcal{U} = id(U)$, we have $\mathcal{U}_n = id(U) \cup S_1 \cup \ldots \cup S_n$, such that there is a $1 \leq j \leq n$, where $S_j \setminus N' \neq \emptyset$ and $\mathcal{U}_{j-1} \subseteq N'$. Define $N_j \equiv S_j \setminus (S_j \setminus N')$. One can show that $N_j \subset S_j$, $N_j|edb(P_j) = \mathcal{U}_{j-1}|edb(P_j)$, and N_j is a model of Σ_{P_j}, which is a contradiction with the minimality of S_j.

For the "if" direction, assume (U, M) is an open answer set of P with $M \models l$. Assume $\mathcal{U}_n \not\models l$. Define $M' \equiv \mathcal{U}_n \setminus id(U)$. By the previous direction, we know that (U, M') is an open answer set of P with $M' \not\models l$, such that $M \models l$ and $M' \not\models l$. Note that $M \models p(\mathbf{x}) \iff M' \models p(\mathbf{x})$ for extensional predicates p in P. Indeed, assume $M \models p(\mathbf{x})$, then $p(\mathbf{x})$ must be in the head of an applied rule since M is an answer set, contradicting that p is extensional, unless p is an equality, and then $\emptyset \models p(\mathbf{x})$ such that $M' \models p(\mathbf{x})$. The other direction is similar.

One can show per induction on k, that for a head predicate p in P_k, $M \models p(\mathbf{x})$ iff $M' \models p(\mathbf{x})$, resulting in $M = M'$, and thus in particular we have a contradiction for l, such that $l \in \mathcal{U}_n$.

In particular, we have $M = M' = \mathcal{U}_n \setminus id(U)$, which proves the last part of the Theorem. □

From Theorem 8.5, we obtain a generalization of Corollary 2 in [Gelfond and Lifschitz 1988] (*If* Π *is stratified, then its unique stable model is identical to its fixed point model.*) for stratified Datalog programs with generalized literals and an open answer set semantics.

COROLLARY 8.6. *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 Theorem 8.5, to take into account arbitrary input structures $\mathcal{U}$. For a stratified Datalog program P, possibly with generalized literals, define $F_P \equiv \{q(\mathbf{X}) \vee not\ q(\mathbf{X}) \leftarrow\mid q \text{ extensional (but not =) in } P\}$.

THEOREM 8.7. *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 $M \models l$.*

PROOF. For the "only if" direction, assume $\mathcal{U}_n \models l$. Define $U \equiv cts(P \cup \mathcal{U})$ and

$$M \equiv \mathcal{U}_n \setminus id(U) \ .$$

Clearly, $M \models l$, and one can show, similarly to the proof of Theorem 8.5, that (U, M) is an open answer set of P.

For the "if" direction, assume (U, M) is an open answer set of $P \cup F_P$ with $M \models l$. Define

$$\mathcal{U} \equiv id(U) \cup \{q(\mathbf{x}) \mid q(\mathbf{x}) \in M \wedge q \text{ extensional (but not equality) in } P\} \ .$$

Take $\mathcal{U}_n$ the least fixed point model with input $\mathcal{U}$. Assume $\mathcal{U}_n \not\models l$. Define $M' \equiv \mathcal{U}_n \setminus id(U)$. By the previous direction, we know that $(cts(\mathcal{U} \cup P)(= U), M')$ is an open answer set of P with $M' \not\models l$, such that $M \models l$ and $M' \not\models l$. The rest of the proof is along the lines of the proof of Theorem 8.5. □

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 (instead of guarded or monadic). As we will see below this is not a restriction. As F_P contains only free rules, $P \cup F_P$ is a GgP if P is a Datalog LITEM program. Furthermore, the size of the GgP $P \cup F_P$ is linear in the size of P.

THEOREM 8.8. *Let P be a Datalog* LITEM *program. Then, $P \cup F_P$ is a GgP whose size is linear in the size of P.*

PROOF. Immediate by the Definition of Datalog LITEM (note also the remark at pp. 39) and the fact that F_P is a set of free rules and thus has no influence on the guardedness of P. □

Satisfiability checking of Datalog LITEM queries can be reduced to satisfiability checking w.r.t. GgPs.

THEOREM 8.9. *Let (P, q) be a Datalog* LITEM *query. Then, (P, q) is satisfiable iff q is satisfiable w.r.t. the GgP $P \cup F_P$. Moreover, this reduction is linear.*

PROOF. Immediate by Theorems 8.7 and 8.8. □

Theorems 8.2 and 8.9 lead to the conclusion that Datalog LITEM and open ASP with GgPs are equivalent (i.e., satisfiability checking in either one of the formalisms can be polynomially reduced to satisfiability checking in the other).[31] Furthermore, since Datalog LITEM, Datalog LITE, and alternation-free μGF are equivalent as well [Gottlob et al. 2002], we have the following result.

THEOREM 8.10. *Datalog* LITE*, alternation-free μGF, and open ASP with GgPs are equivalent.*

Satisfiability checking in both GF and LGF is 2-EXPTIME-complete [Grädel 1999], as are their (alternation-free) extensions with fixed point predicates μGF and μLGF [Grädel and Walukiewicz 1999]. Theorem 8.10 gives us then immediately the following complexity result.

THEOREM 8.11. *Satisfiability checking w.r.t. GgPs is 2-*EXPTIME*-complete.*

Some extra terminology is needed to show that satisfiability checking w.r.t. (L)GPs (i.e., without generalized literals) is 2-EXPTIME-complete as well.

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 LITEM, 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 with all generalized literals 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

$$q(X) \leftarrow f(X), \mathit{not}\ q'(X)$$

and

$$q'(X) \leftarrow r(X, Y), \mathit{not}\ s(Y)\ .$$

As indicated in [Gottlob et al. 2002], this yields an equivalent program $\neg\neg P$, where the recursion-freeness ensures that $\neg\neg P$ is stratified.[32]

THEOREM 8.12. *Let P be a Datalog* LITER *program. Then $\neg\neg P \cup F_{\neg\neg P}$ is a GP.*

[31] Note that (π_φ, q_φ) is a Datalog LITEM query [Gottlob et al. 2002].

[32] Note that this translation cannot work for arbitrary generalized programs as the antecedent of generalized literals can be an arbitrary boolean formula, which cannot appear in bodies of rules. Replace, e.g., $r(X, Y)$ by $r(X, Y) \vee d(X, Y)$.

PROOF. Every rule in P is guarded, and thus every rule in $\neg\neg P$ is too. Since $\neg\neg P \cup F_{\neg\neg P}$ adds but free rules to $\neg\neg P$, all non-free rules of $\neg\neg P \cup F_{\neg\neg P}$ are guarded. □

Satisfiability checking of Datalog LITER queries can be linearly reduced to satisfiability checking w.r.t. GPs.

THEOREM 8.13. *Let (P, q) be a Datalog* LITER *query. (P, q) is satisfiable iff q is satisfiable w.r.t. the GP $\neg\neg P \cup F_{\neg\neg P}$. Moreover, this reduction is linear.*

PROOF. For a Datalog LITER query (P, q), $(\neg\neg P, q)$ is an equivalent stratified Datalog query. Hence, by Theorem 8.7, $(\neg\neg P, q)$ is satisfiable iff q is satisfiable w.r.t. $\neg\neg P \cup F_{\neg\neg P}$. This reduction is linear since $\neg\neg P$ is linear in the size of P and so is $\neg\neg P \cup F_{\neg\neg P}$. □

THEOREM 8.14. *Satisfiability checking w.r.t. (L)GPs is 2-*EXPTIME*-complete.*

PROOF. The reduction from alternation-free μGF sentences φ to Datalog LITE queries (π_φ, q_φ) specializes, as noted in [Gottlob et al. 2002], 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 [Grädel 1999], such that, using Theorem 8.13 and the intermediate Datalog LITER translation, we have that satisfiability checking w.r.t. GPs is 2-EXPTIME-hard. The 2-EXPTIME membership was shown in Theorem 4.16, such that the completeness readily follows.

Every GP is a LGP and satisfiability checking w.r.t. to the former is 2-EXPTIME-complete, thus we have 2-EXPTIME-hardness for satisfiability checking w.r.t. LGPs. Completeness follows again from Theorem 4.16. □

9. CTL REASONING USING GUARDED GENERALIZED PROGRAMS

In this section, we show how to reduce CTL satisfiability checking to satisfiability checking w.r.t. GgPs, i.e., guarded programs with generalized literals, thus arguing the usability of OASP as a suitable formalism for different kinds of knowledge representation.

In order to keep the treatment simple, we will assume that the only allowed temporal constructs are $\mathsf{AF}q$, $\mathsf{E}(p\ \mathsf{U}\ q)$, and $\mathsf{EX}q$, for formulas p and q. They are actually adequate in the sense that other temporal constructs can be equivalently, i.e., preserving satisfiability, rewritten using only those three [Huth and Ryan 2000].

For a CTL formula p, let $clos(p)$ be the *closure* of p: the set of subformulas of p. We construct a GgP $G \cup D_p$ consisting of a generating part G and a defining part D_p. The guarded program G contains free rules (g_1) for every proposition $P \in AP$, free rules (g_2) that allow for state transitions, and rules (g_3) that ensure that the transition relation is total:

$$[P](S) \vee not\ [P](S) \leftarrow \qquad (g_1)$$

$$next(S, N) \vee not\ next(S, N) \leftarrow \qquad (g_2)$$

$$succ(S) \leftarrow next(S, N) \qquad \leftarrow S = S, not\ succ(S) \qquad (g_3)$$

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 GgP 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 $S = S$:

—For a formula $\neg q$ in $clos(p)$, we introduce in D_p the rule

$$[\neg q](S) \leftarrow not\ [q](S) \qquad (d_1)$$

Thus, the negation of a CTL formula is simulated by negation as failure.

—For a formula $q \wedge r$ in $clos(p)$, we introduce in D_p the rule

$$[q \wedge r](S) \leftarrow [q](S), [r](S) \qquad (d_2)$$

Conjunction of CTL formulas thus corresponds to conjunction in the body.

—For a formula $\mathsf{AF}q$ in $clos(p)$, we introduce in D_p the rules

$$[\mathsf{AF}q](S) \leftarrow [q](S) \qquad (d_3^1)$$
$$[\mathsf{AF}q](S) \leftarrow \forall N \cdot next(S, N) \Rightarrow [\mathsf{AF}q](N) \qquad (d_3^2)$$

We define $\mathsf{AF}q$ corresponding to the intuition that $\mathsf{AF}q$ holds if, either q holds at the current state (d_3^1) or for all successors, we have that $\mathsf{AF}q$ holds (d_3^2). Note that we use generalized literals to express the *for all successors* part. Moreover, we explicitly use the minimal model semantics of (open) answer set programming to ensure that eventually $[q]$ holds on all paths: one cannot continue to use rule (d_3^2) to motivate satisfaction of $\mathsf{AF}q$, at a certain finite point, one is obliged to use rule (d_3^1) to obtain a finite motivation.

—For a formula $\mathsf{E}(q\ \mathsf{U}\ r)$ in $clos(p)$, we introduce in D_p the rules

$$[\mathsf{E}(q\ \mathsf{U}\ r)](S) \leftarrow [r](S) \qquad (d_4)$$
$$[\mathsf{E}(q\ \mathsf{U}\ r)](S) \leftarrow [q](S), next(S, N), [\mathsf{E}(q\ \mathsf{U}\ r)](N) \qquad (d_5)$$

based on the intuition that there is a path where q holds until r holds (and r eventually holds) if either r holds at the current state (d_4), or q holds at the current state and there is some next state where again $\mathsf{E}(q\ \mathsf{U}\ r)$ holds (d_5). The minimality will again make sure that we eventually must deduce r with rule (d_4).

—For a formula $\mathsf{EX}q$ in $clos(p)$, we introduce in D_p the rule

$$[\mathsf{EX}q](S) \leftarrow next(S, N), [q](N) \qquad (d_6)$$

saying that $\mathsf{EX}q$ holds if there is some successor where q holds.

Note that replacing the generalized literal in (d_3^2) with a double negation has not the intended effect:

$$\begin{aligned} [\mathsf{AF}q](S) &\leftarrow not\ q'(S) \\ q'(S) &\leftarrow next(S, N), not\ [\mathsf{AF}q](N) \end{aligned}$$

A (fragment) of an open answer set could then be

$$(\{s_0, s_1, \ldots\}, \{next(s_0, s_1), next(s_1, s_2), \ldots, \\ [\mathsf{AF}q](s_0), [\mathsf{AF}q](s_1), \ldots\}) \ ,$$

such that one would conclude that $[\mathsf{AF}q]$ is satisfiable while there is a path $s_0, s_1, \ldots$ where q never holds.

Example 9.1. Consider the absence of starvation formula $t \Rightarrow \mathsf{AF}c$, i.e., if a process tries (t) to access a critical section of code, it must eventually succeed in doing so (c). We rewrite this such that it does not contain $\Rightarrow$, i.e., we consider the equivalent formula $\neg(t \wedge \neg\mathsf{AF}c)$. For $AP = \{c, t\}$, the program G contains the rules

$$\begin{aligned} [t](S) \vee not\ [t](S) &\leftarrow \\ [c](S) \vee not\ [c](S) &\leftarrow \\ next(S, N) \vee not\ next(S, N) &\leftarrow \\ succ(S) &\leftarrow next(S, N) \\ &\leftarrow S = S, not\ succ(S) \end{aligned}$$

The program D_p, with $p \equiv \neg(t \wedge \neg\mathsf{AF}c)$, contains the rules

$$\begin{aligned} [\neg(t \wedge \neg\mathsf{AF}c)](S) &\leftarrow not\ [t \wedge \neg\mathsf{AF}c](S) \\ [t \wedge \neg\mathsf{AF}c](S) &\leftarrow [t](S), [\neg\mathsf{AF}c](S) \\ [\neg\mathsf{AF}c](S) &\leftarrow not\ [\mathsf{AF}c](S) \\ [\mathsf{AF}c](S) &\leftarrow [c](S) \\ [\mathsf{AF}c](S) &\leftarrow \forall N \cdot next(S, N) \Rightarrow [\mathsf{AF}c](N) \end{aligned}$$

One can see that p is (CTL) satisfiable iff $[p]$ is satisfiable w.r.t. $G \cup D_p$.

THEOREM 9.2. *Let p be a CTL formula. p is satisfiable iff $[p]$ is satisfiable w.r.t. the GgP $G \cup D_p$.*

PROOF. For the "only if" direction, assume p is satisfiable. Then there exists a model $K = (S, R, L)$ of p such that $K, s \models p$, for a state $s \in S$. Define

$$\begin{aligned} M \equiv &\{next(s, t) \mid (s, t) \in R\} \cup \{succ(s) \mid (s, t) \in R\} \\ &\cup \{[q](s) \mid K, s \models q \wedge q \in clos(p)\} \ . \end{aligned}$$

Then $[p](s) \in M$; one can show that (S, M) is an open answer set of $G \cup D_p$.

For the "if" direction, assume (U, M) is an open answer set of $G \cup D_p$ such that $[p](s) \in M$ for some s, where p is a CTL formula. Define the model $K = (U, R, L)$ with $R = \{(s, t) \mid next(s, t) \in M\}$, and $L(s) = \{P \mid [P](s) \in M \wedge P \in AP\}$. Remains to show that K is a structure and $K, s \models p$.

The relation R is total, indeed, assume not, then there is a $t \in U$, which has no successors in R. Then, there is no $next(t, t') \in M$, such that $succ(t) \notin M$, and the constraint (g_3) gives a contradiction. One can prove per induction on the structure of a CTL formula q, that

$$K, s \models q \iff [q](s) \in M \ .$$

□

Since CTL satisfiability checking is EXPTIME-complete (see Theorem 2.5, pp. 11) and satisfiability checking w.r.t. GgPs is 2-EXPTIME-complete (see Theorem 8.11, pp. 44), the reduction from CTL to GgPs does not seem to be optimal. However, we can show that the particular GgP $G \cup D_p$ is a *bound* GgP for which reasoning is indeed EXPTIME-complete and thus optimal.

The *width* of a formula ψ is the maximal number of free variables in its subformulas [Grädel 2002b]. We define *bound* programs by looking at their first-order form and the arity of its predicates.

Definition 9.3. Let P be a gP. 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 GgP.

THEOREM 9.4. *Let p be a CTL formula. Then, $G \cup D_p$ is a bound GgP.*

PROOF. 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. □

THEOREM 9.5. *Satisfiability checking w.r.t. bound GgPs is* EXPTIME*-complete.*

PROOF. Let P be a bound GgP. We have that $(P^{\mathrm{f}})_p$ is bound and one can check that $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcompgl}((P^{\mathrm{f}})_p)$ is of bounded width. Note that formula (26) on pp. 32 contains a $p(\mathbf{X})$. The condition that each formula in $\mathtt{sat}(P)$ is of bounded width is not enough to guarantee that $p(\mathbf{X})$ has bounded width. Add, e.g., ground rules r to P with increasing arities of predicates. Although the width of formulas in $\mathtt{sat}(P)$ remains constant (no variables are added), the arity of $p(\mathbf{X})$ in Formula (26) increases, thus increasing the width. Hence, the restriction that the arity of predicates in P should be bounded as well.

By Theorem 7.8 and 7.9, one can reduce satisfiability checking of a bound GgP to satisfiability of a μGF-formula with bounded width. The latter can be done in EXPTIME by Theorem 1.2 in [Grädel and Walukiewicz 1999], such that satisfiability checking w.r.t. bound GgPs is in EXPTIME.

The EXPTIME-hardness follows from Theorem 9.2 and the EXPTIME-hardness of CTL satisfiability checking (Theorem 2.5). □

As indicated in [Gottlob et al. 2002], the objects in the database form the states of the Kripke model. In the open domain case, one, intuitively, allows, of extra states in the Kripke model, not explicitly listed in the database.

10. 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. Satisfiability checking w.r.t. (loosely) guarded OASP was shown to be 2-EXPTIME-complete. We defined GgPs, 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 GgPs, and generalized the result that the unique answer set of a stratified program is identical to its least fixed point. We showed how to optimally simulate CTL in OASP.

We plan to extend GgPs to loosely guarded gPs, 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 GeLi-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 [Halevy et al. 2001], 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 GeLi-reduct.

We established the equivalence of open ASP with GgPs, 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 [Gottlob et al. 2002]. Open ASP with GgPs 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.

In [Gottlob et al. 2002] one also discusses the data complexity which is simpler than the combined complexity as studied here. For the future, we also investigate the data complexity of reasoning in guarded Open Answer Set Programming.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Andréka, H., Németi, I., and Van Benthem, J. 1998. Modal Languages and Bounded Fragments of Predicate Logic. *J. of Philosophical Logic 27,* 3, 217–274.

Attie, P. C. and Emerson, E. A. 2001. Synthesis of Concurrent Programs for an Atomic Read/Write Model of Computation. *ACM Trans. Program. Lang. Syst. 23,* 2, 187–242.

Balduccini, M. and Gelfond, M. 2003. Diagnostic reasoning with a-prolog. *Theory and Practice of Logic Programming (TPLP) 3,* 4-5, 425–461.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

Chandra, A. K. and Harel, D. 1982. Horn Clauses and the Fixpoint Query Hierarchy. In *Proc. of PODS '82.* ACM Press, 158–163.

Clark, K. L. 1987. Negation as Failure. In *Readings in Nonmonotonic Reasoning*. Kaufmann, 311–325.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. 1986. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst. 8,* 2, 244–263.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. 2001. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys 33,* 3, 374–425.

Emerson, E. A. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier Science Publishers B.V., 995–1072.

Emerson, E. A. and Clarke, E. M. 1982. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Sciene of Computer Programming 2,* 3, 241–266.

Emerson, E. A. and Halpern, J. Y. 1982. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In *Proc. of the fourteenth annual ACM symposium on Theory of Computing*. ACM Press, 169–180.

Flum, J. 1999. On the (Infinite) Model Theory of Fixed-point Logics. *Models, Algebras, and Proofs*, 67–75.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proc. of International Conference on Logic Programming (ICLP 1988)*. MIT Press, 1070–1080.

Gelfond, M. and Przymusinska, H. 1993. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*. MIT Press, 397–413.

Gottlob, G., Grädel, E., and Veith, H. 2002. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic 3,* 1, 1–35.

Grädel, E. 1999. On the Restraining Power of Guards. *Journal of Symbolic Logic 64,* 4, 1719–1742.

Grädel, E. 2002a. Guarded Fixed Point Logic and the Monadic Theory of Trees. *Theoretical Computer Science 288*, 129–152.

Grädel, E. 2002b. Model Checking Games. In *Proceedings of WOLLIC 02*. Electronic Notes in Theoretical Computer Science, vol. 67. Elsevier.

Grädel, E., Hirsch, C., and Otto, M. 2002. Back and Forth Between Guarded and Modal Logics. *ACM Transactions on Computational Logic 3*, 418–463.

Grädel, E. and Walukiewicz, I. 1999. Guarded Fixed Point Logic. In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS '99)*. IEEE Computer Society, 45–54.

Halevy, A., Mumick, I., Sagiv, Y., and Shmueli, O. 2001. Static Analysis in Datalog Extensions. *Journal of the ACM 48,* 5, 971–1012.

Halpin, T. 2001. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers.

Heymans, S., Van Nieuwenborgh, D., and Vermeir, D. 2005a. Guarded Open Answer Set Programming. In *8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005)*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Number 3662 in LNAI. Springer, Diamante, Italy, 92–104.

Heymans, S., Van Nieuwenborgh, D., and Vermeir, D. 2005b. Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs. In *2nd European Semantic Web Conference (ESWC 2005)*, A. Goméz-Pérez and J. Euzenat, Eds. Number 3532 in LNCS. Springer, Heraklion, Greece, 392–407.

Heymans, S., Van Nieuwenborgh, D., and Vermeir, D. 2006a. Guarded Open Answer Set Programming with Generalized Literals. In *Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006)*, J. Dix and S. Hegner, Eds. Number 3861 in LNCS. Springer, 179–200.

Heymans, S., Van Nieuwenborgh, D., and Vermeir, D. 2006b. Open Answer Set Programming for the Semantic Web. *Journal of Applied Logic*. To appear, also available from `http://tinf2.vub.ac.be/~sheymans/tech/hvnv-jal2006.pdf`.

Huth, M. R. A. and Ryan, M. 2000. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.

Immerman, N. 1986. Relational queries computable in polynomial time. *Information and Control 68,* 1-3, 86–104.

Kozen, D. 1983. Results on the Propositional μ-calculus. *Theor. Comput. Sci. 27*, 333–354.

Lee, J. and Lifschitz, V. 2003. Loop Formulas for Disjunctive Logic Programs. In *Proc. of ICLP 2003*. LNCS, vol. 2916. Springer, 451–465.

Leone, N. and Perri, S. 2003. Parametric connectives in disjunctive logic programming. In *Answer Set Programming*. CEUR Workshop Proceedings, vol. 78.

Lifschitz, V., Pearce, D., and Valverde, A. 2001a. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic 2,* 4, 526–541.

Lifschitz, V., Pearce, D., and Valverde, A. 2001b. Strongly equivalent logic programs. *ACM Trans. Comput. Log. 2,* 4, 526–541.

Lifschitz, V., Tang, L. R., and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 369–389.

Lin, F. and Zhao, Y. 2002. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. of 18th National Conference on Artificial Intelligence*. AAAI, 112–117.

Lloyd, J. and Topor, R. 1984. Making Prolog More Expressive. *J. Log. Program. 1,* 3, 225–240.

Manna, Z. and Wolper, P. 1984. Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst. 6,* 1, 68–93.

Moschovakis, Y. 1974. *Elementary Induction on Abstract Structures*. North Holland.

Osorio, M., Navarro, J. A., and Arrazola, J. 2004. Applications of intuitionistic logic in answer set programming. *TPLP 4,* 3, 325–354.

Osorio, M. and Ortiz, M. 2004. Embedded implications and minimality in asp. In *In Proc. of the 15th International Conference on Applications of Declarative Programming and Knowledge Management and 18th Workshop on Logic programming*. 241–254.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison Wesley.

Schlipf, J. 1993. Some Remarks on Computability and Open Domain Semantics. In *Proc. of the Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*.

Schlipf, J. 1995. Complexity and Undecidability Results for Logic Programming. *Annals of Mathematics and Artificial Intelligence 15,* 3-4, 257–288.

Sistla, A. P. and Clarke, E. M. 1985. The Complexity of Propositional Linear Temporal Logics. *J. ACM 32,* 3, 733–749.

Syrjänen, T. 2004. Cardinality Constraint Programs. In *Proc. of JELIA'04*. Springer, 187–200.

Tarski, A. 1955. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics 5*, 285–309.

Tobies, S. 2001. Complexity Results and Practical Algorithms for Logics in Knowledge Representation. Ph.D. thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany.

Van Benthem, J. 1997. Dynamic Bits and Pieces. In *ILLC research report*. University of Amsterdam.

van Emden, M. H. and Kowalski, R. A. 1976. The Semantics of Predicate Logic as a Programming Language. *Journal of the Association for Computing Machinery 23,* 4, 733–742.

Received March 2006; accepted September 2006