Vrije Universiteit Brussel

Faculteit Wetenschappen
Vakgroep Computerwetenschappen
Theoretische Informatica

# Beslisbaar Open Answer Set Programmeren

Proefschrift ingediend met het oog op het behalen van de wetenschappelijke graad van Doctor in de Wetenschappen

10 februari 2006

Door: Stijn Heymans
Promotor: Prof. Dr. Dirk Vermeir

Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science
Theoretical Computer Science

# Decidable Open Answer Set Programming

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Science

10 February 2006

By: Stijn Heymans
Supervisor: Prof. Dr. Dirk Vermeir

The Cure - A Forest

# Samenvatting

Traditionele paradigma's voor logisch programmeren hebben een *gesloten wereld* veronderstelling: geldige deducties gebruiken enkel die constanten die in het programma voorkomen. Het *grounden* van een logisch programma met zijn eigen constanten verhindert het gebruik van logisch programmeren als conceptuele modelleertaal. Een kennisingenieur zou alle "invloedrijke" constanten moeten voorzien.

*Open answer set programmeren (OASP)* lost dit gebrek aan modulariteit op door toe te laten dat logische programma's geground worden met de elementen van een willekeurige niet-lege aftelbare verzameling die de constanten in het programma omvat. OASP is echter, in het algemeen, onbeslisbaar. Het onbeslisbare domino probleem kan ernaar gereduceerd worden.

Ten einde beslisbaarheid te herwinnen, beperken we de vorm van logische programma's. Dit levert 3 families van logische programma's op, gebaseerd op 3 verschillende reducties:

- *Conceptuele Logische Programma's (CoLPs)* zijn logische programma's met unaire en binaire predikaten (mogelijk omgekeerde binaire predikaten) waar regels een boomstructuur hebben. Beslisbaarheid van het nakijken of er aan een predikaat *voldaan* kan worden, kan getoond worden door een reductie naar het nakijken of er boomstructuren zijn die door een *two-way alternating tree automaton* aanvaard worden.
- *Forest Logische Programma's (FoLPs)* breiden CoLPs uit met constanten en laten omgekeerde binaire predikaten weg. We identificeren fragmenten die een reductie naar eindig answer set programmeren toelaten (i.e., met een gesloten wereld veronderstelling).
- *Guarded Programma's* laten $n$-aire predikaten toe maar beperken het gebruik van negatieve atomen, zoals bv. ongelijkheid $\neq$. Beslisbaarheid van guarded programma's hangt af van een vertaling naar *guarded fixed point logic*, welke gezien kan worden als een uitbreiding van *Clark's completion* met fixed point formules. We breiden guarded programma's verder uit en

tonen aan dat (alternation-free) guarded fixed point logic equivalent is met het resulterende formalisme.

We bespreken de bovenstaande 3 families in detail. In het bijzonder illustreren we hun expressiviteit door hen te relateren aan kennisrepresentatie formalismen zoals *description logics (DLs)*, mogelijk uitgebreid met *DL-safe* regels, *computation tree logic (CTL)*, Datalog LITE, (alternation-free) guarded fixed point logic, en eindig answer set programmeren met $\omega$-restricted programma's. Bovendien integreren logische programma's onder de open answer set semantiek, in één unificerend formalisme, het beste van zowel het logisch programmeren paradigma (nonmonotoniciteit door *negation by failure*) en het DL paradigma (beslisbaar open domein redeneren). Dit maakt OASP een geschikte kandidaat voor *Semantic Web* redeneren.

# Abstract

Traditional logic programming paradigms have a closed world assumption: they make valid deductions using the logic program's constants only. By grounding a logic program with its own constants, the use of logic programming as a conceptual modeling language is severely hampered. A knowledge engineer would need to provide all "influential" constants.

*Open answer set programming (OASP)* solves this lack of modularity by allowing for the grounding of logic programs with an arbitrary non-empty countable superset of the program's constants. However, OASP is, in general, undecidable: the undecidable domino problem can be reduced to it.

In order to regain decidability, we restrict the shape of logic programs, yielding 3 families of logic programs, based on 3 different decidability vehicles:

- *Conceptual Logic Programs (CoLPs)* are logic programs with unary and binary predicates (possibly inverted) where rules have a tree shape. Decidability of satisfiability checking of predicates is shown by a reduction to non-emptiness checking of two-way alternating tree automata.
- *Forest Logic Programs (FoLPs)* extend CoLPs with constants and leave out inverted predicates. We identify fragments that enable a reduction to finite answer set programming (i.e., with a closed world assumption).
- *Guarded Programs* allow for $n$-ary predicates but restrict the use of negative atoms like, e.g., inequality $\neq$. Their decidability depends on a translation from guarded programs to guarded fixed point logic formulas, which can be seen as an extension of Clark's completion with fixed point formulas. We further extend guarded programs with generalized literals and show that (alternation-free) guarded fixed point logic is equivalent to the resulting framework.

We discuss the above 3 families in depth, in particular, we illustrate their expressiveness by relating them to knowledge representation formalisms such as description logics (DLs), possibly extended with DL-safe rules, computation tree logic (CTL), Datalog LITE, (alternation-free) guarded fixed point logic, and finite answer set programming with $\omega$-restricted programs. Moreover,

logic programs under the open answer set semantics integrate, in one unifying framework, the best of both the logic programming paradigm (a flexible rule-based representation and nonmonotonicity by means of negation as failure) and the DL paradigm (decidable open domain reasoning). This makes OASP a viable candidate for Semantic Web reasoning.

# Acknowledgments

I am infinitely indebted to the supervisor of this dissertation, Prof. Dr. Dirk Vermeir. Actually, the term *supervisor* does not do justice to the indispensable role he played. This dissertation benefited greatly from Dr. Davy Van Nieuwenborgh's input. Davy read drafts of this dissertation, yielding a continuous stream of comments and suggestions for improvements. I want to thank Dr. Mustafa Jarrar for thought-provoking discussions on conceptual modeling. The remaining credits go to my parents and brother.

Brussels, *Stijn Heymans*
November 2005

# Contents

# 1

# Introduction

## 1.1 Motivation

In traditional logic programming paradigms a *closed world assumption* holds. In practice, this means that, in order to make valid deductions, one only takes into account the known objects. More specifically, one only considers the constants that are specified in the logic program. Take, for example, a logic program consisting of the following rules

$$\begin{aligned} study(X) \vee not\ study(X) &\leftarrow \\ pass(X) &\leftarrow study(X) \\ fail(X) &\leftarrow not\ pass(X) \\ pass(john) &\leftarrow \end{aligned}$$

This program represents the knowledge that one may study or not, and if one does, one will pass, otherwise one will fail. In particular, we have a fact stating that student *john* passes.

Logic programming paradigms , as, e.g., answer set programming [GL88], will then *ground* the program with all constants that are present in the program, resulting in a program without variables:

$$\begin{aligned} study(john) \vee not\ study(john) &\leftarrow \\ pass(john) &\leftarrow study(john) \\ fail(john) &\leftarrow not\ pass(john) \\ pass(john) &\leftarrow \end{aligned}$$

with *answer sets* $\{pass(john)\}$ and $\{pass(john), study(john)\}$[1], none of them containing a *fail*-atom. One might then conclude, since there is no *fail*-literal in any answer set, that one can never fail, or, formally, that the *fail*-predicate is not satisfiable. However, in the setting where the first three rules of the

[1] Note the effect of $study(X) \vee not\ study(X) \leftarrow$, which freely allows for *john* to study or not. We call such rules *free rules*.

example program are just specifying some general knowledge about studying, passing, and failing, such a conclusion is wrong. Given other instance data than the rule $pass(john) \leftarrow$ the conclusions of the program may be different and individuals can fail (the predicate *fail* is satisfiable). Thus, listing more students in the program might solve the problem in this case. However, in general, this puts a serious burden on the knowledge engineer, having to handle all "influential" constants.

The illustrated behavior of closed world reasoning indicates a lack of modularity, as discussed in [VS93]. In [VS93], it is argued that, like in normal software, "procedures" should be independent of the environment, i.e., adding new procedures to the system should not interfere with the conclusions the already defined procedures make. In essence, procedures should be able to cope with unknown objects, or, in a logic programming context, deductions made by logic programming semantics should be robust against the addition of new constants and should take into account the existence of unspecified, *anonymous* elements.

The lack of modularity was termed the *universal query problem* in [PP90]: take the universal query $\forall X \cdot p(X)$ asking whether for all elements $x$ from some universe, $p$ holds. Given a rule $p(a) \leftarrow$, one has, with a standard Herbrand least model semantics (i.e., under a closed world assumption), that the only model of the rule is $\{p(a)\}$. No further objects than $a$ are considered such that the query $\forall X \cdot p(X)$ holds. However, adding (unrelated) knowledge $q(b) \leftarrow$ one gets the model $\{p(a), q(b)\}$ such that the query no longer holds: $p(b)$ does not hold.

In [VS93], several alternative assumptions for the closed world assumption are discussed, largely independent of any particular logic programming semantics. E.g., assumptions that always assume an infinite number of freshly named elements by adding a rule with a constant and a function symbol such that the Herbrand Universe is always infinite, or, allowing for an arbitrary number of anonymous elements like in a first-order setting.

[GP93] solves the described problem in the context of answer set programming by introducing $k$ new constants, $k$ finite, and grounding the program with this extended universe; the answer sets of the grounded program are called *$k$-belief sets*.

Instead of allowing for extensions of the constants in the program, one can also allow for so-called *open* predicates as in [VB97] for a well-founded semantics or in [Bon03] for an answer set semantics with function symbols. Instead of dropping the closed world assumption altogether as in [VS93] or [GP93], one restricts the closed world reasoning to predicates that are not open, i.e., roughly, the open predicates receive a first-order semantics while the other ones remain closed world (but, of course, by allowing for open predicates the domain is actually open since the open predicates may introduce anonymous elements in the program).

We extend the principle of $k$-belief sets in [GP93] by allowing for arbitrary, thus possibly infinite, non-empty countable supersets of the program's con-

stants, so-called *universes*. *Open answer sets* are then pairs $(U, M)$ with $M$ an answer set of the program grounded with the universe $U$ of $P$. Recapitulating our example, we have that $(\{a, x\}, \{pass(john), fail(x)\})$ is an open answer set of the program. Indeed, the grounding is now w.r.t. $\{a, x\}$ instead of $\{a\}$ where $x$ is a new *anonymous* element:

$$\begin{aligned} study(john) \vee not\ study(john) &\leftarrow \\ pass(john) &\leftarrow study(john) \\ fail(john) &\leftarrow not\ pass(john) \\ study(x) \vee not\ study(x) &\leftarrow \\ pass(x) &\leftarrow study(x) \\ fail(x) &\leftarrow not\ pass(x) \\ pass(john) &\leftarrow \end{aligned}$$

which has an answer set $\{pass(john), fail(x)\}$ such that the predicate *fail* is indeed satisfiable, or, intuitively, there is instance data such that the *fail* predicate can be populated.

Although open answer sets consist of a universe like first-order logic interpretations, the formalism is not a fragment of first-order logic. Since open answer sets are minimal models of their Gelfond-Lifschitz reduct [GL88], i.e., the transformation of a program with *negation as failure* to a program without it, they are capable of expressing concepts that are not expressible in first-order logic. E.g., a program with rules

$$\begin{aligned} p(X, Y) &\leftarrow f(X, Y) \\ p(X, Z) &\leftarrow f(X, Y), p(Y, Z) \end{aligned}$$

expresses that $p$ is the transitive closure of $f$. It is well-known that transitive closure cannot be expressed in first-order logic, see, e.g., [AHV95].

However, as reasoning with $k$-belief sets is already undecidable [Sch93] it comes as no surprise that *open answer set programming (OASP)* is too. We show this by reducing a well-known undecidable problem, the *domino problem*, to satisfiability checking of predicates under an open answer set semantics.[2] In order to regain decidability but still have the desired openness, we will compromise on the shape of programs and look for specific forms of programs for which reasoning under the open answer set semantics is decidable, but which are still expressive enough to represent useful knowledge.

Consequently, the main topic of this dissertation is:

*The identification of interesting decidable classes of logic programs for which reasoning under the open answer set semantics is decidable.*

A promising area of application for open answer set programming is the envisioned *Semantic Web*. The Semantic Web [BLHL01] seeks to improve on

[2] Note that we cannot use the undecidability of reasoning with $k$-belief sets to show undecidability of reasoning with open answer sets, as the latter may be infinite while the former are always finite.

the current World Wide Web, making knowledge not only viewable and interpretable by humans, but also by software agents. Ontologies play a crucial role in the realization of this next generation web by providing a "shared understanding" [UG96] of certain domains.

*Description Logics (DLs)* [BCM+03] constitute a family of logical formalisms that are based on frame-based systems and useful for knowledge representation, e.g., the representation of taxonomies in certain application domains. Its basic language features include the notions of *concepts* and *roles*. Different DLs can then be identified by the set of constructors that are allowed to form complex concepts or roles. Although DLs are being heavily promoted as an ontology language standard (see the ontology language OWL [BvHH+]), they are by no means synonymous for ontology language. Possible alternatives to DL ontologies include, for example, ORM [Hal01] ontologies as illustrated in the DOGMA framework [JM02]. Or, as we will argue, logic programs under an open answer set semantics.

In the context of the Semantic Web, the integration of *rules* and *ontologies* has gained renewed interest, e.g., in [MSS04]. Note that this naming is rather confusing, in the sense that sets of rules (like in logic programming) can be considered to be ontologies as well, in fact, the programs under an open answer set semantics are, syntactically, rule-based, while they are suitable for expressing "ontological" knowledge as well. What is usually meant in the literature with such an integration of rules and ontologies is the integration between a logic programming paradigm and a particular description logic, intended to provide a more powerful framework, see, e.g., [MSS04, GHVD03, AB02, HMS03, Swi04, VBDDS97, LR96, DLNS98, Ros05, ELST04a, EIST05, HSB+04, HPS04b].

More specifically, from the logic programming side, one can, e.g., attempt to retain the nonmonotonicity (typically provided by negation as failure), while from the description logics side exactly the open domain reasoning is one of the interesting features (besides decidability of reasoning). Logic programs under an open answer set semantics naturally combine both of those strongholds in one unifying decidable framework, allowing for both negation as failure and open domains in a rule-based formalism.

## 1.2 Overview of Decidable Fragments

We can place the different types of programs for which satisfiability checking is decidable in 3 categories, based on the used decidability vehicle:

1. The programs for which satisfiability checking is reduced to checking non-emptiness of two-way alternating tree automata (2ATA): *conceptual logic programs*.
2. The programs for which satisfiability checking is reduced to normal finite answer set programming: *(local) forest logic programs* and variations or extensions thereof.

3. The programs for which satisfiability checking is reduced to satisfiability checking in guarded fixed point logic: *guarded programs* and variations or extensions thereof.

In the next subsections, we present a brief overview of the above three types, e.g., indicating how their decidability is established, how they differ from each other, what the resulting complexity of reasoning is, and that they are expressive enough to capture reasoning in other knowledge representation formalisms.

### 1.2.1 Conceptual Logic Programs

Two-way alternating tree automata (2ATA) [Var98] are automata that take infinite labeled trees as input. They either *accept* or *reject* such an infinite tree based on the notion of *accepting run* of the 2ATA on the tree. A run is again a labeled tree that describes the execution of the 2ATA on a given input tree: its root is labeled by the initial state of the 2ATA and the root of the input tree. In general, the nodes of a run are labeled with the state the 2ATA is in together with the node the automaton is scanning. Each successor of a node in the run corresponds to the state and the scanning node of (a copy) of the 2ATA at a next time step. Those transitions from a node to a successor node are constrained by a transition function.

E.g., when the 2ATA is in a state $q$ and reading a label $a$ of a certain node, the transition function $\delta$ can express that the 2ATA should enter state $q_1$ and move to the predecessor node or enter $q_2$ in the first successor and $q_3$ in the third successor as follows:

$$\delta(q, a) = (-1, q_1) \vee ((1, q_2) \wedge (3, q_3)) \ .$$

Note that, intuitively, a 2ATA can "fork" into multiple instances by starting to scan the first and third successor of the current node. The fact that the automaton can go up in the input tree (indicated by $-1$) explains the naming *two-way* and the *alternating* considers the fact that the definition of the transition function may be a positive boolean formula (in normal tree automata, the automaton always forks one version of itself into all of the successors).

An accepting run is a run of the 2ATA on an infinite tree that satisfies the *acceptance condition*. This acceptance condition can indicate which states of the 2ATA must be visited infinitely often or which states cannot be visited infinitely often. E.g., a 2ATA can recognize infinite trees that contain only a finite number of labels containing some symbol $a$.

One of the basic reasoning procedures associated with 2ATAs is *checking non-emptiness*, i.e., given a 2ATA $A$, is there some infinite tree that is accepted by $A$. In [Var98], it is shown that checking non-emptiness of a 2ATA is in EXPTIME w.r.t. the number of states of the 2ATA. 2ATA can, e.g., be used to show decidability (and tight upper complexity bounds) of expressive *description logics (DLs)*, logics for expressing conceptual knowledge [BCM$^+$03],

as is shown in [CGL02]. These decidability results are based on the reduction of satisfiability checking of DL concepts to checking non-emptiness of a constructed 2ATA.

We want to identify a type of logic programs for which satisfiability checking of predicates can be reduced to checking non-emptiness of a constructed 2ATA. Formally, checking whether a predicate $p$ is satisfiable w.r.t. a program $P$ amounts to checking whether there is an open answer set that contains some atom $p(\mathbf{x})$. Intuitively, we construct a 2ATA $A_{p,P}$ such that an open answer set of $P$ that satisfies $p$ can be rewritten as an (infinite) tree that is accepted by $A_{p,P}$ and, vice versa, such that an infinite tree that is accepted by $A_{p,P}$ can be written as an open answer set of $P$ that satisfies $p$.

This requirement – open answer sets of programs can be rewritten as labeled trees – leads us to the definition of *conceptual logic programs (CoLPs)* which satisfy this *tree model property*. Predicates in CoLPs must be unary or binary. Intuitively, unary literals $p(x)$ can be seen as enforcing that the node corresponding to $x$ has $p$ in its label. Similarly, $f(x, y)$ in an open answer set corresponds to a predecessor/successor relation between the nodes associated with $x$ and $y$, where the connecting edge is labeled with $f$.

We do not allow for constants in CoLPs: intuitively, $\{f(x, y), f(y, x)\}$ is a cycle with anonymous elements $x$ and $y$ in an open answer set and can be replaced by $\{f(x, y), f(y, z)\}$, i.e., introduce a new anonymous element $z$ that is a copy of $x$, yielding a tree structure. A similar cycle $\{f(a, y), f(y, a)\}$ for a constant $a$ cannot be removed this way: $a$ is not anonymous so we cannot necessarily replace the $f(y, a)$ by some $f(y, z)$ since $f(y, a)$ may be introduced by a rule with head $f(Y, a)$ which cannot be used to motivate $f(y, z)$ for an anonymous $z$. In the next section, we show how to cope with constants in a program.

Given those restrictions CoLP rules are one of the following types

- *free rules* $a(X) \vee \mathit{not}\ a(X) \leftarrow$ or $f(X, Y) \vee \mathit{not}\ f(X, Y) \leftarrow$ . Such rules allow for the "free" introduction of unary and binary literals, provided other rules do not impose extra constraints.
- *unary rules*, i.e., rules with a unary literal in the head. E.g.,

$$a(X) \leftarrow f(X, Y_1), \mathit{not}\ g(X, Y_2), h(X, Y_2), Y_1 \neq Y_2$$

  expresses that if $x$ and $y_1$ are connected by $f$ (i.e., $f(x, y_1)$ holds), $x$ and $y_2$ are connected by $h$ and $g$ does not hold for that connection, and $y_1$ and $y_2$ are different, then $a$ must hold at $x$. Unary rules have a *branching* or *tree* structure if we regard $X$ as a node and $Y_1$ and $Y_2$ as its successors. We ensure that we can rewrite open answer sets as trees by imposing the existence of a positive connection between each $X$ and $Y_i$, i.e., in the above rule if $h(X, Y_2)$ were missing it would not be a valid CoLP rule.
  Indeed, take a program containing rules[3]

[3] The example is an adaptation of the DL concept $A \sqcap \forall \neg R.\neg A$ which is not satisfiable by tree models, see, e.g., [LS00].

$$q(X) \leftarrow a(X), \mathit{not}\ p(X)$$
$$p(X) \leftarrow \mathit{not}\ r(X, Y), a(Y)$$

In order to make $q$ satisfiable, one needs some $q(x)$ to hold. By minimality of open answer sets, we have that the body of the first rule must be true, i.e., $a(x)$ holds and $p(x)$ does not hold. The latter implies that the body of the second rule cannot be true, i.e., if there is some $y$ such that $r(x, y)$ does not hold, then $a(y)$ cannot hold. Since $a(x)$ holds, we have that $r(x, x)$ must always hold, resulting in a cycle. Hence, open answer sets of the program that satisfy $q$ can never be rewritten as a tree since such a cycle will always arise.

- *binary rules*, i.e., rules with a binary literal $f(X, Y)$ in the head. E.g.,

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

Similarly as for unary rules, we ensure that there is some connection $g(X, Y)$ in the body, avoiding that connections between arbitrary nodes (i.e., not successor/predecessor) are imposed.

- *constraints* $\leftarrow a(X)$ or $\leftarrow f(X, Y)$. Rules with empty head (i.e., the left hand side of $\leftarrow$) are called constraints; they ensure the body (i.e., the right hand side of $\leftarrow$) can never be true. Actually, the constraints of the simple types above can be equivalently replaced by constraints that have a body like in unary rules or binary rules respectively.

Furthermore, we allow for a special type of predicates in CoLPs, *inverted predicates*, which are denoted $f^{\mathrm{i}}$ for a binary predicate $f$. The intuitive meaning of $f^{\mathrm{i}}(x, y)$ is that it holds in an open answer set iff $f(y, x)$ holds, i.e., $f^{\mathrm{i}}$ is indeed the inverse of $f$. Open answer sets that enforce this *inverted world assumption* are called *open answer sets under IWA* and satisfiability checking of predicates that is only interested in open answer sets under IWA is called *satisfiability checking under IWA*. Inverted predicates are conceptually similar to inverted roles in description logics like $\mathcal{SHIQ}$ [HST99] and allow one to express knowledge that has only infinite open answer sets (under IWA). Intuitively, one can write down rules that continually enforce the introduction of new elements, making use of inverted predicates to prohibit the reuse of previously introduced elements (see Example 3.21, pp. 73).

Conceptual logic programs are a type of programs for which the open answer sets can be rewritten as labeled trees such that they can be given as input to 2ATA, and vice versa, labeled trees recognized by an appropriately constructed 2ATA can be rewritten as open answer sets. The constructed 2ATA has a transition function that is in accordance with the rules of the program. E.g., a rule

$$a(X) \leftarrow f(X, Y), b(Y)$$

amounts to transitions specifying that, if $a$ is not in the label of a node, then there can be no $f$-successor of that node where $b$ holds. Such rules take care of satisfaction of rules and we call the associated states *negative states*. The

minimality of open answer sets is guaranteed by rules that ensure that if $a$ is in the label of a node then there must be some $f$-successor of the node where $b$ holds. States that implement the latter behavior will be called *positive states*.

The acceptance condition ensures that positive states cannot appear infinitely on a path. Intuitively, the infinite occurrence of such a positive state would imply that a predicate cannot be finitely motivated, which contradicts with the minimality of (open) answer sets. E.g., for a rule $a(X) \leftarrow f(X, Y), a(Y)$, an $a$ in the label of a node would amount to a $f$-successor where again $a$ holds, and a similar scenario occurs, resulting in an infinite path containing infinitely many times the positive state that should motivate $a$. By the minimality of open answer sets such "infinite" motivations are not allowed.

As noted above, checking non-emptiness of 2ATA is in EXPTIME in the number of states. Since the number of states of the 2ATA that is constructed from a CoLP $P$ is polynomial in the size of $P$, we have that satisfiability checking w.r.t. CoLPs is in EXPTIME as well.

CoLPs turn out to be useful for expressing conceptual knowledge, hence their naming. They can serve as a formalized representation of graphically represented models that result from the conceptual modeling approach *object-role modeling (ORM)* [Hal01]. Moreover, CoLPs can be used to detect and signal inconsistencies in the conceptual models, thus supporting a continuous quality assessment during the conceptual design phase. Advantages of using CoLP for conceptual modeling include modularity: rules can be added independently, e.g., to express complex constraints, while the consistency of the updated scheme can be verified automatically. We do not claim a mapping of the rich language of ORM to CoLPs. However, we will show the translation of a significant part of the allowed ORM constructs to CoLP, illustrating the usefulness of CoLPs for conceptual modeling.

In description logics, *terminological axioms* encode the knowledge that a concept is subsumed by another one. The simple example that, if a child is popular then it has at least three different popular friends, can be expressed as follows:

$$\mathit{PopChild} \sqsubseteq \mathit{Child} \sqcap (\geq 3\ \mathit{friend}.\mathit{PopChild})$$

A set of such axioms is called a *knowledge base*. *PopChild* and *Child* are called *concept names* and *friend* is a role name. The concept expression $(\geq n\ Q.D)$, a *qualified number restriction*, represents all the items for which there are at least $n$ different $Q$-successors that belong to $D$. More specifically, if $x$ belongs to $(\geq n\ Q.D)$ there are $n$ different $y_i$ such that $(x, y_i)$ belongs to $Q$ (or $x$ is connected through $Q$ with $y_i$) and $y_i$ belongs to the concept expression $D$. The *intersection* constructor $\sqcap$ expresses that an item belongs to both operands: if $x$ belongs to $A \sqcap B$ it belongs to both $A$ and $B$. Similar to intersection, $\sqcup$ (*union*) is also a commonly used constructor, as well as *negation* $\neg$, which have their trivial set-theoretic interpretations.

Other widely used DLs constructors are the *exists restriction* $\exists R.C$ and the *value restriction* $\forall R.C$. They can express items that have a rich father: $\exists father.RichPerson$. Such a concept expression represents all the $x$'s that have at least one $y$, connected through the role *father* (i.e., they have at least one father), such that $y$ belongs to the concept *RichPerson*. People having only rich friends can be represented by the value restriction $\forall friends.RichPerson$.[4] Some DLs also allow for the declaration of roles as being transitive, such that, for example, if $x$ is an *ancestor* of $y$, $y$ is an *ancestor* of $z$, then, in case the *ancestor* role is declared to be transitive, $x$ is an ancestor of $z$. Another role constructor is $P^-$, which takes the inverse of a role such that if $(x, y)$ belongs to $P$, $(y, x)$ belongs to $P^-$.

Satisfiability checking of concept expressions w.r.t. a knowledge base, i.e., is their a model of the knowledge base such that the concept expression has a non-empty extension w.r.t. that interpretation, cannot be simulated by finite answer set programming for several DLs, because of the lack of the finite model property: some DL knowledge bases have only infinite models. Such a DL is, for example, $\mathcal{SHIQ}$ [HS98].

Take, for example, the following knowledge base:

$$\begin{aligned} SalesItem &\sqsubseteq Item \sqcap \exists hasPrice \\ Item \sqcap \exists hasPrice &\sqsubseteq SalesItem \end{aligned}$$

The corresponding CoLP defines *SalesItem*, *Item*, and *hasPrice* with free rules

$$\begin{aligned} SalesItem(X) \vee not\ SalesItem(X) &\leftarrow \\ Item(X) \vee not\ Item(X) &\leftarrow \\ hasPrice(X, Y) \vee not\ hasPrice(X, Y) &\leftarrow \end{aligned}$$

and defines the intersection and the exists restriction $\exists hasPrice$ as follows:

$$\begin{aligned} (Item \sqcap \exists hasPrice)(X) &\leftarrow Item(X), \exists hasPrice(X) \\ \exists hasPrice(X) &\leftarrow hasPrice(X, Y) \end{aligned}$$

Finally, we express both DL axioms directly by the constraints,

$$\begin{aligned} \leftarrow SalesItem(X), not\ (Item \sqcap \exists hasPrice)(X) \\ \leftarrow not\ SalesItem(X), (Item \sqcap \exists hasPrice)(X) \end{aligned}$$

In general, $\mathcal{SHIQ}$ reasoning can be polynomially reduced to reasoning w.r.t. CoLPs. Since $\mathcal{SHIQ}$ reasoning is EXPTIME-complete, this yields EXPTIME-hardness for reasoning w.r.t. CoLPs. Together with the EXPTIME-membership for CoLPs, we have EXPTIME-completeness for CoLP reasoning.

[4] Note that belonging to this concept does not imply having any friends, only that if one has friends, they are rich.

### 1.2.2 Forest Logic Programs

In a next phase, we allow for constants in CoLPs, resulting in *Forest Logic Programs (FoLPs)*. Forest Logic Programs are again programs with only unary and binary predicates.

Essentially, FoLP rules are like their corresponding CoLP versions, with the difference that we allow for constants too. The conditions in unary and binary rules hold for variables only, i.e., the constants are effectively ignored. The main difficulty that is introduced by constants is the loss of the tree model property. Open answer sets of FoLPs can no longer be rewritten as tree structured open answer sets. However, they can be rewritten as open answer sets that have a forest structure where a forest is a set of trees. Intuitively, we associate each constant with the root of its own tree.

Take, for example, the FoLP,

$$\begin{aligned} \mathit{profit}(C,P) \vee \mathit{not}\ \mathit{profit}(C,P) &\leftarrow \\ \mathit{daught}(C,D) \vee \mathit{not}\ \mathit{daught}(C,D) &\leftarrow \\ \mathit{good}(C) &\leftarrow \mathit{profit}(C, {>}10\%) \\ \mathit{good}(C) &\leftarrow \mathit{daught}(C,D), \mathit{not}\ \mathit{bad}(D) \\ \mathit{bad}(C) &\leftarrow \mathit{not}\ \mathit{good}(C) \end{aligned}$$

expressing that a company had a good year if either its profits were more than 10 percent of its turnover or if it has a daughter company that did well. The two rules with disjunction may freely introduce *profit* or *daught* tuples.

The program has an open answer set that is a *forest model* consisting of two trees, one with root $x$, and one with a constant root $>10\%$ (which is a single node tree), indicating that $x$ is a good company ($\mathit{good}(x)$) which makes a lot of profit and has a chain of good daughters. In order to have a valid forest structure, links to constants can be kept in the starting node, e.g. $\mathit{profit}(x, {>}10\%)$ can be stored in the label of $x$ as $\mathit{profit}^{>10\%}$ without losing any information. This forest model is depicted in Figure 1.1. Satisfiability checking w.r.t.

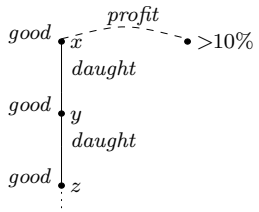


**Fig. 1.1.** Forest Model

CoLPs is shown to be decidable by a reduction to non-emptiness checking of two-way alternating tree automata. However, the definition of FoLPs includes constants and open answer sets can now be rewritten as forests instead of

trees. Since 2ATA take trees as input and not forests, the automata reduction cannot be readily applied and we opt to identify a fragment of FoLPs, *local FoLPs*, that have a *bounded finite model property*, i.e., if there is an open answer set, there is an open answer set with a universe that is bounded by a number of elements that can be specified in function of the program at hand. Since inverted predicates allow the expression of programs that have only infinite open answer sets, we do not allow for inverted predicates in FoLPs. This bounded finite model property allows to reduce satisfiability checking of local FoLPs to finite (normal) answer set programming.

The forest model in Figure 1.1 is infinite, but it can be turned into a finite open answer set by cutting the trees from the moment we have repetition, i.e., when duplicate labels occur on a path, we cut the path below the second occurrence of the label and mimic the outgoing connections of the first node. In the figure, $y$ has the same label "*good*" as $x$ such that we cut the tree below $y$, and, since we have $\mathit{profit}(x, {>}10\%)$ and $\mathit{daught}(x, y)$, we introduce connections $\mathit{profit}(y, {>}10\%)$ and $\mathit{daught}(y, y)$ for $y$. This cutting results in Figure 1.2. However, such a cutting is not possible for arbitrary FoLPs, i.e.,

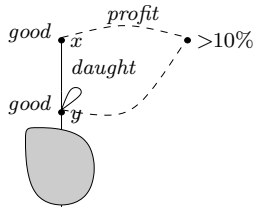


**Fig. 1.2.** Cutting

the cutting does not necessarily result in an open answer set. We identify a basic class for which cutting is possible: *local FoLPs*, which have only negated atoms in the successor part of the tree structure of the unary or binary rules. E.g., we only have the negation *not* $\mathit{bad}(D)$ in the example program. The motivation for particular facts can then be given locally: $\mathit{good}(x)$ is supported by $\mathit{daught}(x, y)$ or $\mathit{profit}(x, {>}10\%)$ with rules $\mathit{good}(x) \leftarrow \mathit{daught}(x, y)$[5] or $\mathit{good}(x) \leftarrow \mathit{profit}(x, {>}10\%)$, involving only $x$, direct successors $y$ of $x$, or a constant. Without locality, cutting the trees may lead to the loss of minimality, e.g., a rule $\mathit{good}(C) \leftarrow \mathit{daught}(C, D), \mathit{good}(D)$ could lead to an answer set

$$\{\mathit{good}(x), \mathit{daught}(x, y), \mathit{good}(y), \mathit{daught}(y, z), \mathit{good}(z), \mathit{profit}(z, {>}10\%)\}$$

and cutting at $y$ would make $\mathit{good}(x)$ unmotivated, leading to a non-minimal model.

[5] The "*not*" in the original body is deleted by the *Gelfond-Lifschitz transformation* [Lif02].

Since the number of different predicates in a program is finite and the branching of forest models is bounded, one can calculate a finite bound $k$ from the program $P$ and a predicate $p$ under consideration only. For this $k$ we then have that for every forest model $(U, M)$ there is an open answer set $(U', M')$ with $|U'| < k$, obtained by using the above cutting technique. As a result, reasoning w.r.t. a local FoLP $P$ can be reduced to finite answer set programming w.r.t. the program $P$ extended with at most $k$ constants. This reduction to finite answer set programming results in a 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ upper bound of reasoning.

We can actually loosen up the syntax of local FoLPs some more by allowing for arbitrary ground rules, or, more formally, let an *extended FoLP (EFoLP)* be the union of a FoLP and an arbitrary ground program with only unary and binary predicates.

Alternatively, one can specify an EFoLP as a pair $(Q, R)$ where $Q$ is a FoLP and $R$ is an arbitrary program containing only unary and binary predicates (possibly with variables) such that $R$ is only ground with constants from $Q \cup R$, and not with anonymous elements. With such arbitrary rules one can then state circular knowledge such as

$$uncle(X, Z) \leftarrow brother(X, Y), father(Y, Z)$$

which states that if for known constants $a, b$ and $c$, $a$ is the brother of $b$ and $b$ is the father of $c$, then $a$ is the uncle of $c$. Such a rule is not a FoLP rule.

EFoLPs have the same nice properties as FoLPs, i.e., a forest model property, and, in case the FoLP part is local, the bounded finite model property. Intuitively, the added rules add arbitrary connections between the roots of the trees in the forest model, but do not interfere with the tree structure of the trees in the forest itself. Complexity of satisfiability checking w.r.t. such local EFoLPs rises, however, to the upper bound 2-EXPTIME$^{\mathrm{NEXPTIME}}$.

Using these local (E)FoLPs as the basis, one can identify further fragments, e.g., *semi-local* and *(free) acyclic* (E)FoLPs. While syntactically different, reasoning w.r.t. to those types can be reduced to reasoning w.r.t. local (E)FoLPs.

If one removes the support for transitive and inverted roles from $\mathcal{SHIQ}$, but adds support for nominals/individuals ($\mathcal{O}$) and intersection and conjunction of roles, one gets the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$. Since FoLPs support constants, we can cope with the nominals in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$. However, FoLPs do not support inverted predicates such that we left out inverted roles from the DL $\mathcal{SHIQ}$. Unfortunately, it does not seem possible to simulate transitive roles in local FoLPs such that we need to leave this out of $\mathcal{SHIQ}$ too.

In [MSS04], DLs are extended with *DL-safe rules*. EFoLPs are an extension of FoLPs with arbitrary rules that can only be ground with constants from the program, which corresponds conceptually with this extension of DL knowledge bases with DL-safe rules. In particular, we can simulate $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ with DL-safe rules by free acyclic EFoLPs.

DL-safe rules do not include the *negation as failure (naf)* operator, and as a consequence, do not cope well with incomplete or dynamically changing

knowledge: like reasoning with DL, reasoning with DL knowledge bases and DL-safe rules is monotonic. However, nonmonotonic reasoning may be useful in closed sub-areas of the Semantic Web as illustrated in the following example. Assume a business is setting up its website for processing customer feedback. It decides to commit to an ontology $\mathcal{O}$ which defines that if there are no complaints for a product, it is a good product, i.e.,

$$good\_product(X) \leftarrow not\ complaint(X)$$

The business has its own particular business rules, e.g.,

$$i : invest(tps, 10K) \leftarrow not\ good\_product(tps)$$

saying that if its particular top selling product *tps* cannot be shown to be a good product, then the business has to invest 10K in *tps*. Finally, the business maintains a repository of dynamically changing knowledge, originating from user feedback collected on the site, e.g., at a certain time the repository contains $R_1 = \{complaint(tps) \leftarrow\}$ with a complaint for *tps*.

If the business wants to know whether to invest more in *tps* it needs to check $\mathcal{O} \cup \{i\} \cup R_1 \models invest(tps, 10K)$, i.e., whether the ontology, combined with its own business rules, and the information repository, demand for an investment or not.

One can use EFoLPs to express the above knowledge. Intuitively, any model of $\mathcal{O} \cup \{i\} \cup R_1$, must verify $complaint(tps)$, and thus $good\_product(X) \leftarrow not\ complaint(X)$ will not trigger and $good\_product(tps)$ will be false, which in turn, with rule $i$, allows to conclude that the business should indeed invest.

Evaluating the same query with an updated repository

$$R_2 = \{complaint(tps) \leftarrow, good\_product(tps) \leftarrow\}$$

containing a survey result saying that *tps* is a good product, no matter what complaints of individual users there may be, leads to $\mathcal{O} \cup \{i\} \cup R_2 \not\models invest(tps, 10K)$, such that no further investments are necessary. Adding knowledge thus invalidates previous conclusions making reasoning nonmonotonic; similar scenarios can easily be imagined in any environment with dynamically changing knowledge.

### 1.2.3 Guarded Programs

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 [GL88]), 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) [GW99] 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 [CH82], Horn clauses were translated to FPL formulas and in [GGV02] 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 set semantics in FPL.

In [LZ02, LL03], 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. However, expressive decidable fragments of FPL have been identified [GW99]: *(loosely) guarded fixed point logic* ($\mu$(L)GF) extends the *(loosely) guarded fragment* (L)GF of FOL with fixed point predicates.

GF is identified in [ANB98] 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 [Ben97], 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 [Grä99], as are their extensions with fixed point predicates $\mu$GF and $\mu$LGF [GW99].

We identify a syntactically restricted class of programs, *(loosely) guarded programs ((L)GPs)*, for which the FPL translation falls in (alternation-free[6]) $\mu$(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.

For example,

$$f(X, Y, Z, b) \leftarrow g(X, Y), h(Y, Z), q(X, Z)$$

is a valid rule of a loosely guarded program since every pair of variables appears together in an atom in the body. Note that constants are allowed and no extra conditions are put on their appearance in rules. Furthermore, compared to (E)FoLPs, $n$-ary predicates are allowed for arbitrary $n$. So, in this respect, (L)GPs are more expressive than (E)FoLPs. However, one cannot express with (L)GPs, e.g., that binary predicates must be functional. Take the CoLP constraint

$$\leftarrow f(X, Y_1), f(X, Y_2), Y_1 \neq Y_2$$

Since $Y_1 \neq Y_2$ is considered equivalent with *not* $Y_1 = Y_2$, we do not have a positive atom in the body that connects $Y_1$ and $Y_2$: the rule is not (loosely) guarded. Thus, (E)FoLPs are more expressive in some ways and less expressive in others than LGPs.

Programs under the normal answer set semantics can be rewritten as LGPs under the open answer set semantics by guarding all variables with atoms that can only deduce constants from the original program. Besides the desirable property that OASP with LGPs is thus a proper decidable extension of normal ASP, this yields that satisfiability checking w.r.t. LGPs is, at least, NEXPTIME-hard.

Datalog LITE [GGV02] 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 devised to ensure linear time model checking while being expressive enough to capture *computational tree logic* [EC82] and alternation-free $\mu$-calculus [Koz83]. Moreover, it is shown to be equivalent to alternation-free $\mu$GF. Our reduction of GPs to alternation-free $\mu$GF 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.

[6] $\mu$(L)GF without nested fixed point variables in alternating least and greatest fixed point formulas.

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 [GGV02], and, since satisfiability checking of GF formulas is 2-EXPTIME-hard [Grä99], 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, $\phi$ is a finite boolean formula and $\psi$ is an atom. Intuitively, such a generalized literal is true in an open interpretation $(U, M)$ if for all substitutions $[\mathbf{Y} \mid \mathbf{y}]$, $\mathbf{y}$ 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 $\phi$ an atom instead of a boolean formula, were introduced in Datalog[7] 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 like it is not feasible to introduce all relevant constants in a program to ensure correct conceptual reasoning, it is not feasible, not even possible, to write knowledge directly as in $r'$ for it has an infinite body. Furthermore, even in the presence of a finite universe, generalized literals allow for a more robust representation of knowledge than would be possible without them. E.g., with critical devices $y_1$ and $y_2$, a rule $s : ok \leftarrow work(y_1), work(y_2)$ does the job as good as $r$ (and in fact $s$ is the GeLi-reduct of $r$), but adding new critical devices, implies revisiting $s$ and replacing it by a rule that reflects the updated situation. Not only is this cumbersome, it may well be impossible as $s$ contains no explicit reference to critical devices, and the knowledge engineer may not have a clue as to which rules to modify.

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

[7] The extension of logic programming syntax with first-order formulas dates back to [LT84].

and all generalized literals are guarded, where a generalized literal is guarded if it can be written as a guarded formula in $\mu$GF. The FPL translation of GgPs then falls into the $\mu$GF 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)* [Eme90], a temporal logic. *Temporal logics* [Eme90] 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)* [Eme90, SC85] and *computation tree logic (CTL)* [Eme90, EH82, CES86], 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 [GL88] 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[8] 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 $\mu$GF, as well as the equivalence of alternation-free $\mu$GF and Datalog LITE [GGV02], lead to the conclusion that alternation-free $\mu$GF, Datalog LITE, and OASP with GgPs, are equivalent, i.e., their satisfiability 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

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

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 most distinct feature of GPs, compared with (E)FoLPs, are its allowing of arbitrary $n$-ary predicates. Usually, DLs only support concepts and binary roles, however, the DL $\mathcal{DLR}$ supports $n$-ary role names. One can simulate a fragment of $\mathcal{DLR}$, called $\mathcal{DLR}^{-\{\leq\}}$, with bound GPs.

We summarized the main complexity results of this dissertation in Table 1.1.

**Table 1.1.** Summary Complexity Results

| Type | Hardness | Membership |
|---|---|---|
| CoLP | EXPTIME (Theorem 3.39) | EXPTIME (Theorem 6.3) |
| Local FoLP | EXPTIME (Theorem 6.10) | 2-EXPTIME$^{\Sigma_2^P}$ (Theorem 4.26) |
| Local EFoLP | EXPTIME (Theorem 6.11) | 2-EXPTIME$^{\text{NEXPTIME}}$ (Theorem 4.35) |
| (L)GP | 2-EXPTIME (Theorem 5.72) | 2-EXPTIME (Theorem 5.28) |
| GgP | 2-EXPTIME (Corollary 5.69 | 2-EXPTIME (Corollary 5.58) |
| bound GgP | EXPTIME (Theorem 5.77) | EXPTIME (Theorem 5.77) |

Table 1.2 contains a summary of the DLs simulations in a decidable class of programs under the open answer set semantics.

**Table 1.2.** Summary Description Logics Simulations

| DL | OASP | Where |
|---|---|---|
| $\mathcal{SHIQ}$ | CoLP | Section 6.1 |
| $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ | acyclic FoLP | Section 6.2 |
| $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ with DL-safe rules | free acyclic EFoLP | Section 6.3 |
| $\mathcal{DLR}^{-\{\leq\}}$ | bound GP | Section 6.4 |

## 1.3 Organization

**Chapter 2** introduces preliminaries to this dissertation such as basic decidability theory, a discussion of undecidable problems such as the domino problem, and an explanation on how to classify decision problems according to their complexity. Next, we introduce the tree data structure and discuss both finite and infinite tree automata. Finally, we introduce four knowledge

representation formalisms that will appear repeatedly throughout this dissertation: answer set programming, description logics, computation tree logic, and fixed point logic.

We define the open answer set semantics for logic programs in Section 3.1 of **Chapter 3** and show in Section 3.2 that for unrestricted programs satisfiability checking for this semantics is undecidable. In Section 3.3, we introduce the notion of inverted predicates and we define an accompanying inverted world assumption. Section 3.4 identifies different syntactical subclasses of logic programs for which reasoning is shown to be decidable by a reduction to 2ATAs. We indicate in Section 3.5 how the restricted programs are still suitable to do conceptual modeling, in particular we show how to simulate a large part of *Object-Role Modeling* constructs. Finally, in Section 3.6, we discuss related work.

In Section 4.1 of **Chapter 4**, we introduce the *forest model property* and define a syntactically restricted class of programs, *forest logic programs (FoLPs)*, satisfying this property. We show in Section 4.2 that a particular type of FoLPs, local FoLPs, has the *bounded finite model property*, which enables a reduction to finite ASP. A type that can be reduced to local FoLPs are the *acyclic FoLPs* from Section 4.3. Section 4.4 identifies an upper bound for the complexity of reasoning. In Section 4.5, we extend FoLPs with an arbitrary finite set of rules that can only be grounded with constants present in the program, resulting in EFoLPs, and show that properties such as the forest model property and the bounded finite model property are valid for suitably restricted classes of EFoLPs.

**Chapter 5** reduces satisfiability checking w.r.t. arbitrary logic programs to satisfiability checking of alternation-free fixed point logic formulas. We identify in Section 5.2 syntactical classes of programs for which this FPL translation falls into the decidable logic $\mu$GF or $\mu$LGF, i.e., guarded or loosely guarded fixed point logic. In Section 5.3, we introduce so-called generalized literals and device a modified translation to FPL in Section 5.4. Section 5.5 mirrors Section 5.2 and identifies classes of programs with generalized literals that can be mapped to guarded FPL. Finally, in Section 5.6, we relate the obtained languages under the open answer set semantics to Datalog LITE which has a least fixed point model semantics.

In Section 6.1 of **Chapter 6**, we reduce satisfiability checking in the $\mathcal{SHIQ}$ DL to satisfiability checking w.r.t. CoLPs, and in Section 6.2, we show how a DL that adds constants and conjunction/disjunction of roles and removes transitive roles from $\mathcal{SHIQ}$, the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, can be simulated by acyclic FoLPs. The DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ extended with DL-safe rules can be simulated using free acyclic EFoLPs as shown in Section 6.3. Section 6.4 describes the DL $\mathcal{DLR}$ which supports $n$-ary relations; a fragment of $\mathcal{DLR}$, so-called $\mathcal{DLR}^{-\{\leq\}}$, can be simulated by bound guarded programs. We discuss in Section 6.5 some of the advantages and disadvantages of using open answer set programming instead of DLs for knowledge representation. We give an overview of related work in Section 6.6.

**Chapter 7** concludes and provides directions for further research.

Part of the results discussed in this dissertation is published in [HV03b, HV03c, HV03a, HVNV04, HVNV05b, HVNV05a, HVNV06].

# 2

# Preliminaries

## 2.1 Decidability, Undecidability, and Complexity

Relying on the exposition in [Pap94], we define Turing Machines as our basic model of computation, show what it means for a problem to be decidable or undecidable, discuss undecidable problems, and explain how to classify decision problems according to their complexity.

### 2.1.1 Decidability and Turing Machines

The main concern of decidability theory is the question "given a problem, is there an algorithm that solves the problem?". We introduce the formal meaning of informal concepts such as *algorithm*, *problem*, and *solves*, using the model of Turing Machines. An example of a problem is *Reachability* [Pap94]:

> Given a graph $G = (V, E)$ and two nodes $x, y \in V$, is there a path from $x$ to $y$?

where a graph $G = (V, E)$ is a pair consisting of a finite set of nodes $V$ and a relation of edges $E \subseteq V \times V$. The reachability problem contains 3 different parameters: a graph $G$, a begin node $x$, and an end node $y$. Instantiating those parameters with actual objects yields an *instance* of the problem. Reachability is a *decision problem* as any instance requires a *yes* or *no* answer: either there is a path or not. An algorithm that solves a decision problem is then, informally, a set of instructions, such that, given an instance of the problem, one gets an answer to the problem. For example, the reachability problem can be solved by taking the begin node, marking it, recursively repeating this marking for all successors, and stopping when there are no more new successors to be marked. If the end node was marked, we answer *yes*, otherwise we answer *no*.

A more formal account of an algorithm is given by the concept of a *Turing Machine (TM)*. A *deterministic Turing Machine (DTM)* is a tuple $(K, \Sigma, \delta, s)$ where $K$ is a finite set of *states*, $\Sigma$ is a finite *alphabet* (a set of symbols),

$\delta : K \times \Sigma \to K \cup \{h, \text{“}yes\text{”}, \text{“}no\text{”}\} \times \Sigma \times \{\leftarrow, \rightarrow, -\}$ is a *transition function*, and $s \in K$ is the *begin state*. A Turing Machine can be seen as a device equipped with a reading head and capable of reading and processing an infinite tape containing symbols from $\Sigma$. Furthermore, we assume $\Sigma$ always contains the symbols $\sqcup$ and $\triangleright$, representing the blank symbol and the start symbol respectively.

The *input* to the machine is a string $x \in (\Sigma \setminus \{\triangleright, \sqcup\})^*$ with the symbol $\triangleright$ pre-pended to $x$, where $X^*$, for a set $X$, is the set of finite strings using elements from $X$. The TM starts computing in the begin state $s$ with the reading head at $\triangleright$. The function $\delta$ can be seen as the program of the machine indicating what the machine is supposed to be doing next: if the machine is in a state $q \in K$ and the reading head is reading a symbol $\sigma \in \Sigma$, then $\delta(q, \sigma) = (q', \sigma', D)$ says that the machine should overwrite $\sigma$ with $\sigma'$, enter state $q'$, and move its reading head in the direction $D \in \{\leftarrow, \rightarrow, -\}$, where $\leftarrow$, $\rightarrow$, or $-$, indicates a move to the left, right, or no move, respectively. The machine knows what to do on any input as $\delta$ is a function, hence the notion *deterministic* TM. Moreover, we assume that for any state $q$, $\delta(q, \triangleright) = (q', \triangleright, \rightarrow)$ for some state $q'$, such that, intuitively, the machine will never read to the left of $\triangleright$, making the infinite tape infinite on the right hand side of $\triangleright$ only, and will start reading the first symbol of its input.

The machine *halts* if one of the three states $h$, “*yes*”, or “*no*” are reached. If it halts in a state “*yes*” (“*no*”), we say the machine *accepts* (*rejects*) the input $x$. If it halts in $h$, the machine is assumed to produce *output* which can be read from the tape as the finite[1] string $y$ following $\triangleright$ whose last symbol is not $\sqcup$ and where only $\sqcup$s appear after $y$ on the tape. Note that it is possible that the machine $M$ does not halt on an input.

Formally, the state of a TM can be described by a *configuration* $(q, w, u) \in K \times \Sigma^* \times \Sigma^*$ indicating the state $q$ the machine is in together with the string $w$ to the left of the reading head (with the position at the reading head included) and the string $u$ to the right of the reading head. A configuration $(q, w, u)$ *yields* $(q', w', u')$ *in one step*, denoted $(q, w, u) \to^M (q', w', u')$ for the DTM $M$, if $\delta(q, \sigma) = (q', \sigma', D)$ where $\sigma$ is the last symbol of $w$ (i.e., the position of the reading head) and,

- if $D = \rightarrow$, then $w'$ is $w$ with $\sigma$ replaced by $\sigma'$ and the first symbol of $u$ appended, and $u'$ is $u$ without its first symbol, or
- if $D = \leftarrow$, then $w'$ is $w$ without $\sigma$, and $u'$ is $u$ with $\sigma'$ pre-pended, or
- if $D = -$, then $w'$ is $w$ with $\sigma$ replaced by $\sigma'$ and $u'$ is $u$.

A configuration $(q, w, u)$ *yields* $(q', w', u')$ *in $k$ steps*, denoted $(q, w, u) \to^{M^k} (q', w', u')$, $k \geq 1$, if there is a $(q, w, u) = (q_0, w_0, u_0) \to^M (q_1, w_1, u_1) \to^M \ldots \to^M (q_k, w_k, u_k) = (q', w', u')$; a configuration $(q, w, u)$ *yields* $(q', w', u')$, denoted $(q, w, u) \to^{M^*} (q', w', u')$, if there is a finite $k$ such that $(q, w, u) \to^{M^k}$

[1] The string $y$ is finite as the machine stopped after a finite number of moves.

$(q', w', u')$. We can then formally reformulate "accepts" and "rejects" in function of configurations: $M$ *accepts* an input $x$ if $(s, \triangleright, x) \rightarrow^{M^*} (\text{“}yes\text{”}, w, u)$ for some strings $w$ and $u$ and $M$ *rejects* $x$ if $(s, \triangleright, x) \rightarrow^{M^*} (\text{“}no\text{”}, w, u)$ for some strings $w$ and $u$. We say that the machine *halts* on $x$ if it either accepts $x$, rejects $x$ or $(s, \triangleright, x) \rightarrow^{M^*} (h, w, u)$ for some strings $w$ and $u$.

*Example 2.1.* Take a DTM $M = (K, \Sigma, \delta, s)$ with $K = \{s, q_0, q_1, q_0', q_1', q\}$, $\Sigma = \{\triangleright, \sqcup, 0, 1\}$, and $\delta$ as in Table 2.1. For a particular input $x \in \{0,1\}^*$, $M$

**Table 2.1.** Transition Function for Palindromes [Pap94]

| $p \in K$, $\sigma \in \Sigma$ | | $\delta(p, \sigma)$ | $p \in K$, $\sigma \in \Sigma$ | | $\delta(p, \sigma)$ |
|---|---|---|---|---|---|
| $s$ | 0 | $(q_0, \triangleright, \rightarrow)$ | $q_0'$ | 0 | $(q, \sqcup, \leftarrow)$ |
| $s$ | 1 | $(q_1, \triangleright, \rightarrow)$ | $q_0'$ | 1 | $(\text{“}no\text{”}, 1, -)$ |
| $s$ | $\triangleright$ | $(s, \triangleright, \rightarrow)$ | $q_0'$ | $\triangleright$ | $(\text{“}yes\text{”}, \sqcup, \rightarrow)$ |
| $s$ | $\sqcup$ | $(\text{“}yes\text{”}, \sqcup, -)$ | | | |
| $q_0$ | 0 | $(q_0, 0, \rightarrow)$ | $q_1'$ | 0 | $(\text{“}no, 1, -)$ |
| $q_0$ | 1 | $(q_0, 1, \rightarrow)$ | $q_1'$ | 1 | $(q, \sqcup, \leftarrow)$ |
| $q_0$ | $\sqcup$ | $(q_0', \sqcup, \leftarrow)$ | $q_1'$ | $\triangleright$ | $(\text{“}yes\text{”}, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_1, 0, \rightarrow)$ | $q$ | 0 | $(q, 0, \leftarrow)$ |
| $q_1$ | 1 | $(q_1, 1, \rightarrow)$ | $q$ | 1 | $(q, 1, \leftarrow)$ |
| $q_1$ | $\sqcup$ | $(q_1', \sqcup, \leftarrow)$ | $q$ | $\triangleright$ | $(s, \triangleright, \rightarrow)$ |

accepts $x$ iff it is a palindrome, a string that can be read both forward and backward, e.g., 10011001, and rejects $x$ otherwise.

Intuitively, the DTM starts by scanning the first symbol of $x$ and remembers it: if it was 0, the DTM enters state $q_0$, if it was 1 it goes in state $q_1$, and if the string is empty it accepts the input. In either $q_0$ or $q_1$ the DTM moves to the end of the string in order to check whether the last element matches the first; a DTM can thus remember a finite amount of information by encoding it in states. In states $q_0'$ and $q_1'$, the DTM is scanning the last element which must match 0 and 1 respectively (the remembered first symbol) if $x$ is to be a palindrome. If the DTM reads 0 in $q_0'$, the palindrome property is not violated such that the DTM removes the last element and goes in state $q$ which brings it back to the beginning of the string (which was moved to the right when in state $s$, such that the beginning is now the original second element of the string). The process starts over with the scanned portion of the string getting smaller on both the left and right hand side. If the DTM reads 1 in $q_0'$, the palindrome property is violated and the DTM immediately enters the rejecting state "$no$". If it reads $\triangleright$ in $q_0'$, the string had an uneven length and the 0 that led to $q_0'$ was the middle element of the string, yielding

a palindrome. If it reads 1 in $q_0'$, the DTM immediately enters the rejecting state "*no*" as this does not match the 0 from the beginning of the string. The case for $q_1'$ is similar.

A set $L \subseteq (\Sigma \setminus \{\sqcup, \triangleright\})^*$ is called a *language*. A DTM $M$ *decides* a language $L$ if the following holds for any $x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$:

- if $x \in L$, then $M$ accepts $x$, and
- if $x \notin L$, then $M$ rejects $x$.

Thus, $M$ knows how to correctly classify every finite string of $(\Sigma \setminus \{\sqcup, \triangleright\})^*$ as an element of $L$ or not.

*Example 2.2.* The language $L$ of palindromes over an alphabet $\{0, 1\}$ is decided by the DTM from Example 2.1.

A DTM $M$ *accepts* a language $L$ if for any $x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$:

- if $x \in L$, then $M$ accepts $x$, and
- if $x \notin L$, then $M$ does not halt on $x$.

*Accepts* is thus a weaker notion than *decides*, as the DTM correctly classifies strings only if they are a member of the language and does not halt otherwise. The important part is that, if the machine has not halted at a certain time $t$ after starting the computation for input $x$, one does not know whether $x$ is in $L$ or not: it may be that the machine halts after $t$ or that it never halts.

If there is some DTM that decides $L$, $L$ is called *recursive*. If there is some DTM that accepts $L$, then $L$ is *recursively enumerable (r.e.)*. The set of r.e. languages encompasses the set of recursive languages as every recursive language is also r.e. [Pap94].

A DTM can be seen as an algorithm for a decision problem by encoding instances of the decision problem as strings. The language $L(d)$, associated with a decision problem $d$, consists then of all encoded instances that have a "yes" answer (*yes-instances*). A DTM $M$ *solves* a decision problem $d$ if $M$ decides $L(d)$, i.e., given an instance $x$ of the decision problem, encoded as a string, $M$ accepts $x$ if it is a yes-instance and rejects it otherwise.

Note that, as is argued in [Pap94], most reasonable string representations of instances differ only polynomially in each others size, e.g., integers can be represented in binary notation or decimal notation. One notable exception, however, is the representation of integers in unary notation which needs exponentially more space than, e.g., a binary representation. E.g., the $n$ in a description logics qualified number restriction $(\leq n\ S.C)$, see Section 2.3.2, is usually assumed to be represented in unary notation $11\ldots1$, i.e., by a string of length $n$, while the binary representation of $n$ has a length in the order of $\log_2 n$.

The transition function $\delta$ of a DTM determines for every possible state $q$ and every possible symbol $\sigma$ in the alphabet, one and only one possible outcome in the form of a new state, the overwriting symbol and a movement

of the reading head. In a *nondeterministic* TM, denoted NDTM, there is no guarantee that every state/symbol combination has an associated outcome, nor is it guaranteed, if there is an outcome, that the outcome is unique. A NDTM is a tuple $(K, \Sigma, \Delta, s)$ where $K$, $\Sigma$ and $s$ are as before, but $\Delta$ is not a function but a relation $\Delta \subseteq (K \times \Sigma) \times (K \cup \{h, \text{“}yes\text{”}, \text{“}no\text{”}\} \times \Sigma \times \{\leftarrow, \rightarrow, -\})$.

For a NDTM $M$, a configuration $(q, w, u)$ *yields* $(q', w', u')$ *in one step*, denoted $(q, w, u) \rightarrow^M (q', w', u')$, if $((q, \sigma), (q', \sigma', D)) \in \Delta$ where the same conditions as in the DTM case hold for $q'$, $w'$, and $u'$. There may be different possible configurations that result from $(q, w, u)$ in one step: since $\Delta$ is no longer a function, $\rightarrow^M$ is not either. The relations $\rightarrow^{M^k}$ and $\rightarrow^{M^*}$ are defined as before. A NDTM $M$ decides a language $L$ if, for every $x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$: $x \in L$ iff $(s, \triangleright, x) \rightarrow^{M^*} (\text{“}yes\text{”}, w, u)$ for some strings $w$ and $u$. Note the difference with DTMs: a NDTM decides a language if for every $x$ there is *some* accepting sequence of nondeterministically chosen configurations (according to $\Delta$). A NDTM $M$ solves a decision problem $d$ if $M$ decides $L(d)$.

An important extension of TMs (both deterministic and nondeterministic), is the *TM with an oracle*. Intuitively, the oracle is a subroutine which the TM can call in unit time. We consider an oracle to be equivalent to a decision problem $d$, such that a call to an oracle amounts to checking whether an instance of $d$ is a yes-instance or not. Note that a TM $M$ with oracle $d$ can be either deterministic or nondeterministic. A TM with oracle solves a decision problem similarly as usual but with an oracle at its disposition. For a more formal account of TMs with oracle, we refer to [Pap94].

Finally, we use TMs to define *decidability* of decision problems, the central topic of a large part of this dissertation: a decision problem is *decidable* if there exists a (N)DTM that solves the problem; it is *decidable w.r.t. an oracle* $d$ if there exists a (N)DTM with oracle $d$ that solves the problem.

### 2.1.2 Undecidability and the Domino Problem

A decision problem is *undecidable* if it is not decidable, i.e., there is no (N)DTM that solves the problem. Since a decision problem that is solved by a NDTM, can be a solved by a DTM as well – possibly taking an exponential time longer than the NDTM [Pap94] – a decision problem is undecidable if there is no DTM that solves the problem. In this subsection, we discuss two undecidable problems: the *halting problem* and the *domino problem*. The former mainly to show undecidability of the latter, and the domino problem itself to prove undecidability of satisfiability checking in unrestricted open answer set programming in Section 3.2.

#### The Halting Problem

The halting problem is the following problem [Pap94]:

> Given the description of a DTM $M$ and its input $x$, will $M$ halt on $x$?

This is an undecidable problem; we sketch the proof as in [Pap94].

**Theorem 2.3.** *The halting problem is undecidable.*

*Proof Sketch.* Assume it is not, i.e., there is a DTM $N$ that decides $L(h)$, where $h$ is the halting problem, and, per definition of languages for decision problems,

$$L(h) = \{M; x \mid M \text{ halts on } x, M \text{ a DTM}\} .$$

Define the DTM $D$ such that on input $M$, where $M$ is a string representation of the equally named DTM, $D$ simulates $N$ on input $M; M$, until it is about to halt (which will happen since $N$ decides $L(h)$). If $N$ accepts $M; M$, $D$ enters a state that moves the reading head to the right forever (and thus does not halt). If $N$ rejects $M; M$, then $D$ halts and accepts $M$.

We show that this gives rise to a contradiction. Either $D$ halts on input $D$, or $D$ does not halt on $D$. In the former case, by the construction of $D$, $N$ rejects $D; D$ such that $D; D \notin L(h)$ (since $N$ decides $L(h)$), and thus, by definition of $L(h)$, $D$ does not halt on $D$, a contradiction. In the latter case, by construction of $D$, $N$ accepts $D; D$, such that $D; D \in L(h)$ and thus $D$ halts on input $D$, again a contradiction.

Thus, there is no DTM that decides $L(h)$, and the halting problem is undecidable. □

A variant of the halting problem is the halting problem on DTMs with empty input $h'$:

Given the description of a DTM $M$, will $M$ halt on $\varepsilon$?

where $\varepsilon$ denotes the empty string. For a DTM $M$ with input $x$, define $e(M; x)$ as the DTM that overwrites its input with $M; x$, goes back to $\triangleright$, and starts executing $M$ on $x$[2]. The DTM $e(M; x)$ halts iff $M$ halts on $x$.

**Theorem 2.4.** *The halting problem on DTMs with empty input is undecidable.*

*Proof Sketch.* Assume it is not, then there exists a DTM $N'$ that decides

$$L(h') = \{M \mid M \text{ halts on } \varepsilon, M \text{ a DTM}\} .$$

Define a DTM $N$ on input $M; x$ that simulates $N'$ on $e(M; x)$.

We show that $N$ decides $L(h)$. Take a $M; x \in L(h)$, then $M$ halts on $x$, by definition of $L(h)$, such that $e(M; x)$ halts on $\varepsilon$. Thus, $e(M; x) \in L(h')$, and, since $N'$ decides $L(h')$, $N'$ accepts $e(M; x)$. Since $N$ simulates $e(M; x)$ on input $M; x$, $N$ accepts $M; x$.

Take a $M; x \notin L(h)$, then $M$ does not halt on $x$, by definition of $L(h)$, such that $e(M; x)$ does not halt on $\varepsilon$. Thus, $e(M; x) \notin L(h')$, and, since $N'$ decides $L(h')$, $N'$ rejects $e(M; x)$. Since $N$ simulates $e(M; x)$ on input $M; x$, $N$ rejects $M; x$. □

[2] $e(M; x)$ is a so-called *universal TM*, i.e., a TM that takes as input the description of another TM $M$ together with an input $x$, and executes $M$ on $x$. For more details, we refer the reader to [Pap94].

**The Domino Problem**

We define the *origin constrained domino problem* and show undecidability of it by a reduction from the halting problem along the lines of [BGG97].

Intuitively, the domino (or tiling) problem asks whether, given a set of dominoes, there is a tiling of the plane $\mathbb{N} \times \mathbb{N}$ using (infinitely many) copies of the available dominoes. Formally, a *domino system* is a tuple $(D, H, V)$ where $D$ is a finite set of dominoes and $H \subseteq D \times D$ ($V \subseteq D \times D$) indicates how the dominoes must be positioned horizontally (vertically). A domino system $(D, H, V)$ *tiles* the plane $\mathbb{N} \times \mathbb{N}$ if there exists a *tiling function* (or *tiling* for short) $\tau : \mathbb{N} \times \mathbb{N} \to D$ of the plane $\mathbb{N} \times \mathbb{N}$ such that, for all $(x, y) \in \mathbb{N} \times \mathbb{N}$,

- $(\tau(x, y), \tau(x + 1, y)) \in H$, and
- $(\tau(x, y), \tau(x, y + 1)) \in V$,

i.e., horizontally (vertically) adjacent positions must be in $H$ ($V$): a domino $d_1$ may be tiled on the left of (below) $d_2$ if the right (upper) side of $d_1$ matches the left (lower) side of $d_2$ ($(d_1, d_2) \in H$, $(d_1, d_2) \in V$ respectively).

The *domino problem* is then

> Given a domino system $\mathcal{D}$, does it tile the plane $\mathbb{N} \times \mathbb{N}$?

In the related *origin constrained domino problem*, we have the additional condition that a particular domino $d$ has to be present in the tiling[3], where a domino is present in a tiling $\tau$ if there is some $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that $\tau(x, y) = d$:

> Given a domino system $\mathcal{D}$ and a domino $d \in D$, does $\mathcal{D}$ tile the plane $\mathbb{N} \times \mathbb{N}$ such that $d$ is present in the tiling?

We sketch the undecidability of the origin constrained domino problem by reducing the halting problem for DTMs on empty input to it.

In the following, we assume that the possible movements of a DTM are $\leftarrow$ and $\rightarrow$ (so we leave out $-$). It is easy to see that this does not restrict its expressiveness, i.e., $-$ can be simulated by $\leftarrow$ and $\rightarrow$.

For a DTM $M = (K, \Sigma, \delta, q_0)$, we construct a domino system $\mathcal{D}$ and take a domino $d$ from $\mathcal{D}$ such that $M$ does not halt on empty input iff $\mathcal{D}$ tiles the plane such that $d$ is present in the tiling.

Intuitively, if $M$ does not halt on an empty input, the computation of configurations $(s, \triangleright, \varepsilon) \to^M (q_1, w_1, u_1) \to^M (q_2, w_2, u_2) \to^M \ldots$ is infinite; we choose the domino system $\mathcal{D}$ such that each configuration is encoded as a row in the tiling of the plane. For each such row in a tiling, the row above it represents the next configuration, and a non-halting DTM corresponds to a tiling of the plane. The particular domino $d$ that has to be present in the tiling corresponds to the the initial configuration.

[3] The name *origin* constrained domino problem is historical; the domino $d$ can appear anywhere in the tiling, but one can see that any tiling containing $d$ defines also a tiling with $d$ in the origin of the plane.

The other way around, if there is a tiling containing the particular domino $d$, the rows of the tiling correspond to an infinite computation of the TM where the row containing $d$ represents the initial configuration.

Formally, we introduce for every element $s_k \in \Sigma$ an *alphabet domino* such as in Figure 2.1; an alphabet domino on a particular position in a tiling of the plane corresponds with the symbol $s_k$ on the tape in the configuration that corresponds with the row. Note that we do not explicitly define $H$ and $V$ for our domino system $\mathcal{D}$. Instead, we assume that dominoes can only be matched in correspondence with their label/edge drawing. E.g., the domino in Figure 2.1 can be matched on its left side by any domino having a blank right side (likewise on the right), on its upper side by a domino that has on its lower side a label $s_k$ together with the start of an edge, and on its lower side by a domino that has on its upper side the label $s_k$ and an arriving edge; one can, for example, always tile the plane with one alphabet domino.[4]

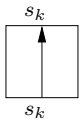


**Fig. 2.1.** Alphabet Domino

For each combination of a state $q_i$ and a symbol $s_j$, we have the *merging dominoes* in Figure 2.2. They read the current symbol $s_j$ from the previous configuration (row). The $q_i$ arrow, coming from the left in the first merging domino in Figure 2.2 indicates that in the previous configuration the machine had to go to the right and in a state $q_i$. The merging domino merges this information and indicates (at its upper side) that this row is in state $q_i$ and reading $s_j$.

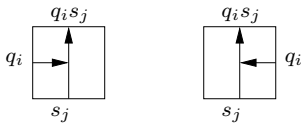


**Fig. 2.2.** Merging Dominoes

[4] Note that this, intuitively, shows why the proposed domino system only works for the origin constrained domino problem: we cannot derive an infinite sequence of configurations from a tiling consisting of only the alphabet domino. The particular tile that has to be present in the tiling (the *origin*) is chosen such that it corresponds with the initial configuration of the DTM and enforces a tiling such that the tiling of the plane corresponds with an infinite sequence of configurations.

The *action dominoes* are the left and right ones in Figure 2.3 for each $\delta(q_i, s_j) = (q_l, s_k, \leftarrow)$ and $\delta(q_i, s_j) = (q_l, s_k, \rightarrow)$ respectively. Intuitively, if the previous row contains a merging domino indicating that the DTM is in state $q_i$ and reading $s_j$, then the action domino propagates the outcome state to the right or to the left, and it overwrites $s_j$ with $s_k$, i.e., the tile has $s_k$ on its upper side instead of the $s_j$ on the previous row.

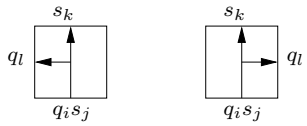


**Fig. 2.3.** Action Dominoes

The 2 dominoes in Figure 2.4 are used to encode the initial configuration of the DTM; the left one indicates that the initial state is $q_0$ while reading $\triangleright$, and the right domino is meant to fill up the rest of the row. We have then

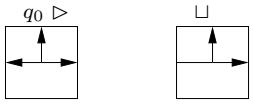


**Fig. 2.4.** Domino's for the Initial Configuration

constructed the domino system $\mathcal{D}$ and we take the left domino from Figure 2.4 as our dedicated domino $d$, such that remains to prove:

**Theorem 2.5.** *Let $M$ be a DTM, and $\mathcal{D}$ the corresponding domino system with domino $d$ selected as described above. Then, $M$ does not halt on empty input iff $\mathcal{D}$ tiles the plane such that $d$ is present in the tiling.*

*Proof Sketch.* Assume $M$ does not halt on empty input. Then there is an infinite sequence $(q_0, \triangleright, \varepsilon) \rightarrow^M (q_1, w_1, u_1) \rightarrow^M (q_2, w_2, u_2) \rightarrow^M \ldots$, and one can construct a tiling $\tau$ by positioning the left domino of Figure 2.4 at the origin of the plane, and filling up the rest of this first row with copies of the right domino in Figure 2.4. The rows above the row that corresponds with the initial configuration, are tiled according to the configurations $(q_1, w_1, u_1), \ldots$ Instead of giving a formal definition, assume for example, that we have the infinite sequence: $(q_0, \triangleright, \varepsilon) \rightarrow^M (q_1, \triangleright \sqcup, \varepsilon) \rightarrow^M (q_2, \triangleright, s_1) \rightarrow^M \ldots$, which resulted from applications $\delta(q_0, \triangleright) = (q_1, \triangleright, \rightarrow), \delta(q_1, \sqcup) = (q_2, s_1, \leftarrow), \ldots$ This sequence of configurations tiles the plane as in Figure 2.5.

For the other direction, assume there is a tiling that contains $d$. There can be no row below $d$ since there are no dominoes with an upper side that

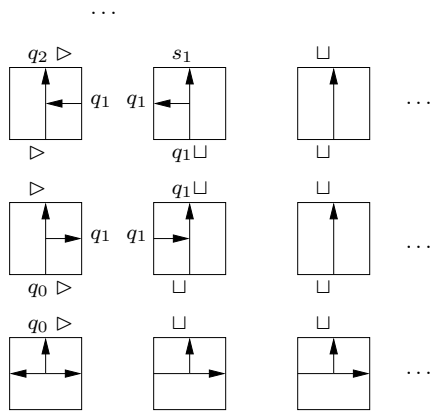


**Fig. 2.5.** Tiling of Plane

is blank to match the lower side of $d$. Thus $d$ resides on the first row of the plane. There are no matches for the left side of $d$ - the only ones that would match the type of arrow are the right merging dominoes, however, they are labeled with a state, and thus not suitable either. Thus, $d$ is at the origin of the plane. The dominoes on the right of $d$ can only be copies of the right domino from Figure 2.4, such that the first row of a tiling of the plane is exactly as in Figure 2.5.

The domino above $d$ must be an action domino (since the upper side of $d$ must match the lower side of this domino). On the right there must be a merging domino. One can continue this reasoning to fill the whole plane. The corresponding configurations can be read from the upper sides of each row. Note that on each row there is only one domino with both a state and a symbol on its upper side. This yields an infinite sequence of configurations such that $M$ does not halt. □

Since the halting problem is undecidable, the origin constrained domino problem is too.

**Corollary 2.6.** *The origin constrained domino problem is undecidable.*

The unconstrained domino problem is undecidable as well. The proof is, however, considerably harder than the constrained case: it uses an aperiodic tiling of the plane with Robinson's dominoes, and then, as in the constrained case, shows that a TM does not halt iff there is a tiling of the plane. Since the constrained domino problem is sufficient for showing undecidability of satisfiability checking w.r.t. unrestricted programs under the open answer set semantics, see Section 3.2, we refer to [BGG97] for a full account of the proof of the undecidability of the unconstrained domino problem.

### 2.1.3 Complexity

We introduce the theory of complexity classes along the lines of [Pap94]. For a DTM $M$ with input $x$, if $(s, \triangleright, x) \rightarrow^{M^t} (q, w, u)$ where $q \in \{\text{“}yes\text{”}, \text{“}no\text{”}, h\}$, then the *time required by $M$ on $x$* is $t$; if $M$ does not halt on $x$, the time required is $\infty$. For a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we say that a DTM $M$ *operates in time* $f(n)$ if for any input $x$ the time required by $M$ on $x$ is at most $f(|x|)$ where $|x|$ is the length of $x$.

A NDTM $M$ operates in time $f(n)$ , if, for any input $x$, the following holds: if $(s, \triangleright, x) \rightarrow^{M^t} (q, u, w)$, then $t \leq f(|x|)$. Thus, for a NDTM to operate in time $f(n)$, *any* computation path should fall within the limits imposed by $f$.

A *complexity class* $\mathcal{C}$ is a set of languages. E.g., $\textsc{time}(f(n))$ is the complexity class of languages that can be decided by DTMs that operate in time $f(n)$. Alternatively, viewing a decision problem $d$ as the language $L(d)$, a complexity class is a set of decision problems and $\textsc{time}(f(n))$ is the set of decision problems that can be solved by DTMs that operate in time $f(n)$. Some examples of $\textsc{time}(f(n))$ for particular $f$ are P for a polynomial $f$, EXPTIME for an exponential $f$, and 2-EXPTIME for a double exponential $f$. In particular,

$$\textsc{exptime} \equiv \bigcup_{k \in \mathbb{N}} \textsc{time}(2^{n^k}) \ ,$$

and

$$\text{2-}\textsc{exptime} \equiv \bigcup_{k \in \mathbb{N}} \textsc{time}(2^{2^{n^k}}) \ .$$

The complexity class $\textsc{ntime}(f(n))$ is defined as the set of all languages (decision problems) that can be decided (solved) by NDTMs that operate in time $f(n)$. Some examples are NP, NEXPTIME, 2-NEXPTIME, i.e., the set of problems that can be solved by a NDTM in polynomial, exponential, double exponential time respectively. Note that, since a DTM is a special case of a NDTM (with a transition function instead of relation), $\textsc{p} \subseteq \textsc{np}$. The other direction, $\textsc{np} \subseteq \textsc{p}$, is an open problem, generally believed not to hold.

Let $\mathcal{D}$ be a complexity class, then we denote with $\textsc{time}(f(n))^{\mathcal{D}}$ the set of decision problems that can be solved in time $f(n)$ by DTMs with an oracle in $\mathcal{D}$. Similarly, $\textsc{ntime}(f(n))^{\mathcal{D}}$ is the set of decision problems that can be solved in time $f(n)$ by NDTMs with an oracle in $\mathcal{D}$. E.g., $\textsc{np}^{\textsc{np}}$, also denoted as $\Sigma_2^{\textsc{p}}$, is the set of decision problems that are decidable in polynomial time by NDTMs with an oracle in NP.

A decision problem $d$ can be *polynomially reduced* by a *reduction function* $\psi$ to a decision problem $d'$ if the following holds: for all instances $x$ of $d$, $x$ is a yes-instance of $d$ iff $\psi(x)$ is a yes-instance of $d'$, and the size of $\psi(x)$, as a string, is polynomial in the size of $x$. A decision problem $d$ is *$\mathcal{C}$-hard* for a complexity class $\mathcal{C}$ if every decision problem $d' \in \mathcal{C}$ can be polynomially reduced to $d$. If, additionally, $d \in \mathcal{C}$, we call $d$ a *$\mathcal{C}$-complete* decision problem. Note that $\mathcal{C}$-hardness of a decision problem $d$ can be shown by a polynomial reduction from a $\mathcal{C}$-hard problem $d'$ to $d$.

## 2.2 Trees and Tree Automata

In this section, we introduce trees and discuss both finite and infinite tree automata. Such automata will prove useful in showing decidability of satisfiability checking under the open answer set semantics for certain restricted classes of programs, see Section 3.4.

### 2.2.1 Trees

For a $x \in \mathbb{N}_0^*$, i.e., a finite sequence of natural numbers (excluding 0), we denote the concatenation of a number $c \in \mathbb{N}_0$ to $x$ as $x \cdot c$, or, abbreviated, as $xc$. Formally, a *(finite) tree* $T$ is a (finite) subset of $\mathbb{N}_0^*$ such that if $x \cdot c \in T$ for $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, we have that $x \in T$. Elements of $T$ are called *nodes* and the empty word $\varepsilon$ is the *root* of $T$. For a node $x \in T$ we call $x \cdot c \in T$, $c \in \mathbb{N}_0$, *successors* of $x$. By convention, $x \cdot 0 = x$ and $(x \cdot c) \cdot -1 = x$ ($\varepsilon \cdot -1$ is undefined). If every node $x$ in a tree has either 0 or $k$ successors we say that the tree is *$k$-ary*; a *complete* tree $T$ is such that $\forall xj \in T, 1 \leq i < j \cdot xi \in T$. In the following, we assume, unless specified otherwise, that trees are complete. E.g., $T_1 = \{\varepsilon, \varepsilon 1, \varepsilon 2, \varepsilon 11\}$ is a finite complete tree with root $\varepsilon$, two successors $\varepsilon 1$ and $\varepsilon 2$, and $\varepsilon 11$ a successor of $\varepsilon 1$; $T_1$ will also be written as $\{\varepsilon, 1, 2, 11\}$. A *path* $P$ in a tree $T$ is a prefix-closed subset of $T$ such that $\forall x \neq y \in P \cdot |x| \neq |y|$, e.g., $\{\varepsilon, 1, 11\}$ is a path in $T_1$. The *length* of a path is the number of elements of the path, e.g., the path $\{\varepsilon, 1, 11\}$ has length 3.

A *labeled tree* is a pair $(T, t)$ where $T$ is a tree and $t : T \to \Sigma$ is a labeling function; usually we will identify the tree $(T, t)$ with $t$ and we will write $t_x$ for trees where the root is identified with some symbol $x$: if the root in $T_1$ is identified with a symbol $\phi$, we write it as $\{\phi, \phi 1, \phi 2, \phi 12\}$, and a labeling function for $T_1$ is denoted as $t_\phi$. Often – to make the notation uniform when dealing with identified roots – we write $t_\varepsilon$ if there is no symbol associated with the root.

*Example 2.7.* The tree $T = \{\varepsilon, 1, 2, 11, 12, 21, 22\}$ is a finite binary complete tree. We label it with labels from $\Sigma = \{a, b\}$: $t(\varepsilon) = t(2) = t(11) = t(21) = t(22) = a$ and $t(1) = t(12) = b$, and we depict the tree such as in Figure 2.6. If the root of $T$ is identified with a symbol $\phi$, the labeled tree is depicted such

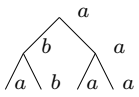


**Fig. 2.6.** Labeled Tree

as in Figure 2.7.

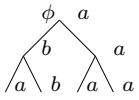


**Fig. 2.7.** Labeled Tree with Identified Root

The *frontier* of a $k$-ary finite tree $T$ is the set

$$\mathrm{fr}(T) \equiv \{x \in T \mid \forall 1 \leq i \leq k \cdot xi \notin T\} .$$

The *outer frontier* of a $k$-ary finite tree $T$ contains the nodes just past the tree

$$\mathrm{fr}^+(T) \equiv \{xi \mid 1 \leq i \leq k, x \in fr(t)\} .$$

We define $T^+ \equiv T \cup \mathrm{fr}^+(T)$, i.e., the tree extended with the nodes past its frontier.

*Example 2.8.* For the tree $T$ from Example 2.7, $\mathrm{fr}(T) = \{11, 12, 21, 22\}$ and $\mathrm{fr}^+(T) = \{111, 112, 121, 122, 211, 212, 221, 222\}$.

We define a partial[5] order $\leq$ on a tree $T$ such that for $x, y \in T$, $x \leq y$ iff $x$ is a prefix of $y$, or, equivalently, there is a path $P$ in $T$ with $x, y \in P$ and $|x| \leq |y|$. As usual, $x < y$ if $x \leq y$ and $y \not\leq x$ such that $<$ is a strict partial order on $T$. We denote the *subtree* of $T$ at $x \in T$ by $T[x]$, i.e.,

$$T[x] \equiv \{y \in T \mid x \leq y\} .$$

The above definitions can be easily extended for labeled trees $t : T \to \Sigma$, e.g., the subtree of $t$ at $x \in T$ is $t[x] : T[x] \to \Sigma$ such that $t[x](y) \equiv t(y)$ for $y \in T[x]$.

*Example 2.9.* For the tree $T$ from Example 2.7, we have, e.g., $1 \leq 1$, $1 < 11$, $1 < 12$, $\varepsilon < 22$, ... The subtree of $T$ at 2 is $T[2] = \{2, 21, 22\}$.

For a finite tree $t : T \to \Sigma$, a tree $s : S \to \Sigma$, and a symbol $a \in \Sigma$, we denote with $t \cdot_a s$, the tree $t$ with every node on $\mathrm{fr}(t)$ with label $a$ replaced by $s$. For example, take $t$ and $s$ such as in Figure 2.8. The *concatenation* $t \cdot_a s$ is the tree in Figure 2.9. If $t$ is an infinite tree, the first (w.r.t. $<$) occurrence of $a$ on each path is replaced (instead of the $a$'s on the frontier). Infinitely repeating such a concatenation is denoted by $t \cdot_a s^\omega$. For the same $t$ and $s$ as above, we have that $t \cdot_a s^\omega$ is as in Figure 2.10.

A *forest* $\mathsf{F}$ is a finite set of trees.

[5] A *partial order* on a set $X$ is a relation on $X$ that is reflexive, anti-symmetric, and transitive. It is a *strict partial* order if it is anti-reflexive and transitive (anti-symmetry is entailed).


**Fig. 2.8.** Trees $t$ and $s$

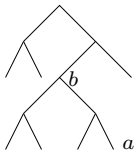


**Fig. 2.9.** Concatenation of Trees

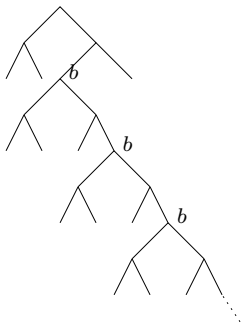


**Fig. 2.10.** Infinite Concatenation

### 2.2.2 Finite Tree Automata

During this section and a large part of the next one, we assume that (labeled) trees are binary and complete. Definitions and results can be extended to the $k$-ary case.

We introduce *nondeterministic finite tree automata* as in [Tho90]. A *nondeterministic finite tree automaton (NFTA)* over an alphabet $\Sigma$ is a tuple $A = (\Sigma, Q, Q_0, \Delta, F)$, where $Q$ is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation. A *run* of $A$ on the finite binary tree $t : T \to \Sigma$ is a tree $r : T^+ \to Q$ where $r(\varepsilon) \in Q_0$, and $(r(x), t(x), r(x1), r(x2)) \in \Delta$ for each $x \in T$. Intuitively, the automaton starts scanning the root of the tree $t$ in an initial state, e.g., in state $q_0$ with $t(\varepsilon) = a$. It then checks its transition relation for occurrences $(q_0, a, q_1, q_2)$, if such a transition is present, the automaton may start two copies of itself, one in state $q_1$ and with the subtree at the first successor as new input, and one in state $q_2$ with the subtree at the second successor as new input; the run is a tree that keeps track of this

behavior by recording the states that are visited by copies of the automaton. A run is an *accepting run* if $r(x) \in F$ for all $x \in \mathrm{fr}^+(t)$. A NFTA accepts a labeled tree if and only if there exists a run that is accepting. We denote the set of trees that are accepted by a particular automaton $A$ as $L(A)$ (the language of $A$).

*Example 2.10.* Take a NFTA $A = (\Sigma, Q, Q_0, \Delta, F)$ with $\Sigma = \{a, b\}$, $Q = \{q_{even}, q_{odd}, q\}$, $Q_0 = \{q_{even}\}$, $F = \{q\}$, and

$$\Delta = \{(q_{even}, a, q_{odd}, q_{odd}), (q_{even}, b, q_{odd}, q_{odd}), (q_{even}, b, q, q), \\ (q_{odd}, a, q_{even}, q_{even}), (q_{odd}, b, q_{even}, q_{even})\} .$$

The NFTA will accept precisely those finite trees for which all leaves are at even depth and have a symbol $b$ as label.

The main decision problem we associate with tree automata is the *non-emptiness problem*:

Given a NFTA $A$, is $L(A) \neq \emptyset$?

or, equivalently, does $A$ accept trees?

**Theorem 2.11 ([Tho90]).** *The non-emptiness problem for NFTAs is decidable.*

*Proof Sketch.* Take a NFTA $A = (\Sigma, Q, Q_0, \Delta, F)$. If $L(A) \neq \emptyset$, then, for $|Q| = n$, there is always a tree in $L(A)$ that has depth at most $n$. Indeed, take an arbitrary tree $t$ that is accepted by $A$: if the depth of $t$ (the maximal length of nodes in the tree) is at most $n$, we are done, otherwise, there is a path in the corresponding run that contains two nodes $x < y$ for which $r(x) = r(y) = q \in Q$. One can then construct a new tree $t'$ with a new corresponding accepting run $r'$ by replacing $t[x]$ by $t[y]$ and $r[x]$ by $r[y]$. The tree $t'$ has now strictly less nodes than $t$. One repeats this process until the resulting $t'$ has depth at most $n$.

In order to check non-emptiness, one can construct all finite trees with depth at most $n$ and check whether there is some tree that is accepted by the NFTA. □

### 2.2.3 Infinite Tree Automata

Whereas Subsection 2.2.2 introduced automata on finite trees, we define in this subsection two types of automata on infinite trees: *Rabin tree automata (RTAs)* and *two-way alternating tree automata (2ATAs)*. We show decidability of the non-emptiness problem for RTAs as in [Tho90] by means of a direct proof (by induction on the number of live states). Decidability of the non-emptiness problem for 2ATAs is shown as in [Var98] by a reduction to RTAs.

**Rabin Tree Automata**

For an infinite path $\sigma$ in a tree $t$, define $\mathrm{In}(\sigma)$ as the set of labels that appear infinitely often on the path $\sigma$. A *Rabin tree automaton (RTA)* over an alphabet $\Sigma$ is a tuple $A = (\Sigma, Q, q_0, \Delta, \Omega)$ with $Q$ a set of states, $q_0$ an initial state, and $\Delta$ a transition relation as before, $\Omega = \{(L_1, U_1), \ldots, (L_n, U_n)\}$ is a collection of pairs $(L_i, U_i)$, $1 \leq i \leq n$, with $L_i, U_i \subseteq Q$.[6] A *run* of $A$ on an infinite tree $t : T \to \Sigma$ is a tree $r : T \to Q$ where $r(\varepsilon) = q_0$, and $(r(x), t(x), r(x1), r(x2)) \in \Delta$ for $x \in T$. It is an *accepting run* if for all paths $\sigma$ of $r$ there exists some *accepting pair* $(L_i, U_i)$, $1 \leq i \leq n$ of $\Omega$ with $\mathrm{In}(\sigma) \cap L_i = \emptyset$ and $\mathrm{In}(\sigma) \cap U_i \neq \emptyset$; thus, $L_i$ contains the states that cannot occur infinitely often on $\sigma$, while there must be some infinitely appearing state $q$ from $\sigma$ that is in $U_i$. A RTA accepts a labeled infinite tree if and only if there exists a run that is accepting. The set of trees that are accepted by a RTA $A$ is the language of $A$ , denoted $L(A)$ as usual.

*Example 2.12 ([Tho90]).* Consider the language $L$ that consists of infinite trees $t : T \to \{a, b\}$ such that every path in $t$ carries only finitely many labels $a$. A RTA that accepts this language has some state $q_a$ that is computed iff the label $a$ is encountered. The acceptance condition is such that for every path $\sigma$ in an accepting run $q_a$ does not appear infinitely often on $\sigma$, or $\mathrm{In}(\sigma) \cap \{q_a\} = \emptyset$, while other states may appear infinitely often, and thus $\mathrm{In}(\sigma) \cap Q \neq \emptyset$: $\Omega = \{(\{q_a\}, Q)\}$ for the state set $Q$.

**Theorem 2.13 ([Tho90]).** *The non-emptiness problem for RTAs is decidable.*

*Proof Sketch.* We first reduce the non-emptiness of RTAs to the non-emptiness of *input-free RTAs*. An input-free RTA is a tuple $(Q, q_0, \Delta, \Omega)$ with $Q$ a set of states, $q_0$ an initial state, and $\Delta \subseteq Q \times Q \times Q$ a transition relation, $\Omega$ is an acceptance condition as before. A *run* of $A$ is a tree $r : T \to Q$, where $T$ is the complete infinite binary tree, with $r(\varepsilon) = q_0$, and $(r(x), r(x1), r(x2)) \in \Delta$ for $x \in T$. Acceptance of runs is defined as for general RTAs.

We transform a RTA $A = (\Sigma, Q, q_0, \Delta, \Omega)$ into $A' \equiv (Q \times \Sigma, Q_0, \Delta', \Omega')$ with $\Delta' \subseteq (Q \times \Sigma) \times (Q \times \Sigma) \times (Q \times \Sigma)$ such that $((q, a), (q', a'), (q'', a'')) \in \Delta'$ iff $(q, a, q', q'') \in \Delta$. $Q_0$ contains all $(q_0, a)$ and $\Omega'$ is such that for a pair $(L, U) \in \Omega'$ with $L, U \subseteq Q \times \Sigma$, the projection onto $Q$ is an accepting pair in $\Omega$. It is easy to check that the successful runs of $A'$ are the $(r, t)$ with $r$ a successful run of $A$ on the tree $t$ (with $(r, t)(x) = (r(x), t(x))$). Furthermore, an input-free automaton $(Q, Q_0, \Delta, \Omega)$ can be reduced to an input-free automaton $(Q, q_0, \Delta, \Omega)$ with a single initial state.

[6] Note that we assume that the set of initial states in the tree automaton is a singleton $q_0$. This does not affect the expressiveness of the automaton: a set of initial states $Q_0 = \{q^1, q^2, \ldots, q^n\}$ can be replaced by a $q_0$ such that $(q_0, a, q', q'') \in \Delta$, for every $(q^i, a, q', q'') \in \Delta$.

Assume $A = (Q, q_0, \Delta, \Omega)$ is an input-free RTA. We call a state $q \in Q$ *live* if $q \neq q_0$ and there are other transitions possible in $\Delta$ than $(q, q, q)$. In the following, we subsequently reduce the number of live states while retaining non-emptiness. For an automaton with 0 live states the non-emptiness problem can be trivially decided: a run is of a form as in Figure 2.11. For

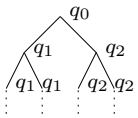


**Fig. 2.11.** No Live States

every path $\pi$ in such a run either $\mathrm{In}(\pi) = \{q_1\}$ or $\mathrm{In}(\pi) = \{q_2\}$, and checking whether it is an accepting run is trivial.

We introduce 4 types of modifications to the original automaton $A$, each of them containing less live states.

1. For a live state $q \in A$, remove $q$ from $A$. The resulting automaton is called $A^1_q$.
2. For a live state $q \in A$, remove all transitions for $q$ and add $(q, q, q)$ to $\Delta$. Transform every accepting pair $(L_i, U_i)$ to $(L'_i, U'_i)$ with $L'_i = L_i \setminus \{q\}$ and $U'_i = U_i \cup \{q\}$. The resulting automaton is called $A^2_q$.
3. For two live states $q$ and $q'$, take $q$ initial and delete $q'$ in the modified automaton. The resulting automaton is called $A^3_{q,q'}$.
4. For a live state $q$, make two copies of $q$, make one copy an initial state and remove all transitions for the other copy while adding $(q, q, q)$. Replace $(L_i, U_i)$ by $(L'_i, U_i)$ such that

$$L'_i = \begin{cases} L_i \cup \{q\} & \text{if there exists a live state in } L_i \\ L_i & \text{else} \end{cases}$$

   The resulting automaton is called $A^4_q$.

*Claim.* $A$ has an accepting run iff

- there exists an $A^1_q$ with an accepting run, or
- there exist $A^2_q$, $A^3_{q,q'}$ with each of them an accepting run, or
- there exist $A^2_q$, $A^4_q$ with each of them an accepting run.

Thus, deciding non-emptiness can be done by writing down all the possible modifications for an automaton (which is finite, since the number of live states is finite), and checking the finite number of the above combinations for accepting runs, which can be done by induction as they contain fewer live states.

We prove the claim. For the "only if" direction, assume $A$ has an accepting run $r$.

- Assume the live state $q$ is missing in $r$. Then, one can show that $r$ is an accepting run of the modified automaton $A^1_q$.
- Assume $r$ contains a node $u$ and $r(u) = q$ with $q$ a live state and $q'$ does not appear as a label of nodes beyond $u$, as in Figure 2.12. Then, $A^2_q$ and

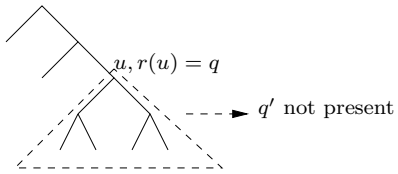


**Fig. 2.12.** Subtree without $q'$

$A^3_{q,q'}$ have accepting runs $r_1$ and $r_2$ respectively, where $r_1$ is the tree in Figure 2.13, i.e. each appearance of $q$ is infinitely followed by $q$'s, and the rest of the tree is like $r$, and $r_2$ is the tree in Figure 2.14, i.e., the subtree $r[u]$.

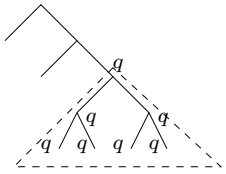


**Fig. 2.13.** Run Accepted by $A^2_q$

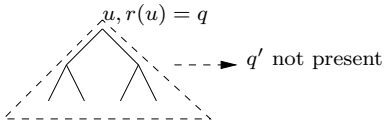


**Fig. 2.14.** Run Accepted by $A^3_{q,q'}$

- Assume all live states appear infinitely beyond every node in $r$. We can then choose a path $\pi_0$ where all live states appear infinitely often. Since $r$ is accepting we have that there exists a $(L_i, U_i)$ such that $\mathrm{In}(\pi_0) \cap L_i = \emptyset$ and $\mathrm{In}(\pi_0) \cap U_i \neq \emptyset$, and thus $L_i$ does not contain any live states (because $\mathrm{In}(\pi_0)$ contains them all). Take $q \in \mathrm{In}(\pi_0) \cap U_i$. Then, $A^2_q$ and $A^4_q$ have accepting runs $r_1$ and $r_2$ respectively, where $r_1$ is the same tree as in Figure

2.13 and $r_2$ is the tree in Figure 2.15, i.e., corresponding to a subtree of $r$ with root $q$, and subsequently considered the second encounter of $q$ to be non-live.

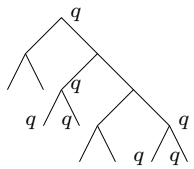


**Fig. 2.15.** Run Accepted by $A^4_q$

For the "if" direction, we distinguish between three cases.

1. Assume there exists an $A^1_q$ with an accepting run $r$. One can show that $r$ is an accepting run of $A$.
2. Assume there exist $A^2_q$, $A^3_{q,q'}$ with runs respectively $r_1$ and $r_2$. One can show that $r_1 \cdot_q r_2$ is an accepting run of $A$.
3. Assume there exist $A^2_q$, $A^4_q$ with respective accepting runs $r_1$ and $r_2$. One can show that $r_1 \cdot_q r_2^\omega$ is an accepting run of $A$.

□

### Two-way Alternating Tree Automata

A transition $(q, a, q'_i, q''_i)$, $1 \leq i \leq n$, in the transition relation $\Delta$ of a RTA $A = (\Sigma, Q, q_0, \Delta, \Omega)$ expresses that, when the automaton reads a node $x$ with label $a$ in state $q$, it goes to node $x1$ in state $q'_i$ and node $x2$ in state $q''_i$ for some $1 \leq i \leq n$. One can alternatively represent the transition relation $\Delta$ by a function $\delta$, such that $\delta(q, a) = \bigvee_{1 \leq i \leq n}((1, q'_i) \wedge (2, q''_i))$, i.e., the disjunction indicates a choice for the different $i$ and the conjunction indicates that the automaton should follow the 1-direction (to $x1$ when in $x$) and enter state $q'_i$ and the 2-direction (to $x2$ when in $x$) and enter state $q''_i$.

In *alternating automata* [MS87] the conjunction and disjunction in the definition of a $\delta$ do not have to adhere to this strict form. Instead, arbitrary positive boolean formulas are allowed, i.e., formulas using $\wedge$ and $\vee$ at liberty. E.g., a definition

$$\delta(q, a) = (1, q_1) \wedge ((2, q_2) \vee (2, q_3))$$

indicates that the automaton, when in some node $x$, proceeds to $x1$ and enters state $q_1$, and subsequently goes to $x2$ and enters either $q_2$ or $q_3$.

The *two-way* aspect is achieved by permitting, besides 1 and 2, also the directions $-1$ and 0, where $-1$ stands for *go one node up in the tree (to* $x \cdot -1$ *when in* $x$*)* and 0 stands for *stay at the current node (to* $x0$ *when in* $x$*).*

Formally, let $\mathcal{B}^+(I)$ be the set of positive boolean formulas over a set $I$. A set $J \subseteq I$ *satisfies* a positive boolean formula $\phi$, if assigning **true** to the elements in $J$ and **false** to the elements in $I \setminus J$ makes $\phi$ true according to the standard inductive semantics for boolean formulas. A *two-way alternating tree automaton (2ATA)* [Var98] over $k$-ary[7] infinite trees is a tuple $(\Sigma, Q, q_0, \delta, \Omega)$ where $\Sigma$ is the input alphabet, $Q$ is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+([k] \times Q)$, with $[k] = \{-1, 0, \ldots, k\}$, $q_0 \in Q$ is the initial state and $\Omega$ is the acceptance condition.

A *run* over a tree $t : T \to \Sigma$ is a tree[8] $r : R \to T \times Q$ such that:

1. $r(\varepsilon) = (\varepsilon, q_0)$,
2. if $y \in R$, $r(y) = (x, q)$, and $\delta(q, t(x)) = \phi$, then there exists a (possibly empty) set $S = \{(c_1, q_1), \ldots, (c_n, q_n)\} \subseteq [k] \times Q$ such that
   a) $S$ satisfies $\phi$, and
   b) $yi \in R$, for all $0 < i \leq n$, $xc_i$ is defined and $r(yi) = (xc_i, q_i)$.

Thus, the label $(x, q)$ of a node in a run indicates the node $x$ that the automaton is scanning as well as the state $q$ it is in. A run $r$ is *accepting* if all its infinite paths satisfy the acceptance condition $\Omega$. We consider *parity acceptance conditions*, i.e. $\Omega = (G_1, \ldots, G_m)$ such that $G_1 \subseteq G_2 \subseteq \ldots \subseteq G_m = Q$, and a run $r$ satisfies $\Omega$ if for every infinite path $\pi$ in $r$, there exists an even $i$ such that $\mathrm{In}(\pi) \cap G_i \neq \emptyset$ and $\mathrm{In}(\pi) \cap G_{i-1} = \emptyset$.[9] Further note that, in contrast with RTAs, the run of a 2ATA on a tree $t$ might have a different structure than $t$.

Decidability of the non-emptiness problem for 2ATAs is shown by a reduction to the non-emptiness problem for RTAs, according to the following steps [Var98]:

- define a notion of run (a *strategy tree*) that has the same tree structure as the input tree,
- since paths in such strategy trees can still go up in the input tree (with the $-1$ direction), as opposed to the one-way runs of RTAs, define an *annotation* of the strategy tree such that paths only go down the tree.

The acceptance of a run of a 2ATA is then reduced to the acceptance of a strategy tree with accepting annotation, where the latter can be performed by a RTA.

First, we introduce the notion of *strategy tree*, a tree with the same structure as the input tree. A *strategy tree* for a 2ATA $A$ on infinite $k$-ary trees is a tree $\tau : \{1, \ldots, k\}^* \to 2^{Q \times [k] \times Q}$. The set of sources in a label $\chi \in 2^{Q \times [k] \times Q}$

[7] Note that we define 2ATAs over $k$-ary trees whereas RTAs were defined over binary trees. However, definitions and results for the latter can be easily extended to the $k$-ary case.

[8] Note that the alphabet of $r$ is infinite.

[9] A parity acceptance condition $(G_1, \ldots, G_m)$ corresponds to an acceptance condition $((G_1, G_2), \ldots, (G_{m-1}, G_m))$ or $((G_1, G_2), \ldots, (G_{m-1}, G_m), (G_m, \emptyset))$ for even or odd $m$ respectively.

is $state(\chi) \equiv \{q \mid (q, c, q') \in \chi\}$. A strategy tree $\tau$ is *over* a $k$-ary input tree $t$ if $q_0 \in state(\tau(\varepsilon))$, and, for each node $x \in \{1, \ldots, k\}^*$ and each state $q \in state(\tau(x))$, the set $\{(c, q') \mid (q, c, q') \in \tau(x)\}$ satisfies $\delta(q, t(x))$. Thus, when confronted with $\delta(q, t(x))$, the satisfying set (and thus the directions and states to go to next) can be simply obtained from $\tau(x)$ such that $\tau$ is rightly called a strategy for *running* the automaton $A$ over $t$.

A *path* $\pi$ in a strategy tree $\tau$ is a sequence

$$(x_1, q_1), (x_2, q_2), \ldots$$

with $(x_i, q_i) \in \{1, \ldots, k\}^* \times Q$ such that there exists a $(q_i, c, q_{i+1}) \in \tau(x_i)$ and $x_{i+1} = x_i c$; $\mathrm{In}(\pi)$ is the set of states that appear infinitely often in $\pi$. A strategy tree $\tau$ over $t$ is *accepting* if for every path $\pi$ in $\tau$ there is an even $i$ such that $\mathrm{In}(\pi) \cap G_i \neq \emptyset$ and $\mathrm{In}(\pi) \cap G_{i-1} = \emptyset$, where $(G_1, \ldots, G_m)$ is the acceptance condition of $A$.

**Theorem 2.14 ([Var98, Cac02]).** *A 2ATA accepts an input tree iff it has an accepting strategy tree over the input tree.*

A path in a strategy tree can, however, still go up in the tree: it may be of the form $(x_1, q_1), (x_2, q_2), (x_1, q_3), \ldots$, i.e., $x_1$ is visited twice. The second step involves removing those "detours".

An *annotation* is a tree $\eta : \{1, \ldots, k\}^* \to 2^{Q \times 2^{\{1,\ldots,m\}} \times Q}$ with $m$ as in the acceptance condition $(G_1, \ldots, G_m)$ of the automaton. Define $index(q)$ as the minimal $i$ such that $q \in G_i$. The intuition for $(q, H, q') \in \eta(x)$ is "for a node $x$ and in state $q$, the automaton moves through states $q_i$ with $index(q_i) \in H$ again to $x$ and state $q'$". We forget about the particular states that are visited by the automaton during its detour and only record the $G_i$ that contained the visited states – for acceptance we are only interested in checking infinite visits to states in such $G_i$'s. The annotation $\eta$ is an *annotation of a strategy tree* $\tau$ if the following closure conditions hold for $x \in \{1, \ldots, k\}^*$:

1. if $(q, H_1, q') \in \eta(x)$ and $(q', H_2, q'') \in \eta(x)$ then $(q, H_1 \cup H_2, q'') \in \eta(x)$. Thus, if, for a node $x$, the automaton moves from state $q$ through states $q_i$ with $index(q_i) \in H_1$ again to $x$ and state $q'$, and it moves from state $q'$ through states $q_i$ with $index(q_i) \in H_2$ again to $x$ and state $q''$, then the automaton moves from state $q$ through states $q_i$ with $index(q_i) \in H_1 \cup H_2$ again to $x$ and state $q''$.
2. if $(q, 0, q') \in \tau(x)$ then $(q, \{index(q')\}, q') \in \eta(x)$. Thus, if the strategy tree says that when reading $x$ and in state $q$ you have to stay in $x$ (the "0") and go to state $q'$, then the $\eta(x)$ remembers the $G_i$ that $q'$ belongs to, i.e. $index(q')$.
3. if $(q, -1, q') \in \tau(xi), (q', H, q'') \in \eta(x)$, and $(q'', i, q''') \in \tau(x)$, then $(q, H \cup \{index(q'), index(q''')\}, q''') \in \eta(xi)$.
4. if $(q, i, q') \in \tau(x), (q', H, q'') \in \eta(xi)$, and $(q'', -1, q''') \in \tau(xi)$, then $(q, H \cup \{index(q'), index(q''')\}, q''') \in \eta(x)$.

The annotation $\eta$ remembers where the automaton has been on its detour. Like for strategy trees, we define a notion of paths: *downward paths*. A downward path in $\eta$ is a sequence

$$(u_1, q_1, t_1), (u_2, q_2, t_2), \ldots$$

with $u_i \in \{1, \ldots, k\}^*$, $q_i \in Q$, and $t_i$ in $\tau(u_i)$ or $\eta(u_i)$, such that

- $t_i$ is $(q_i, c, q_{i+1})$, $c \in \{1, \ldots, k\}$, $u_{i+1} = u_i c$. We define $\mathit{index}(t_i) \equiv \mathit{index}(q_{i+1})$. Note that $c$ is a direction going strictly down the tree, and that the index of $t_i$ records the $G_j$ through which the automaton would pass (by state $q_{i+1}$).
- $t_i$ is $(q_i, H, q_{i+1})$, $H \subseteq \{1, \ldots, m\}$, $u_{i+1} = u_i$. We define $\mathit{index}(t_i) = \mathit{min}(H)$. We just record the minimal $i$ such that a $q \in G_i$, with $q$ on the way from $q_i$ back to $q_{i+1}$ (the minimum is sufficient since $G_i \subseteq G_{i+1} \subseteq \ldots \subseteq G_m$).

We distinguish between finite and infinite downward paths.

1. infinite downward paths $\pi : (u_1, q_1, t_1), (u_2, q_2, t_2), \ldots$ with $\mathit{index}(\pi)$ defined as the minimal $j$ such that $\mathit{index}(t_i) = j$ for infinitely many $t_i$.
2. finite downward paths $\pi : (u_1, q_1, t_1), \ldots, (u_s, q_s, t_s)$ with $t_s = (q_s, H_s, q_s)$ and $\mathit{index}(\pi) \equiv \mathit{index}(t_s)$.

A downward path *violates* the acceptance condition $\Omega$ if $\mathit{index}(\pi)$ is odd. An annotation $\eta$ for a strategy tree $\tau$ is *accepting* if no downward path in $\eta$ violates $\Omega$.

**Theorem 2.15 ([Var98]).** *A 2ATA accepts an input tree iff it has a strategy tree over the input tree and an accepting annotation of the strategy tree.*

*Proof Sketch.* Let $A = (\Sigma, Q, \delta, q_0, \Omega)$ be a 2ATA and $t : \{1, \ldots, k\}^* \to \Sigma$ the input tree.

For the "only if" direction, assume $A$ accepts the input tree. By Theorem 2.14, there is an accepting strategy tree $\tau$ over $t$. Given two annotations $\eta_1$ and $\eta_2$ one can check that $\eta_1 \cap \eta_2$ is also an annotation over $\tau$, defined as $\eta_1 \cap \eta_2(x) = \eta_1(x) \cap \eta_2(x)$. We take $\eta$ the minimal annotation, where $\eta$ is minimal if for every $\eta'$ we have that $\eta \subseteq \eta'$ ($\eta(x) \subseteq \eta'(x)$ for each node $x$). We prove that there is no downward path in $\eta$ that violates $\Omega$.

By contradiction, assume there is a downward path $\kappa$ that violates $\Omega$. We distinguish between two cases:

1. $\kappa$ finite, then $\kappa : (u_1, q_1, t_1), \ldots, (u_s, q_s, t_s)$ with $t_s = (q_s, H_s, q_s)$ and $\mathit{index}(\kappa) = \mathit{index}(t_s) = \mathit{min}(H_s)$ is odd (by definition of violation). We can write down a path in the strategy tree by an expansion of $\kappa$. For every $(u_i, q_i, t_i)$,$(u_{i+1}, q_{i+1}, t_{i+1})$ with $t_i = (q_i, c, q_{i+1})$, we retain $(u_i, q_i)$,$(u_{i+1}, q_{i+1})$. If $t_i = (q_i, H_i, q_{i+1})$ then there are $q_i^1, \ldots, q_i^l$ and $c_i^1, \ldots, c_i^l$ such that $\mathit{index}(q_i^j) \in H_i$ and we retain

$$(u_i, q_i), (u_i c_i^1, q_i^1), \ldots, (u_i c_i^1 \ldots c_i^l, q_i^l), (u_{i+1}, q_{i+1}) \ .$$

The last $(q_s, H_s, q_s)$ is expanded likewise (and repeated infinitely). The result is a path $\kappa'$ in the strategy tree $\tau$, where the states $q_s^j$ appear infinitely often.
We prove that for all even $i$: $\mathrm{In}(\kappa') \cap G_i = \emptyset$ or $\mathrm{In}(\kappa') \cap G_{i-1} \neq \emptyset$. Take an even $i$ and $\mathrm{In}(\kappa') \cap G_i \neq \emptyset$, then there is a $q_s^j \in \mathrm{In}(\kappa')$ and $q_s^j \in G_i$. $index(\kappa)$ is odd such that the smallest element $n$ of $H_s$ is odd. Since $n \in H_s$, there is a $q_s^k$ such that $q_s^k \in G_{n=index(q_s^k)}$. We have that $index(q_s^j) \leq i$, $q_s^j \in G_{index(q_s^j)}$, and $index(q_s^j) \in H_s$ (by definition of $q_s^j$'s). Thus, $index(q_s^k) \leq index(q_s^j) \leq i$; since $index(q_s^k)$ is odd and $i$ is even this cannot be an equality such that $index(q_s^k) \leq i - 1$. By $G_1 \subseteq G_2 \subseteq \ldots \subseteq G_m$, we have that $q_s^k \in G_{n=index(q_s^k)} \subseteq G_{i-1}$. Since $q_s^k \in \mathrm{In}(\kappa')$, $\mathrm{In}(\kappa') \cap G_{i-1} \neq \emptyset$. And thus, by definition of acceptance of paths in strategy trees, $\kappa'$ is not accepted, contradicting that we have an accepting strategy tree and that all paths should be accepting.
2. The proof for an infinite $\kappa$ is similar.

For the "if" direction, assume $\tau$ is a strategy tree over the input tree $t$ and $\eta$ an accepting annotation. Then no downward path in $\eta$ violates $\Omega$. Assume $\eta'$ is the minimal annotation of $\tau$ then $\eta' \subseteq \eta$ and no downward path in $\eta'$ violates $\Omega$. One can then prove that all paths in $\tau$ are accepting (by rewriting them as downward paths and using that no downward path violates $\Omega$), and, by Theorem 2.14, there is an accepting input tree. □

For a 2ATA $A$ and an input tree, there exists a RTA $A^n$ that accepts an accepting notation of a strategy tree over the input tree. Since the non-emptiness problem for RTAs is decidable, it is, with Theorem 2.15, for 2ATAs as well.

**Theorem 2.16.** *The non-emptiness problem for 2ATAs is decidable.*

*Proof Sketch.* Let $A$ be 2ATA. We construct the RTA $A^n$ as the intersection of two automata.

1. The RTA $A_1$ checks, given a tuple $(t, \tau, \eta)$, that $\tau$ is a strategy tree over $t$, and that $\eta$ is an annotation of $\tau$.
2. For a downward path $\kappa : (u_1, q_1, t_1), (u_2, q_2, t_2), \ldots$, a projection of $\kappa$ is $proj(\kappa) \equiv (q_1, t_1), (q_2, t_2), \ldots$ $A_2$ is then constructed in different phases.
   - $B$ is a (word[10]) automaton that accepts projections of downward paths that violate the acceptance condition $\Omega$.
   - $B'$ is constructed from $B$; it reads sequences of labels from $\tau$ or $\eta$ and checks whether they contain a downward path that violates $\Omega$.
   - $B''$ is the complemented determinized version of $B'$ such that $B''$ rejects violated downward paths.
   - The RTA $A_2$ runs $B''$ in parallel over the branches of $(t, \tau, \eta)$.

□

[10] A *word automaton* is a RTA on 1-ary trees, i.e., on strings or words.

**Theorem 2.17.** *The non-emptiness problem for 2ATAs is in* EXPTIME.

*Proof Sketch.* By Theorem 7 in [Var98], the number of states in the RTA $A^n$ is exponential in the number of states of the 2ATA $A$ and the size of the acceptance condition of $A^n$ is linear in the size of the acceptance condition of $A$. Since there are algorithms that solve the non-emptiness problem for RTAs in time polynomial in the number of states but exponential in the size of the acceptance condition [Var98, EJ00], we have, with Theorem 2.16, algorithms that solve the non-emptiness problem for 2ATAs in time exponential in the number of states and the size of the acceptance condition. □

## 2.3 Knowledge Representation Formalisms

In this section, we introduce four knowledge representation formalisms that will appear throughout this dissertation.

### 2.3.1 Answer Set Programming

*Answer set programming (ASP)* is a logic programming paradigm, based on the stable model semantics for negation as failure [GL88]. In ASP, one uses a *logic program*, a set of rules, to declaratively describe a domain, or, more specifically a particular problem. The answers of the program, given by a formally defined *answer set semantics*, correspond to an explicitization of the knowledge in the described domain, or to the solutions of the problem. One does not specify how to derive knowledge from a domain, or how to solve the problem, one merely tries to state *what* the domain is, or, in which terms a solution to a problem can be characterized: ASP is a *declarative* approach to knowledge representation, reasoning, and problem solving.

The answer set methodology has been successfully applied in problem areas such as planning [Lif02, EFL$^+$00, EFL$^+$02], configuration and verification [SN99, SNTS01], diagnosis [EFLP99, VNV03], game theory [DVV99], updates [EFST00], and database repairs [ABC00, VNV02]. Moreover, several *answer set solvers*, i.e., systems that return the answer sets of the program, have reached a mature stage of development. E.g., SMODELS [Sim, NS96, NS97] and DLV [LPF, LRS97, EFLP00]. For a thorough treatment of ASP, we refer to [Bar03].

*Example 2.18 (3-colorability).* Consider the 3-colorability problem, where, given a graph and three colors, one wants to find a coloring of the graph such that no adjacent nodes have the same color. One can encode this problem as a logic program [Col]. The first 2 rules in the program

$$\begin{aligned} node(X) &\leftarrow edge(X, Y) \\ node(Y) &\leftarrow edge(X, Y) \\ colored(X, r) \vee colored(X, g) \vee colored(X, b) &\leftarrow node(X) \end{aligned}$$

say that, if there is an edge from $X$ to $Y$ then $X$ and $Y$ are nodes; the third rule says that if $X$ is a node then $X$ has got to be colored with one of the three colors, $r$, $g$, or $b$. We have a rule enforcing the colorability condition:

$$\leftarrow \mathit{edge}(X, Y), \mathit{colored}(X, C), \mathit{colored}(Y, C)$$

expressing that if there is an edge from $X$ to $Y$ and $X$ and $Y$ are colored with the same color $C$, then we have a contradiction (the empty left hand side of $\leftarrow$). Finally, the graph that is to be colored is represented as the set of edges

$$\begin{array}{ll} \mathit{edge}(2,4) \leftarrow & \mathit{edge}(2,3) \leftarrow \\ \mathit{edge}(3,5) \leftarrow & \mathit{edge}(4,6) \leftarrow \\ \mathit{edge}(4,5) \leftarrow & \mathit{edge}(5,7) \leftarrow \\ \mathit{edge}(6,7) \leftarrow & \mathit{edge}(3,4) \leftarrow \\ \mathit{edge}(5,6) \leftarrow & \end{array}$$

Applying the answer set solver DLV to this program yields, among others, an answer set that contains

$$\{\mathit{colored}(2,b), \mathit{colored}(3,r), \mathit{colored}(4,g), \mathit{colored}(5,b), \\ \mathit{colored}(6,r), \mathit{colored}(7,g)\} ,$$

corresponding to a valid 3-coloring of the graph.[11]

We define the language of ASP. A *term* is a constant or a variable, where the former will be written lower-case and the latter upper-case. An *atom* is of the form $p(t_1, \ldots, t_n)$, $0 \leq n < \infty$[12], where $p$ is an $n$-ary predicate name and $t_i$, $1 \leq i \leq n$, are terms. A *literal* is an atom $a$ or a classically negated atom $\neg a$; an *extended literal* is a literal $l$ or a *naf-literal* not l, i.e., a literal preceded with the *negation as failure* symbol *not*.

A *(logic) program (LP)* is a countable set of *rules*

$$\alpha \leftarrow \beta$$

where $\alpha$ and $\beta$ are finite sets of extended literals, respectively called the *head* and *body* of the rule. For a rule $r$, we denote the head as head(r) and the body as body(r). The body of a rule is considered to be a conjunction of extended literals (denoted as a comma-separated list) and the head as a disjunction of extended literals (denoted as a $\vee$-separated list). If $\alpha = \emptyset$, we denote the rule as $\leftarrow \beta$ and call it a *constraint*. The *positive part* of a set of extended literals $\gamma$ is

$$\gamma^+ \equiv \{l \mid l \in \gamma, l \text{ literal}\} ,$$

[11] Note that 3-colorability is an NP-complete problem such that one does not need the full power of DLV (which is $\Sigma_2^{\mathrm{P}}$-complete due to presence of disjunction in the heads of rules and the demand for minimality in the disjunction).

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

and the *negative part* is

$$\gamma^- \equiv \{l \mid not\ l \in \gamma\}\ .$$

E.g., for $\gamma = \{a, not\ \neg b, not\ c\}$, we have that $\gamma^+ = \{a\}$ and $\gamma^- = \{\neg b, c\}$.

A *ground* atom, (extended) literal, rule, or program does not contain variables. Substituting every variable in a program $P$ with every possible constant in $P$ yields the ground program $gr(P)$.

*Example 2.19.* Grounding a program $P$

$$p(X) \leftarrow not\ q(X, b)$$
$$q(X) \leftarrow not\ p(X, a)$$

yields the program

$$p(a) \leftarrow not\ q(a, b)$$
$$p(b) \leftarrow not\ q(b, b)$$
$$q(a) \leftarrow not\ p(a, a)$$
$$q(b) \leftarrow not\ p(b, a)$$

Note that a variable $X$ in a rule should be grounded with the same constant in that rule (either with $a$ or with $b$), while it may be grounded with other constants in other rules, i.e., the variables in a rule are considered local to the rule. One can, e.g., replace the above program by the equivalent

$$p(X) \leftarrow not\ q(X, b)$$
$$q(Y) \leftarrow not\ p(Y, a)$$

All following definitions in this section assume ground programs and ground (extended) literals; to obtain the definitions for unground programs, replace every occurrence of a program $P$ by $gr(P)$, e.g., an answer set of an unground $P$ is an answer set of $gr(P)$.

The *Herbrand Base* $\mathcal{B}_P$ of a program $P$ is the set of all ground atoms that can be formed using the language of $P$. For a set $X$ of literals, we take $\neg X \equiv \{\neg l \mid l \in X\}$ where $\neg\neg a \equiv a$; $X$ is *consistent* if $X \cap \neg X = \emptyset$, i.e., $X$ does not contain contradictory literals $a$ and $\neg a$. Let $\mathcal{L}_P$ be the set of literals that can be formed with $P$, i.e., $\mathcal{L}_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$.

An *interpretation* $I$ of $P$ is any consistent subset of $\mathcal{L}_P$. For a literal $l$, we write $I \models l$, if $l \in I$, which extends for extended literals $not\ l$ to $I \models not\ l$ if $I \not\models l$. In general, for a set of extended literals $X$, $I \models X$ if $I \models x$ for every extended literal $x \in X$. A rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. $I$, denoted $I \models r$, if $\exists l \in \alpha \cdot I \models l$, for some extended literal $l$, whenever $I \models \beta$, i.e., $r$ is *applied* ($\exists l \in \alpha \cdot I \models l$ and $I \models \beta$) whenever it is *applicable* ($I \models \beta$). Since a constraint has an empty head, the previous yields that constraints cannot be applicable if they are to be satisfied. The set of satisfied rules in $P$ w.r.t. $I$ is the *reduct* $P_I$.

For a *simple* program $P$ (i.e., a program without *not*), an interpretation $I$ is a *model* of $P$ if $I$ satisfies every rule in $P$, i.e., $P_I = P$; it is an *answer*

*set* of $P$ if it is a minimal model of $P$, i.e., there is no model $J$ of $P$ such that $J \subset I$.

*Example 2.20 ([Lif02]).* Take the program

$$p \vee q \leftarrow$$
$$\neg r \leftarrow p$$

Then, we have 4 models[13]: $\{p, \neg r\}$, $\{q, \neg r\}$, $\{q\}$, and $\{p, q, \neg r\}$. Since $\{q\}$ is a strict subset of $\{q, \neg r\}$ and $\{p, q, \neg r\}$, we have that $\{p, \neg r\}$ and $\{q\}$ are the minimal models (or answer sets) of the program.

Adding a constraint $\leftarrow q$ yields $\{p, \neg r\}$ as the unique answer set. Subsequently adding $\leftarrow p$ results in an *inconsistent* program, i.e., a program without answer sets.

As in [Lif02], we define answer sets for programs with *not* in terms of a reduction to simple programs. The *GL-reduct*[14] w.r.t. an interpretation $I$ is the simple $P^I$, where $P^I$ contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in $P$, $I \models \alpha^-$, and $I \models not\ \beta^-$. Thus, given an interpretation $I$ of literals – the items that one supposes true – the GL-reduct contains those rules for which the negative part is consistent with the beliefs in $I$. If there is a naf-literal in the body that is not true in $I$, then the rule is not in the GL-reduct since its whole body is then false and cannot be used to deduce literals. If all naf-literals in the body are true, the rule stays in the GL-reduct (depending on the naf-literals in the head), but with the naf-literals removed (they are known to be true). A similar reasoning holds for the head of a rule: if there is a naf-literal in the head that is true w.r.t. $I$, we have that the rule is automatically true and can be removed; if all naf-literals in the head are false, then we remove them and leave the rule in the GL-reduct.

$I$ is an *answer set* of $P$ if $I$ is an answer set of $P^I$. Thus, given an interpretation $I$, one calculates the GL-reduct, and checks that the minimal model of the GL-reduct is $I$; an answer set is thus *self-motivating* or *stable*.

*Example 2.21.* Take the program $P$ that consists of the rule $p \leftarrow not\ p$. Then $\{p\}$ is a model of this rule. However, it is not stable in the above sense: the GL-reduct $P^{\{p\}}$ is the empty set. Indeed, the naf-literal in the body is false w.r.t. $\{p\}$, thus the rule cannot be used to deduce $p$. Since the minimal model of $\emptyset$ is $\emptyset \neq \{p\}$, $\{p\}$ is not an answer set. Another guess might be the empty set: the GL-reduct w.r.t. $\emptyset$ is the rule $p \leftarrow$ which has the minimal model $\{p\}$, again not confirming the initial guess. In fact, $P$ has no answer sets.

*Example 2.22.* Take now a program $P$

$$a \leftarrow not\ b$$
$$b \leftarrow not\ a$$

[13] Without loss of generality, we ignore models that contain literals that do not appear in the program.

[14] Named after its inventors M. Gelfond and V. Lifschitz.

Consider the following four interpretations: $\emptyset$, $\{a\}$, $\{b\}$, and $\{a, b\}$. The GL-reduct of $P$ w.r.t. $\emptyset$ is $\{a \leftarrow; b \leftarrow\}$ which has $\{a, b\}$ as its minimal model, and thus $\emptyset$ is not an answer set. The GL-reduct of $P$ w.r.t. $\{a, b\}$ is $\emptyset$ which has $\emptyset$ as its minimal model, and thus $\{a, b\}$ is not an answer set. The GL-reduct of $P$ w.r.t. $\{a\}$ is $\{a \leftarrow\}$ which has $\{a\}$ as its minimal model, making $\{a\}$ an answer set. Similarly, one can deduce that $\{b\}$ is an answer set.

*Example 2.23 ([FL05]).* Take a program consisting of the rule $q \leftarrow not\ p$, then this program has one answer set: $\{q\}$. However, its contrapositive $p \leftarrow not\ q$ has the different answer set $\{p\}$. Thus, although those two rules are equivalent as propositional formulas, i.e., $q \leftarrow not\ p$ corresponds to the formula $\neg p \Rightarrow q$ and $p \leftarrow not\ q$ to the formula $\neg q \Rightarrow p$, they are not under the answer set semantics.

We are mainly interested in the following decision problem:

> Given a logic program $P$ and a ground literal $l$, is there an answer set of $P$ that contains $l$?

We summarize some complexity results for this decision problem in Table 2.2. According to [IS98], negation as failure in the head does not add any computational power, such that the results are valid for both programs with and without negation as failure in the head. In the non-disjunctive case [MT91],

**Table 2.2.** Complexity Results Answer Set Programming

| $\alpha \leftarrow \beta$ | ground | not ground |
|---|---|---|
| non-disjunctive ($0 \leq \lvert\alpha\rvert \leq 1$) | NP-complete | NEXPTIME-complete |
| disjunctive ($0 \leq \lvert\alpha\rvert$) | $\Sigma_2^{\mathrm{P}}$-complete | $\mathrm{NEXPTIME}^{\mathrm{NP}}$-complete |

i.e. the heads of the rules are (at most) singletons, checking whether there is an answer set of some ground non-disjunctive program is NP-complete. That the problem is in NP can be seen as follows:

- guess an interpretation (hence the nondeterminism),
- compute the GL-reduct; this can be done in polynomial time, and
- check that the minimal model of the GL-reduct is equal to the guess. Since the GL-reduct does not contain negation as failure nor disjunction this can be done in polynomial time (by a fixed point construction).

In the non-ground case, one has to ground the program first, which may, in the worst case, result in a ground program that has a size that is exponential in the size of the non-ground program, hence the NEXPTIME membership. The disjunctive case [EG93] is similar but an extra guess is needed since the GL-reduct now contains disjunction and one can no longer check in polynomial time that an interpretation is a minimal model of a simple program. For more details, we refer to [Bar03, DEGV01].

### 2.3.2 Description Logics

*Description logics (DLs)* are a family of logical formalisms based on frame-based systems [Min85] and useful for knowledge representation; e.g., the representation of taxonomies in certain application domains [RWRR01]. Its basic language features include the notions of *concepts* and *roles* which are used to define the relevant concepts and relations in some (application) domain. Different DLs can then be identified, among others, by the set of constructors that are allowed to form complex concepts or roles.

Description logics originated from *structural inheritance networks* [Bra77] which were defined to solve ambiguities in semantic networks and frames, and were first implemented in the KL-ONE system [BS85] [BCM+03]. Three ideas drove the development of description logics [BCM+03]:

- The basic syntactic building blocks of a description logic are atomic concepts, atomic roles, and individuals, basically corresponding respectively to unary predicates, binary predicates and constants.
- One tries to balance expressivity and decidability/complexity by considering only a basic set of constructors that can be used to construct complex concept expressions.
- Implicit knowledge can be inferred with the help of sound and complete inference procedures that check, e.g., satisfiability of concepts.

Since KL-ONE, reasoners for expressive DLs have emerged, e.g., RACER [HM01] and FACT [Hor98]. The combination of a formal well-understood semantics and the availability of practical reasoners, has led to the adoption of DLs as the formal underpinning of ontology languages on the Semantic Web.

The "Semantic Web" [BLHL01] seeks to improve on the current World Wide Web, making knowledge not only viewable and interpretable by humans, but also by software agents. Ontologies play a crucial role in the realization of this next generation web, by providing a "shared understanding" [UG96] of certain domains. In order to describe ontologies, one can use ontology languages, such as DAML+OIL, OIL [BGH01, FHvH+00, FvHH+01], or, more recently, OWL [BvHH+]. For example, the OIL language is built on three roots [HFB+00]:

- the concrete syntax is based on web languages such as XML and RDF [LS99, DvHB+00],
- a frame-based language that provides the basic modeling primitives: frames (classes) with attributes,
- by mapping the language to a suitable description logic, one obtains a precise semantics and associated inference procedures.

A DL can then be used to express the formal semantics of an ontology written in an ontology language like OIL, but also provide some basic reasoning services such as checking whether an instance is of a certain type, whether classes are subclasses of other classes, ... [BS00, HST99].

The semantics of DLs is given by interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty domain and $\cdot^{\mathcal{I}}$ is an interpretation function. For a set of data types $\mathbf{D}$, we associate with each $d \in \mathbf{D}$ a set $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ where $\Delta_{\mathbf{D}}$ is the domain of all data types (the *concrete domain*, see [FH91]). We give an overview of the most commonly used concept/role constructors, together with their definition in terms of an interpretation $\mathcal{I}$. The basic building blocks are the following:

- *Concept names* $A$ are interpreted as a subset of the domain: $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, i.e., $A$ is intuitively a set of domain elements that are of the same type. E.g., *Workers* is a concept such that $Workers^{\mathcal{I}}$ are those domain elements in $\Delta^{\mathcal{I}}$ that are considered to be among the workers in a company. The set of available concept names is denoted $\mathbf{C}$.
- We distinguish between two types of *role names*:
  - *Abstract role names* $P$ are interpreted as a relation on the domain: $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, i.e., $P$ relates domain elements. E.g., *boss* is a relation indicating which domain elements are considered to be the boss of other domain elements. The set of available abstract role names is denoted $\mathbf{R}_A$.
  - *Concrete role names* $T$ relate domain elements to concrete domain elements: $T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}$. E.g., *shoesize* could relate domain elements representing persons to a particular integer. The set of available concrete role names is denoted $\mathbf{R}_{\mathrm{d}}$.
- *Nominals* (*individuals*) $\{o\}$ represent particular identified entities in the DL. Their interpretation is such that $\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and $|\{o\}^{\mathcal{I}}| = 1$. E.g., $\{john\}$ is a nominal representing John. We will assume the *unique name assumption* – if $\{o_1\} \neq \{o_2\}$ then $\{o_1\}^{\mathcal{I}} \neq \{o_2\}^{\mathcal{I}}$ – which ensures that different individuals are interpreted as different domain elements. Note that OWL does not have the unique name assumption [SWM04], and thus different individuals can point to the same resource. However, the open answer set semantics, see Chapter 3, gives a Herbrand interpretation to constants, i.e. constants are interpreted as themselves, and for consistency we assume that also DL nominals are interpreted this way.
- For an abstract role $P$, we can define its *inverse role* $P^-$, which is interpreted as the inverse of the interpretation of $P$: $P^{-\mathcal{I}} = \{(y,x) | (x,y) \in P^{\mathcal{I}}\}$. We assume the $\cdot^-$ operator is also defined for inverse roles such that for a role name $P$: $(P^-)^- = P$. Unless specified otherwise, we denote with *roles* either inverted role names or just role names.

Based on those building blocks, we define *concept expressions* as follows:

- Every concept name is a concept expression.
- Every nominal is a concept expression.
- A *concept conjunction* $C \sqcap D$[15] is a concept expression that is interpreted as the conjunction of the interpretations of $C$ and $D$: $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$. E.g., *Management* $\sqcap$ *Workers* are the managers that are also workers.

[15] In the following, we assume $C$ and $D$ are concept expressions.

- A *concept disjunction* $C \sqcup D$ is a concept expression that is interpreted as the disjunction of the interpretations of $C$ and $D$: $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$. E.g., $Management \sqcup Workers$ is the set of elements that are either managers or workers.
- A *negation* $\neg C$ is a concept expression that is interpreted as the complement of $C$ w.r.t. $\Delta^{\mathcal{I}}$: $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$. E.g., $\neg Management$ is everything in the domain that is not a manager.
- An *exists restriction* $\exists R.C$ is a concept expression and its interpretation consists of those elements $x$ that relate via $R$ to some element in $C$:$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$. E.g., $\exists boss.Management$ are those elements that are the boss of some manager.
- A *value restriction* $\forall R.C$ is a concept expression and its interpretation consists of those elements $x$ such that, *if* there is a relation via $R$ with an element $y$, then $y$ belongs to $C$: $(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$. E.g., $\forall take\_orders.Management$ are those elements that if they take orders from someone, then they only take orders from a manager.
- A *qualified at least restriction* $\geq nS.C$, where $S$ is a role expression (defined below) and $n$ is a nonnegative integer, is a concept expression that indicates all those elements that have at least $n$ $S$-successors that belong to $C$: $(\geq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$. An *unqualified at least restriction* $\geq nS$ is a concept expression such that $(\geq nS)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}}\} \geq n\}$.
- A *qualified at most restriction* $\leq nS.C$, where $S$ is a role expression (defined below) and $n$ is a nonnegative integer, is a concept expression that indicates all those elements that have at most $n$ $S$-successors that belong to $C$: $(\leq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$. An *unqualified at most restriction* $\leq nS$ is a concept expression such that $(\leq nS)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}}\} \leq n\}$. We refer to *(un)qualified number restrictions* for either the *at least* or *at most* versions.
- A *data type exists restriction* $\exists T.d$, where $T$ is a concrete role and $d$ a datatype, consists of those elements $x$ that relate via $T$ to some concrete domain element in $d$:$(\exists T.d)^{\mathcal{I}} = \{x \mid \exists y : (x, y) \in T^{\mathcal{I}} \text{ and } y \in d^{\mathbf{D}}\}$. E.g., $\exists shoesize.int$ are those elements that have an integer shoe size.
- A *data type value restriction* $\forall T.d$, where $T$ is a concrete role and $d$ a datatype, consists of those elements $x$ such that, *if* there is a relation via $T$ with a concrete domain element $y$, then $y$ belongs to $d$: $(\forall T.d)^{\mathcal{I}} = \{x \mid \forall y : (x, y) \in T^{\mathcal{I}} \Rightarrow y \in d^{\mathbf{D}}\}$. E.g., $\forall shoesize.int$ are those elements that if they have a shoe size then that shoesize is an integer.

*(Abstract) role expressions* are defined as follows:

- Every role or inverted role name is a role expression.
- A *role conjunction* $R \sqcap S$, with $R$ and $S$ role expressions, is a role expression. It is interpreted as the conjunction of the interpretations of $R$ and $S$: $(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$. E.g., $boss \sqcap older$ is the role expression that contains all $(x, y)$ such that $x$ is the boss of $y$ and $x$ is older than $y$.

- A *role disjunction* $R \sqcup S$, with $R$ and $S$ role expressions, is a role expression. It is interpreted as the disjunction of the interpretations of $R$ and $S$: $(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$. E.g., *boss* $\sqcup$ *older* is the role expression that contains all $(x, y)$ such that $x$ is either the boss of $y$ or is older than $y$.

We summarize the constructs with their interpretation in Table 2.3.

**Table 2.3.** Syntax and Semantics of DL Constructs

| construct name | syntax | semantics |
|---|---|---|
| atomic concept **C** | $A$ | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| abstract role | $R$ | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| inverse abst. role | $R^-$ | $(R^-)^{\mathcal{I}} = \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$ |
| concrete role | $T$ | $T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}$ |
| nominals **I** | $\{o\}$ | $\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \lvert\{o\}\rvert^{\mathcal{I}} = 1$ |
| data types **D** | $d$ | $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ |
| role conjunction | $R \sqcap S$ | $(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$ |
| role disjunction | $R \sqcup S$ | $(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$ |
| concept conj. | $C \sqcap D$ | $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| concept disj. | $C \sqcup D$ | $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| negation | $\neg C$ | $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| exists restriction | $\exists R.C$ | $(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$ |
| value restriction | $\forall R.C$ | $(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$ |
| atleast restriction | $\geq nS.C$ | $(\geq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$ |
| atmost restriction | $\leq nS.C$ | $(\leq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$ |
| data type exists | $\exists T.d$ | $(\exists T.d)^{\mathcal{I}} = \{x \mid \exists y : (x, y) \in T^{\mathcal{I}} \text{ and } y \in d^{\mathbf{D}}\}$ |
| data type value | $\forall T.d$ | $(\forall T.d)^{\mathcal{I}} = \{x \mid \forall y : (x, y) \in T^{\mathcal{I}} \Rightarrow y \in d^{\mathbf{D}}\}$ |

A DL *knowledge base* is a set of *axioms*, where an axiom is of one of the following three types, respectively indicating subset relations between concept expressions, subset relations between role expressions, and transitivity of roles.

- *terminological axioms* $C \sqsubseteq D$ with $C$ and $D$ concept expressions,
- *role axioms* $R \sqsubseteq S$ where $R, S$ may be inverse roles with the underlying roles both abstract or both concrete, and
- *transitivity axioms* $\mathsf{Trans}(R)$ for an (inverse) abstract role.

We often write $A \equiv B$ if both $A \sqsubseteq B$ and $B \sqsubseteq A$ hold in a knowledge base. If the knowledge base contains an axiom $\mathsf{Trans}(R)$, we call $R$ *transitive*. For the role axioms in a knowledge base, we define $\sqsubseteq^*$ as the transitive closure of $\sqsubseteq$. A *simple role* $R$ in a knowledge base is a role that is not transitive nor does it have any transitive subroles (w.r.t. to reflexive transitive closure $\sqsubseteq^*$ of $\sqsubseteq$). Note that, if the particular DL allows for inverted roles, for $R \sqsubseteq S$ a role axiom with (possibly inverted) abstract roles, we always assume $R^- \sqsubseteq S^-$ is also present in the knowledge base; similarly, if $\mathsf{Trans}(R)$ is in the knowledge base, we assume $\mathsf{Trans}(R^-)$ is as well.

Traditionally, a knowledge base contains also assertional statements like $C(a)$ (or $R(a,b)$) which intuitively means that the individual $a$ is an instance of $C$ ($a$ is related to $b$ by means of the role $R$). However, in the presence of individuals, we can simulate the assertions with terminological axioms:

$$C(a) \Leftrightarrow \{a\} \sqsubseteq C$$
$$R(a,b) \Leftrightarrow \{a\} \sqsubseteq \exists R.\{b\}$$

Terminological and role axioms express a subset relation: an interpretation $\mathcal{I}$ *satisfies* an axiom $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ ($R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$). An interpretation satisfies a transitivity axiom $\mathsf{Trans}(R)$ if $R^{\mathcal{I}}$ is a transitive relation. An interpretation is a *model* of a knowledge base $\Sigma$ if it satisfies every axiom in $\Sigma$. A concept $C$ is *satisfiable* w.r.t. $\Sigma$ if there is a model $\mathcal{I}$ of $\Sigma$ such that $C^{\mathcal{I}} \neq \emptyset$. The *number restrictions* (at most and at least) are always such that the role $R$ in, e.g., $\geq nR.C$, is simple; this in order to avoid undecidability of satisfiability checking (see, e.g. , [HST99]).

*Example 2.24.* The human resources department specifies the company's structure: (a) *Personnel* consists of *Management*, *Workers* and *john*, (b) *john* is the boss of some manager, and (c) managers only take orders from other managers and are the boss of at least three *Workers*. This corresponds to the following knowledge base $\Sigma_1$:

$$\begin{aligned} \mathit{Personnel} &\equiv \mathit{Management} \sqcup \mathit{Workers} \sqcup \{\mathit{john}\} \\ \{\mathit{john}\} &\sqsubseteq \exists \mathit{boss}.\mathit{Management} \\ \mathit{Management} &\sqsubseteq (\forall \mathit{take\_orders}.\mathit{Management}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Workers}) \end{aligned}$$

A model of this knowledge base is $\mathcal{I} = (\{j, w_1, w_2, w_3, m\}, \cdot^{\mathcal{I}})$, with $\cdot^{\mathcal{I}}$ defined by $\mathit{Workers}^{\mathcal{I}} = \{w_1, w_2, w_3\}$, $\mathit{Management}^{\mathcal{I}} = \{m\}$, $\{\mathit{john}\}^{\mathcal{I}} = \{j\}$, $\mathit{Personnel}^{\mathcal{I}} = \{j, w_1, w_2, w_3, m\}$, $\mathit{boss}^{\mathcal{I}} = \{(j,m), (m,w_1), (m,w_2), (m,w_3)\}$, and $\mathit{take\_orders}^{\mathcal{I}} = \emptyset$.

A particular DL is $\mathcal{ALC}$: the DL where concept expressions may be formed using atomic concepts, concept conjunction and disjunction, negation of concept expressions, exists restrictions, and value restrictions. Satisfiability checking of $\mathcal{ALC}$ concept expressions w.r.t. a knowledge base containing only terminological axioms is EXPTIME-complete [Tob01].

If $\mathcal{ALC}$ knowledge bases allow for transitivity axioms, we speak of $\mathcal{S}$; adding support for role axioms leads to the DL $\mathcal{SH}$ and, subsequently adding inverse roles gives the DL $\mathcal{SHI}$. The DL $\mathcal{SHI}$ extended with qualified number restrictions is $\mathcal{SHIQ}$, where satisfiability checking of $\mathcal{SHIQ}$ concept expressions w.r.t. $\mathcal{SHIQ}$ knowledge bases (i.e., with terminological, role, and transitivity axioms) is EXPTIME-complete [Tob01].

Adding nominals to $\mathcal{SHIQ}$ gives the DL $\mathcal{SHOIQ}$ where reasoning, i.e. satisfiability checking as above, is NEXPTIME-complete [Tob01]. The DL corresponding to the ontology language OWL DL, a fragment of the language

OWL, is $\mathcal{SHOIN}(\mathbf{D})$, i.e., $\mathcal{SHOIQ}$, with, instead of qualified number restrictions, unqualified number restrictions ($\mathcal{N}$ instead of $\mathcal{Q}$), and with added support for data types (the $\mathbf{D}$); reasoning in OWL DL is NEXPTIME-complete [HPS04a].

A final DL that we mention is $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ as it plays an important role in Chapter 6. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ differs from the DL $\mathcal{SHOIN}(\mathbf{D})$ by its lack of inverted roles, data types ($\mathbf{D}$) and transitivity of roles (which distinguishes $\mathcal{S}$ from $\mathcal{ALC}$); it adds qualified number restrictions and the role constructs $\sqcup$ and $\sqcap$ though.

As we noted above, OWL does not have the unique name assumption. However, this may lead to unintuitive results as noted in [dBPLF05]. E.g., assume we have assertions $hasPassenger(seat1, mary)$ and $hasPassenger(seat1, john)$ together with an axiom[16]

$$\top \sqsubseteq (\leq 1\ hasPassenger)$$

which indicates the *hasPassenger* role is functional. In OWL, this yields to the conclusion that John and Mary are the same person, while with the unique name assumption this gives a contradiction.

### 2.3.3 Computation Tree Logic

*Temporal logics* [Eme90] 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)* [Eme90, SC85] and *computation tree logic (CTL)* [Eme90, EH82, CES86], which, among others, 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.

We introduce in this subsection the temporal logic CTL. Let AP be the finite set of available proposition symbols. *Computation tree logic (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.

[16] $\top$ is the universal concept, i.e., for any interpration $\mathcal{I}$, $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

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 $\pi$ 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 $\pi_i$. 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 $(s,t) \in R$, $K, t \models p$,
- $K, s \models \mathsf{E}(p\ \mathsf{U}\ q)$ iff there exists a path $\pi$ 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 $\pi$ 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 2.25.* Consider the expression of *absence of starvation* $t \Rightarrow \mathsf{AF}c$ [CES86] for a process in a mutual exclusion problem[17] . 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.

[17] 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., [EC82, Eme90, CES86, AE01, HR00, MW84].

We will usually represent structures by diagrams as in Figure 2.16, 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 2.16. 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 $t \notin L(s_2)$.

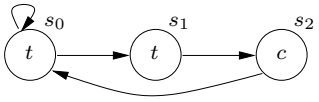


**Fig. 2.16.** Example Structure $t \Rightarrow \mathsf{AF}c$

Satisfiability checking of CTL formulas is EXPTIME-complete.

**Theorem 2.26 ([Eme90]).** *The problem of testing satisfiability for CTL is complete for deterministic exponential time.*

*Proof Sketch.* Membership in EXPTIME is based on a tableau construction from which a model can be generated [Eme90, EC82]. The tableau can be constructed in time that is exponential in the size of the formula such that membership follows. Hardness can be shown by a reduction from alternating polynomial-space bounded TMs [Eme90]. □

### 2.3.4 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., [Mos74]. Also in the presence of infinite models, so-called *fixed point logic (FPL)* proves to be an interesting logic [Flu99]. E.g., a decidable subclass of FPL is the *guarded fixed point logic* [GW99], which lifts propositional $\mu$-calculus [Koz83] 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 [GW99], i.e., as an extension of first-order logic, where formulas may additionally be *fixed point formulas* of the form

$$[\text{LFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{X}) \quad \text{or} \quad [\text{GFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{X}) \;, \tag{2.1}$$

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 $\psi(W,\mathbf{X})$.[18]

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

$$\psi^{(U,M),\chi}(S) \equiv \{\mathbf{x} \in U^n \mid (U,M), \chi \cup \{W \to S\} \models \psi(W,\mathbf{x})\} \;, \tag{2.2}$$

where $\chi \cup \{W \to S\}$ is the valuation $\chi$ 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 $\chi$ and write just $\psi^{(U,M)}$. By definition, $W$ appears only positively in $\psi$ such that $\psi^{(U,M),\chi}$ is monotonic on sets of $n$-ary $U$-tuples and thus has a least and greatest fixed point [Tar55], which we denote by $\text{LFP}(\psi^{(U,M),\chi})$ and $\text{GFP}(\psi^{(U,M),\chi})$ respectively. Finally, we have that

$$(U,M), \chi \models [\text{LFP } W\mathbf{X}.\psi(W,\mathbf{X})](\mathbf{x}) \iff \mathbf{x} \in \text{LFP}(\psi^{(U,M),\chi}) \;, \tag{2.3}$$

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 $\eta$ and $\theta$ is a subformula of $\varphi$, or $S$ occurs in $\varphi$ and $\psi$ is a subformula of $\eta$. We can eliminate greatest fixed point formulas from a formula, by the equivalence:

$$[\text{GFP } W\mathbf{X}.\psi] \equiv \neg[\text{LFP } W\mathbf{X}.\neg\psi[W/\neg W]] \;, \tag{2.4}$$

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.

As in [Grä02a], we define

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

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

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 $\alpha$ such that $\psi^{(U,M)} \uparrow \alpha = \text{LFP}(\psi^{(U,M)})$.

*Example 2.27.* Take the conjunction of the following formulas, i.e., the infinity axiom[19] from [GW99]:

$$\exists X, Y \cdot F(X, Y) \tag{2.5}$$

$$\forall X, Y \cdot (F(X, Y) \Rightarrow (\exists Z \cdot F(Y, Z))) \tag{2.6}$$

$$\forall X, Y \cdot F(X, Y) \Rightarrow [\text{LFP } WX.\forall Y \cdot F(Y, X) \Rightarrow W(Y)](X) \tag{2.7}$$

A model of these formulas contains at least one $F(x, y)$ (by formula (2.5)), which then leads to a $F$-chain by formula (2.6). Formula (2.7) ensures that each element $x$ is on a well-founded chain (and thus formula (2.6) actually generates an infinite chain).

For example, take an infinite interpretation $(U, M)$ with $U = \{x_0, x_1, \ldots\}$ and $M = \{F(x_0, x_1), F(x_1, x_2), \ldots\})$. Clearly, this model satisfies formulas (2.5) and (2.6). Denote $\psi \equiv \forall Y \cdot F(Y, X) \Rightarrow W(Y)$, then we calculate $\text{LFP}(\psi^{(U,M)})$ as follows:

$$\begin{aligned}
\psi^{(U,M)} \uparrow 0 &= \emptyset \\
\psi^{(U,M)} \uparrow 1 &= \{x_0\} \\
\psi^{(U,M)} \uparrow 2 &= \{x_0, x_1\} \\
&\vdots
\end{aligned}$$

Indeed, $\psi^{(U,M)} \uparrow 1 = \psi^{(U,M)}(\emptyset)$ such that $\psi$ is reduced to $\forall Y \cdot F(Y, X) \Rightarrow$ **false**, or, equivalently, $\neg\exists Y \cdot F(Y, X)$, i.e., we want those $X$'s that have no predecessor, which is exactly $x_0$. In the next step, we deduce again $x_0$ plus all successors of $x_0$, yielding $\{x_0, x_1\}$. Finally, we have that $\text{LFP}(\psi^{(U,M)}) = \{x_0, x_1, x_2, \ldots\}$ such that formula (2.7) is also satisfied by $(U, M)$.

Moreover, no finite model can satisfy the above formulas. First, note that a model $(U, M)$ cannot contain loops, i.e., $\{F(x_0, x_1), \ldots, F(x_n, x_0)\} \subseteq M$ is not possible. Assume otherwise. By formula (2.7), $x_n \in \text{LFP}(\psi^{(U,M)})$, and thus there is some ordinal $\alpha$ such that $x_n \in \psi^{(U,M)} \uparrow \alpha$. By the definition of $\psi^{(U,M)} \uparrow \alpha$, we then have that $x_{n-1} \in \psi^{(U,M)} \uparrow \alpha - 1$. Since we have a loop, one can continue this way and eventually deduce for some $x_i$ that $x_i \in \psi^{(U,M)} \uparrow 0 = \emptyset$, a contradiction. Thus $M$ does not contain loops.

By the first formula, we need some $F(X, Y) \in M$ if $M$ is to be a model. Since $M$ does not contain loops, we have some $F(x_0, x_1) \in M$. Formula (2.6) then calls for some $X$ such that $F(x_1, X) \in M$. Since $M$ cannot contain loops, $X$ must be different from $x_0$ and from $x_1$ and we need some new $x_2$. One can continue this way, and the loop-freeness of $M$ will impose the deduction of an infinite number of domain elements.

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

Finally, note that reasoning in FPL is undecidable as FPL is an extension of the undecidable FOL.

# 3

# Open Answer Set Programming

We define the open answer set semantics for logic programs in Section 3.1 and show in Section 3.2 that for unrestricted programs satisfiability checking for this semantics is undecidable. In Section 3.3, we introduce the notion of inverted predicates and we define an accompanying inverted world assumption. Section 3.4 identifies different syntactical subclasses of logic programs for which reasoning is shown to be decidable by a reduction to 2ATAs. We indicate in Section 3.5 how the restricted programs are still suitable to do conceptual modeling, in particular we show how to simulate a large part of *Object-Role Modeling* constructs. Finally, in Section 3.6, we discuss related work.

## 3.1 Open Answer Set Programming

Logic programs are defined as in Section 2.3.1. We additionally assume the existence of binary predicates $=$ and $\neq$, where $t = s$ is considered as an atom and $t \neq s$ is shorthand for *not* $t = s$. E.g., for $\alpha = \{X \neq Y, Y = Z\}$, we have $\alpha^+ = \{Y = Z\}$ and $\alpha^- = \{X = Y\}$. We call an atom for which the predicate is not equality or inequality, a *regular* atom. We further forbid the appearance of equality atoms in the positive head of a rule. The *Herbrand Base* of a program is modified such that it is now the set of ground *regular* atoms that can be formed using the language of the program, i.e., we do not allow $=$ in the Herbrand Base.

Regarding the semantics, we interpret $=$ directly, i.e., for an atom $s = t$ and an interpretation $I$, we have that $I \models s = t$ if $s$ and $t$ are equal terms. The other definitions in Section 2.3.1 remain unmodified.

For a program $P$, let $cts(P)$ be the constants in $P$, $vars(P)$ its variables, $preds(P)$ its predicates, $upreds(P)$ its unary predicates, and $bpreds(P)$ its binary predicates.

**Definition 3.1.** *A* universe *$U$ for a program $P$ is a non-empty countable superset of the constants in $P$: $cts(P) \subseteq U$. We call $P_U$ the ground program*

*obtained from $P$ by substituting every variable in $P$ by every possible element from $U$.*

Computing the (normal) answer sets of a program amounts to grounding the program $P$ with the universe $cts(P)$, resulting in $P_{cts(P)}$. In the following, a program $P$ is, unless specified otherwise, assumed to be a finite set of rules; infinite programs will only appear as byproducts of grounding a finite program with an infinite universe.

**Definition 3.2.** *An open interpretation of a program $P$ is a pair $(U, M)$ where $U$ is a universe for $P$ and $M$ is an interpretation of $P_U$. An open answer set of $P$ is an open interpretation $(U, M)$ of $P$ where $M$ is an answer set of $P_U$.*

*Example 3.3.* Take a program $P$:

$$\begin{aligned} p(X) &\leftarrow \mathit{not}\ q(X) \\ q(a) &\leftarrow \end{aligned}$$

Then $cts(P) = \{a\}$ such that the universes for $P$ have to be countable supersets of $\{a\}$. Some possible universes are $\{a\}$, $\{a, b\}$, and $\{a, x_1, x_2, \ldots\}$ where the latter is an infinite one. Grounding $P$ with $\{a, x_1, x_2, \ldots\}$ yields the program

$$\begin{aligned} p(a) &\leftarrow \mathit{not}\ q(a) \\ p(x_1) &\leftarrow \mathit{not}\ q(x_1) \\ p(x_2) &\leftarrow \mathit{not}\ q(x_2) \\ &\vdots \\ q(a) &\leftarrow \end{aligned}$$

which has an answer set $\{q(a), p(x_1), p(x_2), \ldots\}$ such that

$$(\{a, x_1, \ldots\}, \{q(a), p(x_1), \ldots\})$$

is an open answer set of $P$. The open answer set that corresponds to the normal answer set is $(\{a\}, \{q(a)\})$.

The main reasoning procedures we consider for the open answer set semantics are *satisfiability checking*, *consistency checking*, and *query answering*.

**Definition 3.4.** *A program $P$ is* consistent *if it has an open answer set. For an $n$-ary predicate $p$, appearing in $P$, $p$ is* satisfiable *w.r.t. $P$ if there exists an open answer set $(U, M)$ of $P$ and a $\mathbf{x} \in U^n$ such that $p(\mathbf{x}) \in M$.*

Note that the program $P$ in Example 3.3 is consistent, and that $p$ is satisfiable. This example also shows that the open and normal answer set semantics yield different conclusions: in the normal, closed world, answer set semantics one concludes that the predicate $p$ is not satisfiable since there is no answer set that contains a $p$-literal. In some settings, however, this may not be desirable: assume the rule $p(X) \leftarrow \mathit{not}\ q(X)$ plays the role of a schema constraint and

$q(a) \leftarrow$ is the particular data against which to check the schema constraint. One wants to conclude that $p$ is satisfiable, i.e., the schema constraint makes sense, since there are indeed cases, for other data, where $p$ can be populated. The open answer set semantics gives you this desired behavior.

Consistency checking can be reduced to satisfiability checking.

**Theorem 3.5.** *Let $P$ be a program. $P$ is consistent iff $p$ is satisfiable w.r.t. $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$, where $p$ is a unary predicate not appearing in $P$.*

*Proof.* For the "only if" direction, assume $P$ is consistent, then there is an open answer set $(U, M)$ of $P$. Take the open interpretation $(U, M \cup \{p(x)\})$ for some $x \in U$ ($U$ is non-empty by definition of a universe). Then $((P \cup \{p(X) \vee not\ p(X) \leftarrow\})_U)^{M \cup \{p(x)\}} = P_U^M \cup \{p(x) \leftarrow\}$, and, since $M$ is an answer set of $P_U^M$, we have that $M \cup \{p\}$ is an answer set of $P_U^M \cup \{p \leftarrow\}$, and thus $(U, M \cup \{p(x)\})$ is an open answer set of $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$ that contains $p$.

For the "if" direction, assume $p$ is satisfiable w.r.t. $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$, then there is an answer set $(U, M)$ of $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$ such that $p(x) \in M$ for some $x \in U$. Take $(U, M' \equiv M \setminus \{p(y) \mid y \in U\})$, then $M'$ is indeed an answer set of $P_U$ such that $(U, M')$ is an open answer set of $P$. □

For a ground literal $\alpha$, we define $P \models \alpha$ if for all open answer sets $(U, M)$ of $P$, $\alpha \in M$. Checking whether $P \models \alpha$ is called *query answering*. We can reduce query answering to consistency checking.

**Theorem 3.6.** *Let $P$ be a program. $P \models \alpha$ iff $P \cup \{\leftarrow \alpha\}$ is not consistent.*

*Proof.* For the "only if" direction, assume, by contradiction, that $P \cup \{\leftarrow \alpha\}$ is consistent, then there is an open answer set $(U, M)$ of $P \cup \{\leftarrow \alpha\}$ such that $\alpha \notin M$. Since $(U, M)$ is also an open answer set of $P$, we have a contradiction.

For the "if" direction, assume, by contradiction, that there is some open answer set $(U, M)$ of $P$ such that $\alpha \notin M$. Then, $(U, M)$ is an open answer set of $P \cup \{\leftarrow \alpha\}$ such that the latter is consistent, a contradiction. □

There are programs such that a predicate is only satisfiable w.r.t. that program by an infinite open answer set. We call such programs *infinity programs*.

*Example 3.7.* Take the program

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

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 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.

We can, without loss of generality, restrict ourselves in the rest of this dissertation, as in [LPV01], to programs without classical negation $\neg$.

**Theorem 3.8.** *Let $P$ be a program. Then,*

- *$(U, M)$ is an open answer set of $P$ iff $(U, (M \cup \{p'(\mathbf{x}) \mid \neg p(\mathbf{x}) \in M\}) \setminus \{\neg p(\mathbf{x})\})$ is an open answer set of $P'$, and*
- *$(U, M)$ is an open answer set of $P'$ iff $(U, (M \cup \{\neg p(\mathbf{x}) \mid p'(\mathbf{x}) \in M\}) \setminus \{p'(\mathbf{x})\})$ is an open answer set of $P$,*

*where $P'$ is $P$ with every occurrence of $\neg p(\mathbf{t})$ replaced by a new $p'(\mathbf{t})$ and the constraint $\leftarrow p(\mathbf{X}), p'(\mathbf{X})$ added.*

*Proof.* In answer set programming, a classically negated atom is basically treated as a new atom, while making sure that contradicting literals $a$ and $\neg a$ do not appear together in an interpretation. This is exactly what the construction of $P'$ encodes. □

For an open answer set $(U, M)$ of a ground program $P$ and an arbitrary universe $U'$ for $P$, we have that $(U', M)$ is also an open answer set, i.e., for ground programs the universe does not matter and one can stick to $cts(P)$ such as in the normal answer set semantics.

**Theorem 3.9.** *Let $P$ be a ground program. $(U, M)$ is an open answer set of $P$ iff $\forall U' \cdot (U', M)$ is an open answer set of $P$, where $U'$ is a universe for $P$.*

*Proof.* This follows from $\forall U' \cdot P_{U'} = P$. □

The groundness is necessary for Theorem 3.9 to hold.

*Example 3.10.* Take the unground program

$$q(a) \leftarrow not\ p(X)$$
$$p(a) \leftarrow$$

Then $(\{a, x\}, \{p(a), q(a)\})$ is an open answer set, while $(\{a\}, \{p(a), q(a)\})$ is not.

A type of rules that we will use frequently are *free rules*, i.e., 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. A predicate $p$ is *free* if there is a free rule $p(\mathbf{t}) \vee not\ p(\mathbf{t}) \leftarrow$. 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.

**Theorem 3.11.** *Let $P$ be a program and $p$ a free $n$-ary predicate. Then, $p$ is satisfiable w.r.t. $P$ iff $p'$ is satisfiable w.r.t. $P \cup \{p'(\mathbf{X}) \leftarrow p(\mathbf{X})\}$. Moreover, this is a linear reduction.*

*Proof.* For the "only if" direction, assume $p$ is satisfiable w.r.t. $P$, then there is an open answer set $(U, M)$ of $P$ such that $p(\mathbf{x}) \in M$ for an $n$-ary $\mathbf{x} \in U^n$. Define

$$M' \equiv M \cup \{p'(\mathbf{t}) \mid p(\mathbf{t}) \in M\} .$$

One can see that $(U, M')$ is an open answer set of $P' \equiv P \cup \{p'(\mathbf{X}) \leftarrow p(\mathbf{X})\}$.

For the "if" direction, assume $p'$ is satisfiable w.r.t. $P'$, then there is an open answer set $(U', M')$ of $P'$ that contains some $p'(\mathbf{x})$ and, by the minimality of $M'$ and the rule $p'(\mathbf{X}) \leftarrow p(\mathbf{X})$, also $p(\mathbf{x})$. Define

$$M \equiv M' \setminus \{p'(\mathbf{t})\} .$$

Then, $(U', M)$ is an open answer set of $P$ that satisfies $p$. □

In order to be able to define an *immediate consequence operator*, we restrict ourselves in the rest of this dissertation to programs where 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* [vEK76] $T$ 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 $T^{n+1}(B) = T(T^n(B))$.[1]

*Example 3.12.* Take the program $P$:

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

For an open interpretation $(U, M) = (\{x\}, \{c(x)\})$, $P_U^M$ is the program

$$\begin{aligned} a(x) &\leftarrow c(x) \\ c(x) &\leftarrow \end{aligned}$$

Such that $T^1 = \{c(x)\}$ and $T^2 = \{c(x), a(x)\}$.

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

*Proof.* [2] Assume $\exists a_1 \in M \cdot \forall n < \infty \cdot a_1 \notin T^n$.

- We write down all $r_1^{i_1} : a_1 \leftarrow \beta_1^{i_1} \in P_U^M$ such that $M \models \beta_1^{i_1}$ and such that there exists a regular atom $a_{1j_1}^{i_1} \in \beta_1^{i_1}$ such that $\forall n < \infty \cdot a_{1j_1}^{i_1} \notin T^n$. There always exists such an $r_1^{i_1}$, because otherwise[3] , we would have that for all $r : a_1 \leftarrow \beta$, $M \not\models \beta$ or $M \models \beta$ and for all regular $b_i \in \beta$ $\exists n_{b_i} < \infty \cdot b_i \in T^{n_{b_i}}$. Assume the latter, then $a_1 \in T^{\max n_{b_i}+1}$ with $\max n_{b_i} + 1$ finite, which is impossible. So for all $r : a_1 \leftarrow \beta$, $M \not\models \beta$, but then is $M \setminus \{a_1\}$ a model of $P_U^M$, which is also a contradiction (by the minimality of $M$).

[1] 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] Alternatively, one can show that $T$ is *finitizable*, i.e., $T(B) = \cup_{B' \subseteq B, |B'| < \infty} T(B')$ (see, e.g., [EG05]). Together with the monotonicity of $T$, the theorem follows.

[3] Note that there is a rule $r \in P_U^M$ with head $a_1$, otherwise $M \setminus \{a_1\}$ would be a model of $P_U^M$, contradicting the minimality of $M$.

- Next, we write down all $r^{i_1 i_2}_{1j_1} : a^{i_1}_{1j_1} \leftarrow \beta^{i_1 i_2}_{1j_1} \in P^M_U$ such that $M \models \beta^{i_1 i_2}_{1j_1}$ and such that there exists a regular atom $a^{i_1 i_2}_{1j_1 j_2} \in \beta^{i_1 i_2}_{1j_1}$ such that $\forall n < \infty \cdot a^{i_1 i_2}_{1j_1 j_2} \notin T^n$. There always exists such an $r^{i_1 i_2}_{1j_1}$, because otherwise, we would have that for all $r : a^{i_1}_{1j_1} \leftarrow \beta$, $M \not\models \beta$ or for all regular $b_i \in \beta$ $\exists n < \infty \cdot b_i \in T^n$. Assume the latter, then $a^{i_1}_{1j_1} \in T^{\max n+1}$ with $\max n + 1$ finite, which is impossible. So for all $r : a^{i_1}_{1j_1} \leftarrow \beta$, $M \not\models \beta$, but then is $M \setminus \{a^{i_1}_{1j_1}\}$ a model, which is also a contradiction.
- Continue this ad infinitum.

Let $M_2 = M \setminus \{a_1, a^{i_1}_{1j_1}, a^{i_1 i_2}_{1j_1 j_2}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$. Clearly, $M_2 \subset M$. Furthermore, $M_2$ is a model of $P^M_U$. Indeed, take an arbitrary $R : c \leftarrow \beta \in P^M_U$ with $M_2 \models \beta$. Because $M$ is a model we have that $c \in M$.

Assume $c \in \{a_1, a^{i_1}_{1j_1}, a^{i_1 i_2}_{1j_1 j_2}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$.

- Take $c = a_1$. If, for all $i_1$, $\beta \neq \beta^{i_1}_1$, then (since $M_2 \models \beta$, we have $M \models \beta$) for all regular $b_i \in \beta$ we have that $\exists n < \infty \cdot b_i \in T^n$. Then $a_1 \in T^{\max n+1}$ with $n$ finite, which is impossible. And thus there is a $i_1$, $\beta = \beta^{i_1}_1$, but $M_2 \not\models \beta^{i_1}_1$ (since the regular $a^{i_1}_{1j_1} \notin M_2$), and thus $M_2 \not\models \beta$. A contradiction.
- More general, take $c = a^{i_1 \ldots i_k}_{1j_1 \ldots j_k}$. If, for all $i_{k+1}$, $\beta \neq \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, then for all $b_i \in \beta$ we would have that $\exists n < \infty \cdot b_i \in T^n$. Then $a^{i_1 \ldots i_k}_{1j_1 \ldots j_k} \in T^{\max n+1}$ with $n$ finite, which is impossible. And thus there is a $i_{k+1}$, $\beta = \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, but $M_2 \not\models \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, and thus $M_2 \not\models \beta$. A contradiction.

Thus $c \notin \{a_1, a^{i_1}_{1j_1}, a^{i_1 i_2}_{1j_1 j_2}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$, and as a consequence $c \in M_2$. We conclude that $M_2$ is a model, in contradiction with the minimality of $M$. □

## 3.2 Undecidability of Open Answer Set Programming

We show the undecidability of open answer set programming for unrestricted programs by a reduction from the undecidable origin constrained domino problem (see Corollary 2.6, pp. 30). [4]

Let $\mathcal{D} = (D, H, V)$ be a domino system where $D = \{d_1, \ldots, d_k\}$. We define the corresponding *domino program* $[\mathcal{D}]$ as in Table 3.1. The rules in the $\mathbb{N} \times \mathbb{N}$ part of the table encode the plane: $h$ ($v$) makes sure that every point in $\mathbb{N} \times \mathbb{N}$

[4] Undecidability can be shown by a reduction from (undecidable fragments of) first-order logic as well. E.g., take the undecidable class of formulas of the form $\psi \equiv \forall \mathbf{X}_1 \forall \mathbf{X}_2 \ldots \forall \mathbf{X}_j \exists \mathbf{Y} \phi(\mathbf{X}_1, \ldots, \mathbf{X}_j, \mathbf{Y})$ for natural numbers $j$ (see, e.g., [BGG97], pp. 10). In these formulas, $\phi$ is a first-order logic formula that contains no unary predicates, at most one binary predicate and no predicates of higher arity. Furthermore, it contains no function symbols, no equality and no constants. Rewriting $\psi$ as $\neg\exists \mathbf{X}_1 \ldots \exists \mathbf{X}_j \neg\exists \mathbf{Y} \phi(\mathbf{X}_1, \ldots, \mathbf{X}_j, \mathbf{Y})$, we have the corresponding rules

**Table 3.1.** Domino Program

| | |
|---|---|
| $\mathbb{N} \times \mathbb{N}$ | $h: \leftarrow h(U, V_1), h(U, V_2), V_1 \neq V_2$ |
| | $v: \leftarrow v(U, V_1), v(U, V_2), V_1 \neq V_2$ |
| | $s: \leftarrow h(U, X), v(X, V_1), v(U, Y), h(Y, V_2), V_1 \neq V_2$ |
| | $hh: hh(U) \leftarrow h(U, X)$ |
| | $hv: hv(U) \leftarrow v(U, X)$ |
| | $hhc: \leftarrow not\ hh(U)$ |
| | $hvc: \leftarrow not\ hv(U)$ |
| | $f_1: h(U, V) \vee not\ h(U, V) \leftarrow$ |
| | $f_2: v(U, V) \vee not\ v(U, V) \leftarrow$ |

| | | |
|---|---|---|
| Domino Conditions | $d_1^{i,j}: \leftarrow d_i(U), d_j(V), h(U, V)$ | for $(d_i, d_j) \notin H$ |
| | $d_2^{i,j}: \leftarrow d_i(U), d_j(V), v(U, V)$ | for $(d_i, d_j) \notin V$ |
| | $d_3: \leftarrow not\ d_1(X), \ldots, not\ d_k(X)$ | |
| | $d_4^{i,j}: \leftarrow d_i(X), d_j(X)$ | for $i \neq j$ |
| | $f_3^i: d_i(U) \vee not\ d_i(U) \leftarrow$ | for $1 \leq i \leq k$ |

has only one horizontal right (vertical upper) successor, $s$ ensures that going up vertically and then horizontally right is the same as going horizontally right and then vertically up. $hh$ encodes a horizontal *has-successor* relation such that $hhc$ makes sure that every element in the domain has a horizontal successor, and similarly for $hv$ and $hvc$ in the vertical case. Finally, $f_1$ and $f_2$ are free rules; they can be used to introduce the $h$ and $v$ atoms.

The *domino conditions* ensure that we can construct a valid tiling out of an open answer set of the domino program: $d_1^{i,j}$ ($d_2^{i,j}$) ensure that horizontally (vertically) adjacent domino types are allowed according to $H$ ($V$), $d_3$ ensures that every position in the grid is assigned to some domino, and $d_4^{i,j}$ ensures that at most 1 domino type is assigned to each position. Finally, $f_3^i$ introduces the dominoes itself.

**Theorem 3.14.** *Let $\mathcal{D}$ be a domino system and $d$ a domino in $\mathcal{D}$. Then, $\mathcal{D}$ tiles the plane $\mathbb{N} \times \mathbb{N}$ such that $d$ is present in the tiling iff $d$ is satisfiable w.r.t. $[\mathcal{D}]$.*

$$\begin{aligned} \psi &\leftarrow not\ \psi' \\ \psi' &\leftarrow \psi''(\mathbf{X}_1, \ldots, \mathbf{X}_j) \\ \psi''(\mathbf{X}_1, \ldots, \mathbf{X}_j) &\leftarrow not\ \psi'''(\mathbf{X}_1, \ldots, \mathbf{X}_j) \\ \psi'''(\mathbf{X}_1, \ldots, \mathbf{X}_j) &\leftarrow \phi(\mathbf{X}_1, \ldots, \mathbf{X}_j, \mathbf{Y}) \end{aligned}$$

The translation can be trivially completed by adding rules that define $\phi$ (where one can assume that $\phi$ is in disjunctive normal form) and assuming the predicate in $\phi$ is defined by a free rule, ensuring the correspondence with first-order logic.

*Proof.* For the "only if" direction, assume $\mathcal{D}$ tiles the plane such that $d$ is present in the tiling $\tau$. Define $U \equiv \mathbb{N} \times \mathbb{N}$, and

$$M \equiv \{d(u) \mid \tau(u) = d\} \cup \{h((x,y),(x+1,y)) \mid x,y \in \mathbb{N}\} \cup \{v((x,y),(x,y+1)) \mid x,y \in \mathbb{N}\} \cup \{hh(u), hv(u) \mid u \in \mathbb{N} \times \mathbb{N}\} .$$

We have that $d$ is satisfied in $M$: $d$ is present in the tiling $\tau$, such that there is a $(x,y) \in \mathbb{N}\times\mathbb{N}$ with $\tau(x,y) = d$. By definition of $M$, we have that $d(x,y) \in M$[5]. It remains to show that $(U,M)$ is an open answer set of $[\mathcal{D}]$.

- $M$ is a model of $[\mathcal{D}]^M_U$. We check satisfiability of every rule in $[\mathcal{D}]^M_U$.
  - The free rules are satisfied.
  - Take $r : \leftarrow h(u,v_1), h(u,v_2) \in [\mathcal{D}]^M_U$, originating from $h$ (and thus $v_1 \neq v_2$), and assume $M \models \text{body}(r)$, then $u = (x,y)$, $v_1 = (x+1,y)$, and $v_2 = (x+1,y)$ such that $v_1 = v_2$, a contradiction.
  - Constraints originating from $v$ can be checked similarly.
  - Take $r : \leftarrow h(u,z_1), v(z_1,v_1), v(u,z_2), h(z_2,v_2) \in [\mathcal{D}]^M_U$, originating from $s$ (and thus $v_1 \neq v_2$), with $M \models \text{body}(r)$. Then $u = (x,y)$, $z_1 = (x+1,y)$, $v_1 = (x+1,y+1)$, $z_2 = (x,y+1)$, and $v_2 = (x+1,y+1)$, such that $v_1 = v_2$, a contradiction.
  - Take $hh(u) \leftarrow h(u,x)$, originating from $hh$. We have that $hh(u) \in M$ for all $u \in U$.
  - The rules originating from $hv$ can be done similarly.
  - For all $u \in U$, we have that $hh(u) \in M$ such that no constraint originating from $hhc$ is in $[\mathcal{D}]^M_U$.
  - The constraint $hvc$ can be done similarly.
  - Take $d_1^{i,j} : \leftarrow d_i(u), d_j(v), h(u,v)$. Assume $M \models \text{body}(d_1^{i,j})$, then $u = (x,y)$ and $v = (x+1,y)$ for some $(x,y) \in \mathbb{N} \times \mathbb{N}$. Since $d_i(u) \in M$, we have that $\tau(x,y) = d_i$, and, with $d_j(v) \in M$, we have that $\tau(x+1,y) = d_j$. Thus, since $\tau$ is a tiling, $(d_i,d_j) \in H$, a contradiction.
  - The rules $d_2^{i,j}$ can be done similarly.
  - $\tau$ is a function, thus for every $(x,y) \in \mathbb{N} \times \mathbb{N}$ there is a $d$ such that $\tau(x,y) = d$. Then, for every $u \in U$, there is a $d(u) \in M$, such that $[\mathcal{D}]^M_U$ does not contain constraints originating from $d_3$.
  - Take $d_4^{i,j} : \leftarrow d_i(u), d_j(u)$ with $M \models \text{body}(d_4^{i,j})$, then $\tau(u) = d_i$ and $\tau(u) = d_j$ with $d_i \neq d_j$; this is a contradiction, since $\tau$ is a function.
- $M$ is a minimal model of $[\mathcal{D}]^M_U$. Assume not, then there is a $N \subset M$, model of $[\mathcal{D}]^M_U$, such that there is some $l \in M \setminus N$. We distinguish between some cases:
  - $l = d(u)$. Since $d$ is free, we have that $d(u) \leftarrow\in [\mathcal{D}]^M_U$, such that $d(u) \in N$ since $N$ is a model of $[\mathcal{D}]^M_U$, a contradiction.
  - $l = h(u,z)$. Since $h$ is free, this can be done similarly, as can the case $l = v(u,z)$.

[5] We assume $d(x,y)$ is shorthand for $d((x,y))$

– $l = hh(u)$. Then, there is some $h(u,z) \in M$ and with $hh(u) \leftarrow h(u,z) \in [\mathcal{D}]_U^M$ and $l \notin N$, we have that $h(u,z) \notin N$ such that $h(u,z) \notin M$ (by the freeness of $h$), a contradiction. The case for $hv$ is similar.

For the "if" direction, assume that $(U, M)$ is an open answer set of $[\mathcal{D}]$ containing a $d(u_0)$ for $u_0 \in U$. For each $(x, y) \in \mathbb{N} \times \mathbb{N}$, define $\tau$ such that $\tau(x, y) \equiv d$ if there is a sequence

$$h(u_0, s_1), h(s_1, s_2), \ldots, h(s_{x-1}, s_x), v(s_x, t_1), v(t_1, t_2), \ldots, v(t_{y-1}, t_y)$$

in $M$ such that $d(t_y) \in M$; one thus assigns $d$ to position $(x, y)$ if for the element $t_y \in U$ that is obtained by "moving" horizontally $x$ times with $h$ and vertically $y$ times with $v$, we have that $d(t_y) \in M$ (thus $t_y$ corresponds with $(x, y)$).

First, we show that $\tau$ is well-defined:

- Every element in $\mathbb{N}\times\mathbb{N}$ has an image through $\tau$. Indeed, take $(x, y) \in \mathbb{N}\times\mathbb{N}$. We have that $u_0 \in U$. And thus $hh(u_0) \in M$ such that $h(u_0, s_1) \in M$ (by minimality of $M$). With a similar reasoning, we can thus deduce a sequence $h(u_0, s_1), h(s_1, s_2), \ldots, h(s_{x-1}, s_x), v(s_x, t_1), v(t_1, t_2), \ldots, v(t_{y-1}, t_y)$ in $M$. With $d_3$, we then have that there is some $d_i$ such that $d_i(t_y) \in M$, and thus $\tau(x, y) = d_i$, per definition of $\tau$.
- An element $(x, y) \in \mathbb{N} \times \mathbb{N}$ has at most one image: assume not, i.e., there are $d_i$ and $d_j$ for $i \neq j$ such that $\tau(x, y) = d_i$ and $\tau(x, y) = d_j$. We have then two sequences

  $$h(u_0, s_1), h(s_1, s_2), \ldots, h(s_{x-1}, s_x), v(s_x, t_1), v(t_1, t_2), \ldots, v(t_{y-1}, t_y)$$

  and

  $$h(u_0, s'_1), h(s'_1, s'_2), \ldots, h(s'_{x-1}, s'_x), v(s'_x, t'_1), v(t'_1, t'_2), \ldots, v(t'_{y-1}, t'_y)$$

  with $d_i(t_y) \in M$ and $d_j(t'_y) \in M$. Using the functionality of predicates $h$ and $v$ in $M$ (with constraints $h$ and $v$), one can deduce that $s_i = s'_i$, $1 \leq i \leq x$, and $t_i = t'_i$, $1 \leq i \leq y$. Such that $d_i(t_y) \in M$ and $d_j(t_y) \in M$ for $i \neq j$, a contradiction with $d_4^{i,j}$.

Next, we show that

- $(\tau(x, y), \tau(x + 1, y)) \in H$, and
- $(\tau(x, y), \tau(x, y + 1)) \in V$.

We only check the first condition (the second condition is similar). Take $d_i \equiv \tau(x, y)$ and $d_j \equiv \tau(x + 1, y)$. By definition of $\tau$, we have that

$$h(u_0, s_1), h(s_1, s_2), \ldots, h(s_{x-1}, s_x), v(s_x, t_1), v(t_1, t_2), \ldots, v(t_{y-1}, t_y) \in M$$

and

$$h(u_0, s'_1), h(s'_1, s'_2), \ldots, h(s'_{x-1}, s'_x), h(s'_x, s'_{x+1}), v(s'_{x+1}, t'_1), v(t'_1, t'_2), \ldots, v(t'_{y-1}, t'_y) \in M$$

with $d_i(t_y) \in M$ and $d_j(t'_y) \in M$. We show that $h(t_y, t'_y) \in M$, which leads, with $d_1^{i,j}$, to the conclusion that $(d_i, d_j) \in H$.

With the functionality of $h$, we can deduce that $s_i = s'_i$, $1 \leq i \leq x$. Thus, we have that $v(s_x, t_1) \in M$, $h(s_x, s'_{x+1}) \in M$, and $v(s'_{x+1}, t'_1) \in M$. We have that for $t_1$, there is some $h(t_1, t''_1) \in M$ (every element has a successor in $M$). Then, with constraint $s$, we have that $t''_1 = t'_1$, and $h(t_1, t'_1) \in M$.

We then have that $v(t_1, t_2) \in M$, $h(t_1, t'_1) \in M$, and $v(t'_1, t'_2) \in M$. We have that for $t_2$, there is some $h(t_2, t''_2) \in M$ (every element has a successor in $M$). Then, with constraint $s$, we have that $t''_2 = t'_2$ and $h(t_2, t'_2) \in M$.

Continuing this way, eventually, leads to $h(t_y, t'_y) \in M$. Figure 3.1 illustrates the two used sequences; one can use rule $s$ to subsequently prove $h(t_1, t'_1) \in M, \ldots, h(t_y, t'_y) \in M$.

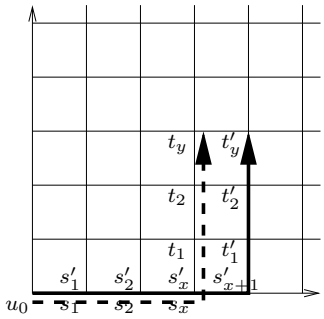


**Fig. 3.1.** Checking the Tiling Conditions

Finally, we have that $d$ is present in the tiling $\tau$: we have that $d(u_0) \in M$ and thus $\tau(0, 0) = d$ by definition of $\tau$. □

With a similar proof, we can reduce the unconstrained domino problem to consistency checking.

**Theorem 3.15.** *Let $\mathcal{D}$ be a domino system. Then, $\mathcal{D}$ tiles the plane $\mathbb{N} \times \mathbb{N}$ iff $[\mathcal{D}]$ is consistent.*

**Corollary 3.16.** *Satisfiability checking is undecidable.*

*Proof.* This is an immediate consequence of Corollary 2.6 and Theorem 3.14. □

**Corollary 3.17.** *Consistency checking is undecidable.*

*Proof.* This is an immediate consequence of the undecidability of the unconstrained domino problem (pp. 30) and Theorem 3.15. □

## 3.3 The Inverted World Assumption

We restrict ourselves in the remainder of this chapter and in the following chapter to programs with unary and binary predicates only. This allows us to introduce, similar to some DLs (see Section 2.3.2), *inverted predicates* $f^{\mathsf{i}}$ for a binary predicate $f$.[6] For a set $X$ of binary (possibly inverted) predicate names, $X^{\mathsf{i}} \equiv \{f^{\mathsf{i}} \mid f \in X\}$ where $f^{\mathsf{i}^{\mathsf{i}}} \equiv f$. We call atoms $f^{\mathsf{i}}(s,t)$, where $f$ is a predicate, *inverted* atoms. The Herbrand Base is still the set of ground regular atoms that can be formed from the language in $P$, but a language includes now the inverted predicates that can be formed: if there is a binary $f^{\mathsf{i}}$ or a binary $f$ in the program, the Herbrand Base contains atoms with predicate $f^{\mathsf{i}}$ and $f$. We further have that $\mathit{bpreds}(P)$ includes both $f$ and $f^{\mathsf{i}}$ for a $f$ or $f^{\mathsf{i}}$ in $P$.

*Example 3.18.* Take the ground program $P$:

$$q(a) \leftarrow f(a,b)$$
$$q(a) \leftarrow g^{\mathsf{i}}(a,b)$$

The Herbrand Base $\mathcal{B}_P$ is

$$\{q(a), q(b), f(a,b), f(b,a), f(a,a), f(b,b), f^{\mathsf{i}}(b,a), f^{\mathsf{i}}(a,b), f^{\mathsf{i}}(a,a), f^{\mathsf{i}}(b,b),$$
$$g(a,b), g(b,a), g(a,a), g(b,b), g^{\mathsf{i}}(b,a), g^{\mathsf{i}}(a,b), g^{\mathsf{i}}(a,a), g^{\mathsf{i}}(b,b)\} .$$

Possible interpretations of $P$ are then subsets of $\mathcal{B}_P$ as before[7] and $\mathit{bpreds}(P) = \{f, f^{\mathsf{i}}, g, g^{\mathsf{i}}\}$. The set of predicates in a program $P$ is then $\mathit{preds}(P) = \mathit{upreds}(P) \cup \mathit{bpreds}(P)$.

Intuitively, $f^{\mathsf{i}}(x,y)$ is defined, like in DLs, as the inverse of $f$. We formally capture this using an *inverted world assumption (IWA)*:

**Definition 3.19.** *Let $P$ be a ground program and $M$ an interpretation of $P$. Then $\mathit{IWA}(P,M)$ is the formula*

$$\forall f \in \mathit{bpreds}(P) \cdot f(x,y) \in M \iff f^{i}(y,x) \in M . \tag{3.1}$$

We define open answer sets *under IWA* by defining, for ground programs $P$, an interpretation $M$ *under IWA* of $P$ as an interpretation $M$ of $P$ such that $\mathit{IWA}(P,M)$ holds. Models, minimal models, and answer sets under IWA of ground program $P$ are then defined as usual but with interpretations under IWA, instead of just interpretations.

[6] We deviate from the convention in DLs to denote inverted roles as $f^-$, and instead denote them with $f^{\mathsf{i}}$, this to avoid confusion with the negative part $\beta^-$ of a body $\beta$ in (open) answer set programming.

[7] Remember that we assumed the absence of classical negation.

**Definition 3.20.** *An open interpretation under IWA of a program $P$ is a pair $(U, M)$ where $U$ is a universe for $P$ and $M$ is an interpretation under IWA of $P_U$. An open answer set under IWA of $P$ is an open interpretation under IWA $(U, M)$ of $P$ with $M$ an answer set under IWA of $P_U$. For an $n$-ary predicate $p$, $1 \leq n \leq 2$, appearing in $P$, $p$ is* satisfiable under IWA *w.r.t. $P$ if there exists an open answer set under IWA $(U, M)$ of $P$ and a $\mathbf{x} \in U^n$ such that $p(\mathbf{x}) \in M$. Consistency checking under IWA and query answering under IWA can be defined accordingly.*

*Example 3.21.* Modify the program of Example 3.7 by removing classical negation and adding inverted predicates to obtain the program

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

We replaced $\leftarrow y(Y_1, X), y(Y_2, X), Y_1 \neq Y_2$ by its counterpart $\leftarrow y^{\mathrm{i}}(X, Y_1), y^{\mathrm{i}}(X, Y_2), Y_1 \neq Y_2$ with inverses, and replaced $\neg \mathit{crash}(X)$ by $\mathit{not}\ \mathit{crash}(X)$. An open answer set under IWA $M$ that satisfies $P$ is (we omit $U$ if it is clear from $M$):

$$
\begin{aligned}
\{\mathit{restore}(x), \mathit{crash}(x), \mathit{backFail}(x), y(x, x_1), y^{\mathrm{i}}(x_1, x), \\
\mathit{backSucc}(x_1), y(x_1, x_2), y^{\mathrm{i}}(x_2, x_1) \\
\mathit{backSucc}(x_2), y(x_2, x_3), y^{\mathrm{i}}(x_3, x_2), \ldots\} \,.
\end{aligned}
$$

One can reduce consistency checking under IWA and query answering under IWA to satisfiability checking under IWA.

**Theorem 3.22.** *Let $P$ be a program.*

- *$P$ is consistent under IWA iff $p$ is satisfiable under IWA w.r.t. $P \cup \{p(X) \vee \mathit{not}\ p(X) \leftarrow\}$, where $p$ is a unary predicate not appearing in $P$.*
- *$P \models_{iwa} \alpha$ iff $P \cup \{\leftarrow \alpha\}$ is not consistent under IWA, where $\models_{iwa}$ denotes "query answering under IWA".*

*Proof.* Similar to the proofs of Theorem 3.5 and Theorem 3.6 (pp. 63). □

Satisfiability under IWA does not imply (normal) satisfiability.

*Example 3.23.* Take the program $P$:

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

Then $q$ is satisfiable under IWA by the open answer set

$$(\{x, y\}, \{q(x), f(x, y), f^{\mathrm{i}}(y, x)\}) .$$

However, there are no rules with an $f$-atom in the head such that $q$ is not satisfiable.

The other way around, we have that satisfiability does not imply satisfiability under IWA either.

*Example 3.24.* Take the program $P$:

$$\begin{aligned} f(X, Y) &\leftarrow \\ p(X) &\leftarrow \mathit{not}\ q(X) \\ q(X) &\leftarrow f^{\mathrm{i}}(X, Y) \\ f^{\mathrm{i}}(X, Y) \vee \mathit{not}\ f^{\mathrm{i}}(X, Y) &\leftarrow \end{aligned}$$

Then $p$ is satisfiable by the open answer set

$$(\{x, y\}, \{f(x, y), f(y, x), f(x, x), f(y, y), p(x), p(y)\}) .$$

However, $p$ is not satisfiable under IWA: the rule $f(X, Y) \leftarrow$ introduces all possible groundings of $f(X, Y)$, which then leads, by the IWA, to all possible groundings of $f^{\mathrm{i}}(X, Y)$, such that all possible groundings of $q(X)$ are in an open answer set under IWA. With the rule $p(X) \leftarrow \mathit{not}\ q(X)$ one then has that $p$ is never satisfiable.

If we allow for a modification of the program, we can, nevertheless reduce satisfiability checking under IWA to satisfiability checking.

**Theorem 3.25.** *Let $P$ be a program and $p$ a predicate in $P$. Then, $p$ is satisfiable under IWA w.r.t. $P$ iff $p$ is satisfiable w.r.t. $P'$, where $P'$ is $P$ with all $f^i$ replaced by $f'$ and the following rules added:*

$$\begin{aligned} f'(X, Y) &\leftarrow f(Y, X) \\ f(X, Y) &\leftarrow f'(Y, X) \end{aligned}$$

*Proof.* Intuitively, the added rules ensure that a $f'(x, y)$ is in an open answer set if $f(y, x)$ is (and similarly for a $f(x, y)$). Note that one still needs to motivate either $f$ or $f'$ with other rules (just as is the case with $f^{\mathrm{i}}$ and $f$).

For the "only if" direction, assume $(U, M)$ is an open answer set under IWA of $P$. Define $(U, M')$ with

$$M' \equiv (M \setminus \{f^{\mathrm{i}}(x, y)\}) \cup \{f'(x, y) \mid f^{\mathrm{i}}(x, y) \in M\}$$

One can show that $(U, M')$ is an open answer set of $P'$ that satisfies $p$.

For the "if" direction, assume $(U, M)$ is an open answer set of $P$. Define $(U, M')$ with

$$M' \equiv (M \backslash \{f'(x,y)\}) \cup \{f^{\mathsf{i}}(x,y) \mid f'(x,y) \in M\}$$

One can show that $(U, M')$ is an open answer set under IWA of $P$ that satisfies $p$. □

For programs that do not contain inverted predicates satisfiability is equivalent to satisfiability under IWA.

**Theorem 3.26.** *Let $P$ be a program without inverted predicates and $p$ a $n$-ary predicate, $1 \leq n \leq 2$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable under IWA w.r.t. $P$.*

*Proof.* For the "only if" direction, assume $p$ is satisfiable w.r.t. $P$. Then there is an open answer set $(U, M)$ of $P$ such that $p(\mathbf{x}) \in M$. Define

$$M' \equiv M \cup \{f^{\mathsf{i}}(x,y) \mid f(y,x) \in M\} .$$

Clearly, $(U, M')$ is an open interpretation under IWA. We prove that $(U, M')$ is an open answer set under IWA of $P$; it satisfies $p$ since $p(\mathbf{x}) \in M'$.

- $M'$ is a model under IWA of $P_U^{M'}$. Take a rule $\alpha^+ \leftarrow \beta^+ \in P_U^{M'}$ originating from $\alpha \leftarrow \beta \in P_U$ with $M' \models \alpha^-$ and $M' \models \mathit{not}\ \beta^-$. Assume $M' \models \beta^+$. We have then that $\alpha^+ \leftarrow \beta^+ \in P_U^M$ and $M \models \beta^+$ since $\alpha \leftarrow \beta$ does not contain inverted predicates. Thus, $\exists l \in \alpha^+ \cdot M \models l$, and $M' \models l$.
- $M'$ is a minimal model under IWA of $P_U^M$. Assume not, then there is a $N' \subset M'$, with $N'$ a model under IWA of $P_U^{M'}$. Define $N \equiv N' \backslash \{f^{\mathsf{i}}(x,y)\}$. Then, $N \subset M$: $N \subseteq M$ follows immediately, furthermore, we have that there is some $l \in M' \backslash N'$. Thus $l \notin N$. From $l \in M'$, we have that $l \in M$ or $l = f^{\mathsf{i}}(x,y)$ for $f(y,x) \in M$. In the former case, we are done, since then $l \in M \backslash N$; in the latter case, we have that $f^{\mathsf{i}}(x,y) \notin N'$, such that by the IWA, $f(y,x) \notin N$, and thus $f(y,x) \in M \backslash N$.
  We additionally have that $N$ is a model of $P_U^M$ which leads to a contradiction with the minimality of $M$.

For the "if" direction, assume $p$ is satisfiable under IWA w.r.t. $P$. Then, there is an open answer set under IWA $(U, M)$ of $P$ such that $p(\mathbf{x}) \in M$. Define

$$M' \equiv M \backslash \{f^{\mathsf{i}}(x,y)\} .$$

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

**Corollary 3.27.** *Satisfiability checking under IWA is undecidable.*

*Proof.* The domino program in Table 3.1 (pp. 68) contains only unary and binary predicates and no inverted predicates such that Theorem 3.26 is applicable. The result follows from the undecidability of satisfiability checking that was established in Corollary 3.16. □

**Theorem 3.28.** *Let $P$ be a program without inverted predicates. Then, $P$ is consistent iff $P$ is consistent under IWA.*

*Proof.* $P$ is consistent iff $p$ is satisfiable w.r.t. $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$, where $p$ is a unary predicate not appearing in $P$ (by Theorem 3.5). The latter holds iff $p$ is satisfiable under IWA w.r.t. $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$ (by Theorem 3.26) iff $P$ is consistent under IWA (by Theorem 3.22). □

**Corollary 3.29.** *Consistency checking under IWA is undecidable.*

*Proof.* The domino program in Table 3.1 (pp. 68) contains only unary and binary predicates and no inverted predicates such that Theorem 3.28 is applicable. The result follows from the undecidability of consistency checking that was established in Corollary 3.17. □

We define a modified immediate consequence operator for programs with inverted predicates. For a program $P$ and an open interpretation under IWA $(U, M)$ of $P$, $T_P^{\mathrm{i}(U,M)} : \mathcal{B}_{P_U} \rightarrow \mathcal{B}_{P_U}$ is defined as $T^{\mathrm{i}}(B) = B \cup \{a, a^{\mathrm{i}} \mid a \leftarrow \beta \in P_U^M \wedge B \models \beta\}$, where $a^{\mathrm{i}} \equiv a$ if $a$ is a unary atom and $f(s,t)^{\mathrm{i}} \equiv f^{\mathrm{i}}(t,s)$ otherwise. Additionally, we have $T^{\mathrm{i}^0}(B) = B$[8], and $T^{\mathrm{i}^{n+1}}(B) = T^{\mathrm{i}}(T^{\mathrm{i}^n}(B))$.

We can still motivate the presence of literals in open answer sets under the IWA in a finite way.

**Theorem 3.30.** *Let $P$ be a program and $(U, M)$ an open answer set under IWA of $P$. Then, $\forall a \in M \cdot \exists n < \infty \cdot a \in T^{i^n}$.*

*Proof.* Assume $\exists a_1 = a \in M \cdot \forall n < \infty \cdot a_1 \notin T^{\mathrm{i}^n}$.

- We write down all $r_1^{i_1} : a_1' \leftarrow \beta_1^{i_1} \in P_U^M$ with $a_1' = a_1$ or $a_1' = a_1{}^{\mathrm{i}}$ such that $M \models \beta_1^{i_1}$ and such that there exists a regular atom $a_{1j_1}^{i_1} \in \beta_1^{i_1}$ such that $\forall n < \infty \cdot a_{1j_1}^{i_1} \notin T^{\mathrm{i}^n}$. There always exists such an $r_1^{i_1}$, because otherwise, we have that for all $r : a_1' \leftarrow \beta$, $M \not\models \beta$ or $M \models \beta$ and for all regular $b_i \in \beta$ $\exists n < \infty \cdot b_i \in T^{\mathrm{i}^n}$. Assume the latter, then $a_1' \in T^{\mathrm{i}^{\max n+1}}$ with $n$ finite, which is impossible (because then $a_1 \in T^{\mathrm{i}^{\max n+1}}$). So for all $r : a_1' \leftarrow \beta$, $M \not\models \beta$, but then is $M \setminus \{a_1, a_1{}^{\mathrm{i}}\}$ a model under IWA, which is also a contradiction (by the minimality of $M$).
- Next, we write down all $r_{1j_1}^{i_1 i_2} : a'^{i_1}_{1j_1} \leftarrow \beta_{1j_1}^{i_1 i_2} \in P_U^M$ with $a'^{i_1}_{1j_1} = a_{1j_1}^{i_1}$ or $a'^{i_1}_{1j_1} = a_{1j_1}^{i_1}{}^{\mathrm{i}}$ such that $M \models \beta_{1j_1}^{i_1 i_2}$ and such that there exists a regular $a_{1j_1j_2}^{i_1 i_2} \in \beta_{1j_1}^{i_1 i_2}$ such that $\forall n < \infty \cdot a_{1j_1j_2}^{i_1 i_2} \notin T^{\mathrm{i}^n}$. There always exists such an $r_{1j_1}^{i_1 i_2}$, because otherwise, we would have that for all $r : a'^{i_1}_{1j_1} \leftarrow \beta$, $M \not\models \beta$

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

or $M \models \beta$ and for all regular $b_i \in \beta$ $\exists n < \infty \cdot b_i \in T^{\mathrm{i}^n}$. Assume the latter, then $a'^{i_1}_{1j_1} \in T^{\mathrm{i}^{\max n+1}}$ with $n$ finite, which is impossible (because then $a^{i_1}_{1j_1} \in T^{\mathrm{i}^{n+1}}$). So for all $r : a'^{i_1}_{1j_1} \leftarrow \beta$, $M \not\models \beta$, but then is $M \setminus \{a^{i_1}_{1j_1}, a^{i_1}_{1j_1}{}^{\mathrm{i}}\}$ a model under IWA, which is also a contradiction.

- Continue this ad infinitum.

Let $M_2 \equiv M \setminus \{a_1, a_1{}^{\mathrm{i}}, a^{i_1}_{1j_1}, a^{i_1}_{1j_1}{}^{\mathrm{i}}, a^{i_1 i_2}_{1j_1 j_2}, a^{i_1 i_2}_{1j_1 j_2}{}^{\mathrm{i}}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$. Clearly, $M_2 \subset M$. Furthermore, $M_2$ is a model under IWA of $P^M_U$. Indeed, take an arbitrary $R : c \leftarrow \beta \in P^M_U$ with $M_2 \models \beta$ (then $M \models \beta$), and, because $M$ is a model under IWA, $c \in M$.

Assume $c \in \{a_1, a_1{}^{\mathrm{i}}, a^{i_1}_{1j_1}, a^{i_1}_{1j_1}{}^{\mathrm{i}}, a^{i_1 i_2}_{1j_1 j_2}, a^{i_1 i_2}_{1j_1 j_2}{}^{\mathrm{i}}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$.

- Take $c = a_1$ or $c = a_1{}^{\mathrm{i}}$. If, for all $i_1$, $\beta \neq \beta^{i_1}_1$, then (because $M \models \beta$) for all regular $b_i \in \beta$ we have that $\exists n < \infty \cdot b_i \in T^{\mathrm{i}^n}$. Then $a_1 \in T^{\mathrm{i}^{\max n+1}}$ (or $a_1{}^{\mathrm{i}} \in T^{\mathrm{i}^{\max n+1}}$) with $n$ finite, which is impossible. And thus there is a $i_1$, $\beta = \beta^{i_1}_1$, but $M_2 \not\models \beta^{i_1}_1$, and thus $M_2 \not\models \beta$. A contradiction.
- More general, take $c = a^{i_1 \ldots i_k}_{1j_1 \ldots j_k}$ or $c = a^{i_1 \ldots i_k}_{1j_1 \ldots j_k}{}^{\mathrm{i}}$. If, for all $i_{k+1}$, $\beta \neq \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, then (because $M \models \beta$) for all $b_i \in \beta$ we have $\exists n < \infty \cdot b_i \in T^{\mathrm{i}^n}$. Then $a^{i_1 \ldots i_k}_{1j_1 \ldots j_k} \in T^{\mathrm{i}^{\max n+1}}$ (or $a^{i_1 \ldots i_k}_{1j_1 \ldots j_k}{}^{\mathrm{i}} \in T^{\mathrm{i}^{\max n+1}}$) with $n$ finite, which is impossible. And thus there is a $i_{k+1}$ such that $\beta = \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, but $M_2 \not\models \beta^{i_1 \ldots i_{k+1}}_{1j_1 \ldots j_k}$, and thus $M_2 \not\models \beta$. A contradiction.

Thus $c \notin \{a_1, a_1{}^{\mathrm{i}}, a^{i_1}_{1j_1}, a^{i_1}_{1j_1}{}^{\mathrm{i}}, a^{i_1 i_2}_{1j_1 j_2}, a^{i_1 i_2}_{1j_1 j_2}{}^{\mathrm{i}}, \ldots | i_1, i_2, \ldots, j_1, \ldots\}$, and as a consequence $c \in M_2$. We conclude that $M_2$ is a model, in contradiction with the minimality of $M$. □

## 3.4 Decidable Open Answer Set Programming under the IWA using 2ATAs

In this subsection, we identify an expressive class of programs, so-called *conceptual logic programs (CoLPs)*, for which reasoning is decidable.

Inspired by modal logics (and DLs in particular), we restrict arbitrary programs to CoLPs as to obtain programs such that if a unary predicate is satisfied by an open answer set, it can be satisfied by an open answer set with a tree structure, i.e., CoLPs have the *tree model property*. In [Var97], this tree model property is held responsible for the robust decidability of modal logics. Confirming this, the tree model property proves to be of significant importance to the decidability of satisfiability checking in CoLPs; it allows the reduction of satisfiability checking w.r.t. a CoLP to checking non-emptiness of a 2ATA.

### 3.4.1 Conceptual Logic Programs

Recall the program in Example 3.7 (pp. 63), which has an open answer set $(U, M)$ with $U \equiv \{x, x_1, \ldots\}$ and

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

One can rewrite this open answer set as an open answer set $(U', M')$ such that $U'$ is a tree: take $U' \equiv \{\varepsilon, 1, 11, 111, 1111, \ldots\}$ and

$$M' \equiv \{restore(\varepsilon), crash(\varepsilon), backFail(\varepsilon), y(\varepsilon, 1), \\ backSucc(1), \neg crash(1), y(1, 11) \\ backSucc(11), \neg crash(11), y(11, 111), \ldots\} .$$

Then $(U', M')$ is clearly also an open answer set of the program.

Observe that this open answer set can be encoded as a labeled tree $t : U' \to 2^{preds(P)}$: it maps nodes to a set of unary or binary predicates such that, for unary predicates $a$ in $P$ and binary predicates $f$ in $P$:

- $a(x) \in M'$ iff $a \in t(x)$, and
- $f(x, y) \in M'$ iff $y = x \cdot i \wedge f \in t(y)$.

Intuitively, unary literals $a(x)$ can be encoded in the label of node $x$ and binary literals $f(x, x \cdot i)$ can be encoded in the label of $x \cdot i$. A particular $f$ in the label of a node $x \cdot i$ indicates that $f(x, x \cdot i) \in M$ since each node $x \cdot i$ has the unique predecessor $x$. The open answer set $(U', M')$ can be encoded as the tree in Figure 3.2.

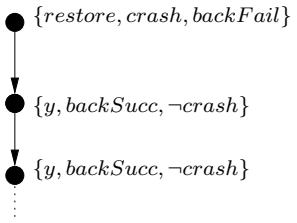


**Fig. 3.2.** Backup Example Tree

If we consider open answer sets under the IWA, we can also encode literals $f(x \cdot i, x)$, where the first argument is a successor of the second argument. Indeed, by the IWA we know that open answer sets under the IWA that contain $f(x \cdot i, x)$ also contain $f^{\mathsf{i}}(x, x \cdot i)$. Similarly as above, we place $f^{\mathsf{i}}$ in

the label of $x \cdot i$. Since $x \cdot i$ has only one predecessor, $x$, such a label uniquely identifies $f^{\mathsf{i}}(x, x \cdot i)$ and thus also $f(x \cdot i, x)$.

Similarly, we can encode $f^{\mathsf{i}}(x \cdot i, x)$ in open answer sets under the IWA since $f(x, x \cdot i)$ is present in the open answer set under the IWA: place $f$ in the label of $x \cdot i$.

*Example 3.31.* Modify the program in Example 3.21 by adding the rule

$$tomor(Y, X) \leftarrow y(X, Y) .$$

The modified program has then an open answer set under IWA $(U, M)$ with $U \equiv \{\varepsilon, 1, 11, 111, 1111, \ldots\}$ and

$$\{restore(\varepsilon), crash(\varepsilon), backFail(\varepsilon), y(\varepsilon, 1), y^{\mathsf{i}}(1, \varepsilon), tomor(1, \varepsilon), tomor^{\mathsf{i}}(\varepsilon, 1),$$
$$backSucc(1), y(1, 11), y^{\mathsf{i}}(11, 1), tomor(11, 1), tomor^{\mathsf{i}}(1, 11),$$
$$backSucc(11), y(11, 111), y^{\mathsf{i}}(111, 11), tomor(111, 11), tomor^{\mathsf{i}}(11, 111), \ldots\} ,$$

i.e., for every $y(u, v) \in M'$, add $tomor(v, u)$, and make sure the IWA holds. One can encode this open answer set under IWA as the labeled tree $t$ in Figure 3.3.

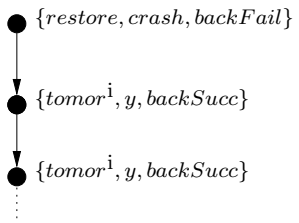


**Fig. 3.3.** Modified Backup Example Tree

Such a labeling function $t$ maps nodes to a set of unary and/or (possibly inverted) binary predicates such that, for unary predicates $a$ in $P$ and (possibly inverted) binary predicates $f$ in $P$:

- $a(x) \in M'$ iff $a \in t(x)$,
- $f(x, y) \in M'$ iff $y = x \cdot i \wedge f \in t(y)$ or $x = y \cdot i \wedge f^{\mathsf{i}} \in t(x)$.

Further note that the encoded trees in both of the above examples are minimal, in the sense that for every node $z \cdot i$ in the tree-shaped universe there is some $f(z, z \cdot i)$ in the open answer set under the IWA where $f$ is possibly inverted. Intuitively, the tree cannot contain *dangling* nodes.

*Example 3.32.* Take an open answer set under IWA $(\{\varepsilon, 1\}, \{a(\varepsilon), b(1)\})$ of some program $P$. Node 1 is dangling, since there is no binary literal connecting $\varepsilon$ and 1 in the open answer set.

A unary predicate $p$ is *tree satisfiable under IWA* if there is an open answer set under the IWA $(U, M)$ that can be encoded as a tree, as described above, and such that $p(\varepsilon) \in M$, i.e., the predicate $p$ is in the label of the root.

**Definition 3.33.** *Let $P$ be a program. A $p \in upreds(P)$ is* tree satisfiable under IWA *w.r.t. $P$ if there exists*

- *an open answer set under IWA $(U, M)$ of $P$ such that $U$ is a tree of bounded arity, and*
- *a labeling function $t : U \to 2^{preds(P)}$ such that*
  - *$p \in t(\varepsilon)$ and $t(\varepsilon)$ does not contain (possibly inverted) binary predicates, and*
  - *$z \cdot i \in U$, $i > 0$, iff there is some $f(z, z \cdot i) \in M$ where $f$ is possibly inverted, and*
  - *for $y \in U$, $q \in upreds(P)$, $f \in bpreds(P)$,*
    - *$q(y) \in M$ iff $q \in t(y)$, and*
    - *$f(x, y) \in M$ iff $y = x \cdot i \wedge f \in t(y)$ or $x = y \cdot i \wedge f^i \in t(x)$, where $f$ is possibly an inverted predicate.*

*We call such a $(U, M)$ a* tree model (under IWA) *and a program $P$ has the* tree model property (under IWA) *if the following property holds: if $p \in upreds(P)$ is satisfiable under IWA w.r.t. $P$ then $p$ is tree satisfiable under IWA w.r.t. $P$. The* label $\mathcal{L}(z)$ *of a node $z \in U$ is $\mathcal{L}(z) \equiv \{q \mid q \in t(z), q \in upreds(P)\}$.*

We will often denote a set like, e.g., $\{a(X), not\ b(X)\}$ as $\alpha(X)$ with $\alpha = \{a, not\ b\}$; similarly for sets of binary (possibly inverted) literals, e.g., $\{f(X,Y), not\ g^{-}(X,Y)\}$ will be denoted as $\alpha(X,Y)$ for $\alpha = \{f, not\ g^{-}\}$. If we only write $\alpha(X)$, without specifying $\alpha$, it is assumed that $\alpha$ is a (possibly empty) set of unary predicate names, possibly preceded with the negation as failure symbol, and similarly for $\alpha(X,Y)$.

We next identify a syntactical class of programs such that every program of that type has the tree model property.

**Definition 3.34.** *A* conceptual logic program (CoLP) *is a program with only unary and binary predicates, without constants, and such that any rule is of one of the following types,*

- free rules $a(X) \vee not\ a(X) \leftarrow$ *or* $f(X, Y) \vee not\ f(X, Y) \leftarrow$ *, where $f$ is possibly inverted (similarly for the subsequent rule types),*
- unary rules

$$r : a(X) \leftarrow \beta(X), \bigcup_{1 \le m \le k} \gamma_m(X, Y_m), \bigcup_{1 \le m \le k} \delta_m(Y_m), \psi$$

*where*

   1. $\psi \subseteq \bigcup_{1 \leq i \neq j \leq k} \{Y_i \neq Y_j\}$ *and* $\{=, \neq\} \cap \gamma_m = \emptyset$ *for* $1 \leq m \leq k$,
   2. $\forall Y_i \in vars(r) \cdot \gamma_i^+ \neq \emptyset$, *i.e., for variables* $Y_i$ *there is a positive atom that connects* $Y_i$ *and* $X$.
- binary rules $f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$ *with* $\gamma^+ \neq \emptyset$, $\{=, \neq\} \cap \gamma = \emptyset$,
- constraints $\leftarrow a(X)$ *or* $\leftarrow f(X, Y)$.

The term *conceptual logic program* refers to the ability of CoLPs to represent and reason with a diversity of conceptual knowledge, see, e.g., Section 3.5.

Intuitively, unary rules

$$r : a(X) \leftarrow \beta(X), \bigcup_{1 \leq m \leq k} \gamma_m(X, Y_m), \bigcup_{1 \leq m \leq k} \delta_m(Y_m), \psi$$

allow to deduce $a(X)$ if $\beta(X)$ hold, and for all *neighbors* $Y_m$, $\gamma_m(X, Y_m)$ as well as $\delta_m(Y_m)$ hold. Furthermore, one can impose that some of those neighbors must be different. E.g., a rule

$$a(X) \leftarrow f(X, Y_1), f(X, Y_2), Y_1 \neq Y_2$$

deduces $a$ at $X$ if $X$ has 2 different neighbors $Y_1$ and $Y_2$. We speak of neighbors in the following sense. For a tree model $(U, M)$ with associated labeling function $t$, we have that an $a(x) \in M$ corresponds to an $a \in t(x)$. In order to deduce $a$ at node $x$, one can use, e.g., the above rule: there must be different $y_1$ and $y_2$ such that $f(x, y_1) \in M$ and $f(x, y_2) \in M$. Since $(U, M)$ is a tree model, we must have that $y_1 = x \cdot i$, i.e., a successor of $x$ or $y_1 = x \cdot -1$, the predecessor of $x$. In the former case, we have that $y_2$ can be $x \cdot -1$ or $x \cdot j$ with $j \neq i$. In the latter case, we have that $y_2$ is some $x \cdot i$. Thus $y_1$ and $y_2$ are indeed *neighbors* of $x$. We then have the following cases for the labeling function associated with $(U, M)$:

1. $y_1 = x \cdot i$, $y_2 = x \cdot -1$, $f \in t(y_1)$, $f^{\mathrm{i}} \in t(x)$,
2. $y_1 = x \cdot i$, $y_2 = x \cdot j$, $f \in t(y_1)$, $f \in t(y_2)$, and
3. $y_1 = x \cdot -1$, $y_2 = x \cdot i$, $f^{\mathrm{i}} \in t(x)$, $f \in t(y_2)$.

The restriction

$$\forall Y_i \in vars(r) \cdot \gamma_i^+ \neq \emptyset$$

is necessary to have the tree model property. E.g.,

$$q(X) \leftarrow not\ f(X, Y), not\ q(Y)$$

is not a valid CoLP rule. Intuitively, one cannot transform an open answer set under IWA to a tree model: we have that $(\{x, y\}, \{q(x)\})$ is an open answer set under IWA, however, it is impossible to make a tree out of it since we need at least two domain elements $x$ and $y$ to make $q$ satisfiable, but we cannot connect them through a binary predicate.

A similar restriction, $\gamma^+ \neq \emptyset$, holds for binary rules. E.g., a rule

$$f(X, Y) \leftarrow v(X)$$

is not a valid CoLP rule; a true $v(x)$ may impose connections between $x$ and $y$ without $y$ being a successor of $x$.

The idea of ensuring such connectedness of models in order to have desirable properties, like decidability, is similar to the motivation behind the *guarded fragment* of predicate logic [ANB98]. In fact, in Chapter 5, we take the correspondence between the guarded fragment and syntactical classes of programs a step further.

A unary rule

$$r : a(X) \leftarrow \beta(X), \bigcup_{1 \leq m \leq k} \gamma_m(X, Y_m), \bigcup_{1 \leq m \leq k} \delta_m(Y_m), \psi$$

is a *live* rule if there is a $\gamma_m \neq \emptyset$. A unary predicate $a$ is live if there is a live rule $r$ with $a$ in head($r$) and $a$ is not free. The intuition behind a live predicate $a$ is that a new individual $y$ might need to be introduced in order to make $a(x)$ true for an existing $x$. We denote the set of live predicates for a CoLP $P$ with *live*($P$). A *degree* for the liveliness of a rule $r$, i.e., how many new individuals might need to be introduced to make the head true, is

$$\mathit{degree}(r) \equiv |\{m \mid \gamma_m \neq \emptyset\}| \,. \tag{3.2}$$

The *degree of a live predicate* $a$ in $P$ is

$$\mathit{degree}(a) \equiv \max\{\mathit{degree}(r) \mid a \in \text{head}(r)\} \,. \tag{3.3}$$

The *rank of a CoLP* $P$ is the sum of the degrees of the live predicates in $P$:

$$\sum_{a \in \mathit{live}(P)} \mathit{degree}(a) \,. \tag{3.4}$$

Intuitively, given a node in an encoded tree with a certain label that contains some unary predicates[9], every live unary predicate in the node appears in the head of some rule and its degree indicates precisely those neighboring nodes that need to be present to motivate the predicate in the node. The sum of those degrees corresponds then to the maximum branching of the tree at that node. The rank of a program is the maximum number of successor nodes one may need to introduce at any time.

*Example 3.35.* Take a program $P$ that contains, for predicates $a$, $b$, and $c$, the following rules:

$$r_1 : a(X) \leftarrow f(X, Y)$$
$$r_2 : a(X) \leftarrow g(X, Y_1), g(X, Y_2)$$
$$r_3 : b(X) \leftarrow h(X, Y)$$
$$r_4 : c(X) \leftarrow a(X)$$

[9] The binary predicates do not introduce new nodes in a tree – all variables of the body appear in the head.

Then, $degree(r_2) = 2$, $degree(r_1) = degree(r_3) = 1$ and $degree(r_4) = 0$, such that $\{a, b\} \subseteq live(P)$ and $degree(a) = 2$ and $degree(b) = 1$. The rank of $P$ is 3. Intuitively, for $\{a(x), b(x)\} \subseteq M$, where $(U, M)$ is some open answer set, one needs to motivate the presence of $a$ and $b$ in the label of $x$ in the corresponding tree. One needs a rule with applicable body and head predicate $a$ ($r_1$ or $r_2$) and a rule with applicable body and head predicate $b$ ($r_3$). Motivating $a$ in $x$ with $r_1$ may introduce one new successor $y$ of $x$ by the true $f(x, y)$; $r_2$ may introduce two new successors $y_1$ and $y_2$. In the worst case, this leads to the introduction of at most 2 new successors of $x$ to motivate $a$. For $b$ at $x$, one needs an applicable body of $r_3$ which introduces at most one new successor $y$. Combining this – $a$ and $b$ are present in $x$ – yields that one may need to introduce 3 new successors of $x$ to motivate both $a$ and $b$ at $x$. Of course, this is only in the worst case, in practice one can often reuse successors and/or the predecessor.

**Theorem 3.36.** *Conceptual logic programs have the tree model property.*

*Proof.* Take a CoLP $P$ and $p \in upreds(P)$ s.t. $p$ is satisfiable under IWA, i.e., there exists an open answer set $(U, M)$ under IWA with $p(u) \in M$. Let $n$ be the rank of $P$.

We first define $\theta : \{1, \ldots, n\}^* \to U$, a mapping from the complete $n$-ary tree to the domain $U$. Intuitively, $\theta$ associates some of the nodes in the complete tree with elements in the domain.

Initially, assume $\theta$ is undefined for the whole tree $\{1, \ldots, n\}^*$. If $\theta$ is defined on some node $x$, we will call the node $x$ *defined*. $\theta$ is constructed as follows:

- Define $\theta(\varepsilon) = u$.
- Assume we have considered, as in [Var98], every node in $\{1, \ldots, n\}^k$, for some $k$, as well as every successor node of the defined $z' \in \text{fr}(\{1, \ldots, n\}^k)$ until[10] $z \cdot m$ for some defined $z \in \text{fr}(\{1, \ldots, n\}^k)$. Consequently, we have considered the nodes $z \cdot 1, \ldots, z \cdot m$.
  Since $\theta$ is defined on $z$, we have that $\theta(z) \in U$. For every $q(\theta(z)) \in M$, there is, by Theorem 3.13, some $l < \infty$ s.t. $q(\theta(z)) \in T^{\text{i}^l}$. By definition of the immediate consequence operator, we have that there is a rule[11]

$$r_{q(\theta(z))} : q(\theta(z)) \leftarrow \beta^+[] \in P_U^M$$

  with $M \models \beta^+[]$, originating from $r : q(X) \vee \alpha \leftarrow \beta \in P$ such that
  – $M \models \alpha^-[]$,

[10] By saying "until", we assume that there is an ordering from left to right in the graphical representation of the tree.

[11] 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.

– $M \models not\ \beta^-[]$,

and $T^{i^{l-1}} \models \beta^+[]$. If $r$ is not live, we do nothing. Else, the body of $r_{q(\theta(z))}$ is of the form

$$\gamma^+(\theta(z)), \bigcup_i \gamma_i^+(\theta(z), y_i), \bigcup_i \delta_i^+(y_i)$$

with at least one $\gamma_i^+ \neq \emptyset$. Without loss of generality, we can assume that for all $i$, $\gamma_i^+ \neq \emptyset$. If there is a $z \cdot j \in \{z \cdot -1, z \cdot 1, \ldots, z \cdot m, \ldots, z \cdot (m+i-1))\}$ with $\theta(z \cdot j) = y_i$ then $\theta$ remains undefined on $z \cdot (m+i)$, otherwise $\theta(z \cdot (m+i)) = y_i$. Intuitively, if $\theta$ is already defined on a neighbor of $z$ as equal to $y_i$, there is no need to define $\theta$ on another successor as equal to $y_i$.

Define a labeled tree $t : \mathrm{dom}(\theta) \to 2^{preds(P)}$, where $\mathrm{dom}(\theta)$ are those elements for which $\theta$ is defined, as follows[12]:

- $t(\varepsilon) \equiv \{q \mid q(u) \in M\}$,
- $t(z{\cdot}i) \equiv \{q \mid q(\theta(z{\cdot}i)) \in M\} \cup \{f \mid f(\theta(z), \theta(z{\cdot}i)) \in M, f \text{ possibly inverted}\}$.

Define the open interpretation $(V, N)$ such that $V \equiv \mathrm{dom}(\theta)$ and

$$N \equiv \{q(z) \mid q \in t(z)\} \cup \{f(z, z \cdot i), f^{i}(z \cdot i, z) \mid f \in t(z \cdot i), f \text{ possibly inverted}\} .$$

We have to check that $(V, N)$ is a tree model under IWA satisfying $p$ according to Definition 3.33; it is easy to see that $(V, N)$ is indeed an open interpretation under IWA.

- *$(V, N)$ is an open answer set of $P$ such that $V$ is a tree of bounded arity.* The universe $V$ is indeed a tree of bounded arity such that remains to show that $N$ is a minimal model under IWA of $P_V^N$.
  Note that, for $z \in V$,
  – $q(z) \in N$ iff $q(\theta(z)) \in M$,
  – $f(z, z \cdot i) \in N$ iff $f(\theta(z), \theta(z \cdot i)) \in M$,
  – $f(z \cdot i, z) \in N$ iff $f(\theta(z \cdot i), \theta(z)) \in M$, and
  – $f(x, y) \in N$ then $f(\theta(x), \theta(y)) \in M$.
  We show that $N$ is a minimal model under IWA of $P_V^N$.
  – $N$ is a model under IWA of $P_V^N$. Rules in $P_V^N$ that originate from a free rule in $P$ are satisfied. Binary rules and constraints can be easily checked.
    Take a unary rule $r : a(x) \leftarrow \beta^+(x), \gamma_m^+(x, y_m), \delta_m^+(y_m) \in P_V^N$ originating from
    $a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), Y_i \neq Y_j \in P$[13] with $\beta^-(x) \cap N = \gamma_m^-(x, y_m) \cap N = \delta_m^-(y_m) \cap N = \emptyset$ and $y_i \neq y_j$ for $Y_i \neq Y_j$.
    Assume $\mathrm{body}(r) \subseteq N$. We have that

[12] In the following, we assume the $i$ in $z \cdot i$ is such that $i > 0$.
[13] We use a shorthand notation for rules.

- $\beta^-(\theta(x)) \cap M = \delta_m^-(\theta(y_m)) \cap M = \emptyset$.
- $\gamma_m^-(\theta(x), \theta(y_m)) \cap M = \emptyset$. Indeed, assume $g(\theta(x), \theta(y_m)) \in M$. Since $\gamma_m^+$ is not empty, there is some $f(x, y_m) \in \gamma^+(x, y_m) \subseteq N$, and thus $y_m$ is successor of $x$ or vice versa. But then we have that $g(x, y_m) \in N$, a contradiction.
- $\beta^+(\theta(x)), \gamma_m^+(\theta(x), \theta(y_m)), \delta_m^+(\theta(y_m)) \subseteq M$.
- $\theta(y_m) \neq \theta(y_k)$ if $y_m \neq y_k$. Indeed, both $y_m$ and $y_k$ are in $V$, thus $\theta$ is defined on both $y_m$ and $y_k$. Since $\gamma_m^+$ and $\gamma_k^+$ are not empty, we have that $y_m$ and $y_k$ are among the successors of $x$ or the predecessor of $x$. By construction of $\theta$, we then have that $\theta(y_m) \neq \theta(y_k)$.

Thus $r' : a(\theta(x)) \leftarrow \beta^+(\theta(x)), \gamma_m^+(\theta(x), \theta(y_m)), \delta_m^+(\theta(y_m)) \in P_U^M$ with $\text{body}(r') \subseteq M$, such that $a(\theta(x)) \in M$ and thus $a(x) \in N$.

– $N$ is a minimal model under IWA of $P_V^N$. Assume not, then there is a model $N' \subset N$ of $P_V^N$. We show per induction that
  - if $q(z) \in N$ and $q(\theta(z)) \in T^{\mathrm{i}^n}_M$, then $q(z) \in N'$,
  - if $f(z, z \cdot i) \in N$ and $f(\theta(z), \theta(z \cdot i)) \in T^{\mathrm{i}^n}_M$, then $f(z, z \cdot i) \in N'$, and
  - if $f(z \cdot i, z) \in N$ and $f(\theta(z \cdot i), \theta(z)) \in T^{\mathrm{i}^n}_M$, then $f(z \cdot i, z) \in N'$.
  - Take $n = 1$, and assume $q(z) \in N$ and $q(\theta(z)) \in T^{\mathrm{i}^1}_M$, then there is a $q(\theta(z)) \leftarrow \ \in P_U^M$
    - originating from a $q(X) \vee not\ q(X) \leftarrow\in P$, such that $q(z) \leftarrow\in P_V^N$ and thus $q(z) \in N'$, or
    - originating from a rule

$$r : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), Y_i \neq Y_j \in P$$

      with $\text{body}(r)^+ = \emptyset$. We have that there can be no $Y_i$ in the body of $r$, otherwise ${\gamma_i}^+$ should be non-empty, which is not possible. We have that $\beta^-(z) \cap N = \emptyset$ such that $q(z) \leftarrow\in P_V^N$, and thus $q(z) \in N'$.

    The binary cases can be done similarly.
  - Assume it is true for $n - 1$ (IH).
  - For $n$, there is a rule

$$r' : q(\theta(z)) \leftarrow \beta^+(\theta(z)), \gamma_m^+(\theta(z), y_m), \delta_m^+(y_m) \in P_U^M$$

    with $\text{body}(r') \subseteq T^{\mathrm{i}^{n-1}}_M$, $\beta^-(\theta(z)) \cap M = \gamma_m^-(\theta(z), y_m) \cap M = \delta_m^-(y_m) \cap M = \emptyset$, and $y_i \neq y_j$ if $Y_i \neq Y_j$ in the originating rule. Assume $r'$ is the rule we took in the construction of $\theta$ for $q(\theta(z)) \in T^{\mathrm{i}^n}_M$.
    By the construction of $\theta$ we have for every $m$ where $\gamma_m \neq \emptyset$, a $z \cdot m_i$ such that $\theta(z \cdot m_i) = y_m$. Note that $m_i$ may be equal to $-1$.
    We have that $\beta^-(z) \cap N = \emptyset$, and, $\gamma_m^-(z, z \cdot m_i) \cap N = \delta_m^-(z \cdot m_i) \cap N = \emptyset$. Moreover, we have that $z \cdot m_i \neq z \cdot k_j$ if $y_m \neq y_k$. For the latter,

assume $z \cdot m_i = z \cdot k_j$, then, since $\theta$ is a function, $\theta(z \cdot m_i) = \theta(z \cdot k_j)$ and thus $y_m = y_k$, a contradiction.
Thus $a(z) \leftarrow \beta^+(z), \gamma_m^+(z, z \cdot m_i), \delta_m^+(z \cdot m_i) \in P_V^N$, with a body true in $N'$ by induction, and $a(z) \in N'$.
The binary cases are similar.

Since $N' \subset N$, there must be $a(z) \in N \setminus N'$ or $f(z, v) \in N \setminus N'$, such that, by the previous, we have a contradiction.

- *$t$ is a labeling function such that*
  - *$p \in t(\varepsilon)$, and $t(\varepsilon)$ does not contain (possibly inverted) predicates.* Immediate from the definition of $t$.
  - *$z \cdot i \in V$, $i > 0$, iff there is some $f(z, z \cdot i) \in N$ where $f$ is possibly inverted.* If $z \cdot i \in V$, $\theta$ is defined on $z \cdot i$, and there is some $y \in U$ such that $\theta(z \cdot i) = y$ for some $f(\theta(z), \theta(z \cdot i) \in M$. By definition of $N$, we then have that $f(z, z \cdot i) \in N$.
  - *for $y \in V$, $q \in upreds(P)$, $f \in bpreds(P)$,*
    - *$q(y) \in N$ iff $q \in t(y)$.* By the definition of $N$.
    - *$f(x, y) \in N$ iff $y = x \cdot i \wedge f \in t(y)$ or $x = y \cdot i \wedge f^i \in t(x)$, where $f$ is possibly an inverted predicate.* By the definition of $N$.

□

### 3.4.2 Decidability of Conceptual Logic Programs

For a given conceptual logic program with a unary predicate to test for satisfiability, we construct a 2ATA such that we can reduce satisfiability checking under IWA to checking non-emptiness of the automaton.

We define the notion of *well-behaved trees*. Well-behaved trees are trees with certain basic properties that make the definition of the main 2ATA for a CoLP less cumbersome:

- The root cannot contain binary predicates since a binary predicate in a label indicates that there is a connection in the open answer set with the predecessor (and the root has no predecessor).
- We allow for nodes that are labeled with $\{dummy\}$ and make sure that all successors of such nodes are labeled likewise. The dummy nodes allow us to construct infinite trees from finite open answer sets.

**Definition 3.37.** *An infinite $k$-ary tree $t : T \to 2^{preds(P)} \cup \{\{dummy\}\}$ for a program $P$ with rank $k$ is well-behaved if*

- *The root label does not contain binary predicates (possibly inverted) from $P$,*
- *If the label of a node is $\{dummy\}$, the labels of all its successors are $\{dummy\}$.*

One can easily construct a 2ATA that accepts exactly the set of well-behaved trees of a program $P$; call this the *well-behaved automaton of $P$*.

Let $P$ be a CoLP with rank $k$ and $p$ a unary predicate in $P$. We define the 2ATA $A_{p,P}$ as the intersection of the well-behaved automaton of $P$ and the 2ATA $(\Sigma, Q, \delta, q_0, \Omega)$:

### The Alphabet $\Sigma$

The alphabet of the automaton is $2^{preds(P)} \cup \{\{dummy\}\}$, i.e., the label of a node of the input tree is either a set of unary and binary (possibly inverted) predicates or the dummy label $\{dummy\}$.

### The Transition Function $\delta$

Instead of first defining the states, we immediately define the transition function and assume the states we introduce in this definition are also defined in $Q$.

- The transition for the initial state $q_0$ is

$$\delta(q_0, n) = p \in n \wedge (0, q_1) \ . \tag{3.5}$$

  for any $n \in 2^{preds(P)} \cup \{\{dummy\}\}$. In the initial state, we check whether $p$ is in the label $n$, i.e., we ensure that the infinite tree corresponds to an open interpretation that makes $p$ satisfiable. We next enter the state $q_1$, which will check every node of our tree for conditions that make sure that the tree corresponds to an open answer set.
- The transition for the recurring state $q_1$ is

$$\delta(q_1, n) = \Big( \bigwedge_{a \in n} (0, q_a) \wedge \bigwedge_{a \notin n} (0, \overline{q_a}) \wedge \bigwedge_{c \text{ constraint}} (0, q_c) \wedge \bigwedge_{1 \leq i \leq k} (i, q_1) \Big) \vee (n = \{dummy\}) \tag{3.6}$$

  where $a \in preds(P)$. In state $q_1$, the 2ATA needs to motivate the presence of every predicate $a$ in the label by means of the state $q_a$, i.e., there must be some rule in the program that forces $a$ to be there. On the other hand, if there is some predicate $a$ that is not in the label, $\overline{q_a}$ motivates this as well, i.e., there may be no rule that forces $a$ to be in the label. It checks in every node that the constraints $c$ are satisfied by entering the state $q_c$, and it does the same check for the entire tree by entering $q_1$ again for all its successors, unless the label is the dummy label in which case it does not perform any more checks.
- We define a function $free : preds(P) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ such that *free(q)* returns true if $q$ (or its inverse) is free. For unary predicates $a \in preds(P)$ and binary (possibly inverted) predicates $f \in preds(P)$, we have the transitions:

$$\delta(q_a, n) = a \in n \wedge \left( \mathit{free}(a) \vee \bigvee_{r:a(X)\leftarrow\beta} (0, q_r) \right) \tag{3.7}$$

and

$$\delta(q_f, n) = f \in n \wedge \left( \mathit{free}(f) \vee \bigvee_{r:f(X,Y)\leftarrow\beta} (0, q_r) \vee \bigvee_{r:f^{\mathrm{i}}(X,Y)\leftarrow\beta} (0, q_{r^{\mathrm{i}}}) \right) . \tag{3.8}$$

The transitions $q_a$ and $q_f$ need to motivate the presence of $a$ and $f$ in the label. They each start by checking that $a$ and $f$ respectively are indeed in the label. If $a$ (or $f$) is free, the presence of $a$ (or $f$) is vacuously motivated. Otherwise, there has to be some rule $r$ with $a$ (respectively $f$) in the head such that the body of the rule can be made true; the latter happens by entering the state $q_r$. For binary predicates $f$, we have that $f$ may also be introduced by rules with $f^{\mathrm{i}}$ in the head, hence the presence of $q_{r^{\mathrm{i}}}$.

- Consider a unary rule

$$r : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), \psi \ .$$

A multi-set $I = \{i_{Y_i} \mid Y_i \in \mathrm{body}(r), i_{Y_i} \in \{0, \ldots, k\}\}$ satisfies $\psi$ if the following holds:

$$\forall i_{Y_i}, j_{Y_j} \in I \cdot Y_i \neq Y_j \in \psi \Rightarrow i_{Y_i} \neq j_{Y_j} \ .$$

Intuitively, such a multi-set $I$ indicates the allowed directions of the automaton making sure that none of the inequalities in $\psi$ are violated: if $Y_i \neq Y_j$ then the direction $i_{Y_i}$ cannot be equal to $j_{Y_j}$. The transition for $r$ is then

$$\delta(q_r, n) = (0, q_\beta) \wedge \exists I \text{ satisfies } \psi \cdot \left( \bigwedge_{m_{Y_m} \in I} (m_{Y_m}, q'_{\gamma_m}) \wedge (m'_{Y_m}, q_{\delta_m}) \right) , \tag{3.9}$$

with

$$q'_{\gamma_m} = \begin{cases} q_{\gamma_m{}^{\mathrm{i}}} & \text{if } m_{Y_m} = 0 \\ q_{\gamma_m} & \text{else} \end{cases}$$

and

$$m'_{Y_m} = \begin{cases} -1 & \text{if } m_{Y_m} = 0 \\ m_{Y_m} & \text{else} \ . \end{cases}$$

Intuitively, when reading a label with $a$ at node $X$, one has to verify that $\beta$ holds at the current node $X$ (hence the 0-direction). One also has to pick a multi-set $I$ corresponding to a set of directions that does not violate $\psi$ and check $\gamma_m$ and $\delta_m$. If a direction $m_{Y_m}$ is such that $0 < m_{Y_m}$, i.e., down

the tree, then one has to check $\gamma_m$ in the label of the successor $m_{Y_m}$. E.g., if $f(X, Y_m) \in \gamma_m(X, Y_m)$ and $m_{Y_m} = 2$, the 2ATA moves to the second successor $X \cdot 2$ of $X$ and checks whether $f$ is in the label of $X \cdot 2$ (recall that a $f$ in a label of $z \cdot i$ indicates a connection $f(z, z \cdot i)$).
If $m_{Y_m} = 0$, we assume the $Y_m$ is the predecessor of $X$ and we check that ${\gamma_m}^{\mathsf{i}}$ holds at $X$ itself and we go one node up (direction $-1$) to check $\delta_m$. E.g., assume $f(X, Y_m) \in \gamma_m(X, Y_m)$ and $b(Y_m) \in \delta_m$, with $m_{Y_m} = 0$. Then, we check that $f^{\mathsf{i}}$ is in the label of $X$ and $b$ is in the label of the predecessor $Y_m$ of $X$ (recall that a $f^{\mathsf{i}}$ in a label of $z$ indicates a connection $f^{\mathsf{i}}(z \cdot -1, z)$ or $f(z, z \cdot -1)$).

- The transition function for a binary rule

$$r : f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$$

comprises

$$\delta(q_r, n) = (-1, q_\beta) \wedge (0, q_\gamma) \wedge (0, q_\delta) \tag{3.10}$$

and

$$\delta(q_{r^{\mathsf{i}}}, n) = (-1, q_\delta) \wedge (0, q_{\gamma^{\mathsf{i}}}) \wedge (0, q_\beta)\ . \tag{3.11}$$

Intuitively, in the former transition, to motivate $f$ at node $Y$, we need to go up and check $\beta$ at the predecessor $X$, and $\gamma$ and $\delta$ at the current node. The latter transition follows from the equivalence of $f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$ and $f^{\mathsf{i}}(Y, X) \leftarrow \beta(X), \gamma^{\mathsf{i}}(Y, X), \delta(Y)$.

- For a set $\gamma \subseteq \mathit{preds}(P)$ and a $q_\gamma$ as introduced in one of the previous steps ($\gamma$ contains possibly inverted predicates), we have the transition

$$\delta(q_\gamma, n) = \bigwedge_{a \in \gamma} (0, q_a) \wedge \bigwedge_{not\ a \in \gamma} a \notin n \tag{3.12}$$

where $a$ is unary or (possibly inverted) binary. Intuitively, motivating positive predicates amounts to recursively motivating each positive predicate. The negative predicates can be directly checked in the node label: this corresponds to the GL-reduct strategy where naf-literals are removed according to their trueness w.r.t. some open interpretation.

- This concludes the definition of the transition function for *positive* states, i.e., states that motivate the presence of predicates in a label. Next, we define the states $\overline{q_a}$ that motivate the lack of a predicate in a label. Intuitively, there can be no applicable rule with $a$ in the head. The transition function for $\overline{q_a}$ is then basically the *De Morgan* rules applied to the transitions for $q_a$.
For unary predicates $a \in \mathit{preds}(P)$ and binary (possibly inverted) predicates $f \in \mathit{preds}(P)$, we have the transitions:

$$\delta(\overline{q_a}, n) = a \notin n \wedge \left( \neg \mathit{free}(a) \wedge \bigwedge_{r : a(X) \leftarrow \beta} (0, \overline{q_r}) \right) \tag{3.13}$$

and

$$\delta(\overline{q_f}, n) = f \notin n \wedge \left( \neg \mathit{free(f)} \wedge \bigwedge_{r:f(X,Y)\leftarrow\beta} (0, \overline{q_r}) \wedge \bigwedge_{r:f\dot{\mathrm{i}}(X,Y)\leftarrow\beta} (0, \overline{q_{r\dot{\mathrm{i}}}}) \right) \tag{3.14}$$

- For a unary rule

$$r : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), \psi$$

we have the transition

$$\delta(\overline{q_r}, n) = (0, \overline{q_\beta}) \vee \forall I \text{ satisfies } \psi \cdot \left( \bigvee_{m_{Y_m} \in I} (m_{Y_m}, \overline{q_{\gamma_m}}') \vee (m'_{Y_m}, \overline{q_{\delta_m}}) \right) \tag{3.15}$$

with

$$\overline{q_{\gamma_m}}' = \begin{cases} \overline{q_{\gamma_m \dot{\mathrm{i}}}} & \text{if } m_{Y_m} = 0 \\ \overline{q_{\gamma_m}} & \text{else} \end{cases}$$

and

$$m'_{Y_m} = \begin{cases} -1 & \text{if } m_{Y_m} = 0 \\ m_{Y_m} & \text{else } . \end{cases}$$

- The transition function for a binary rule

$$r : f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$$

comprises

$$\delta(\overline{q_r}, n) = (-1, \overline{q_\beta}) \vee (0, \overline{q_\gamma}) \vee (0, \overline{q_\delta}) \tag{3.16}$$

and

$$\delta(\overline{q_{r\dot{\mathrm{i}}}}, n) = (-1, \overline{q_\delta}) \vee (0, \overline{q_{\gamma\dot{\mathrm{i}}}}) \vee (0, \overline{q_\beta}) \; . \tag{3.17}$$

- For a set $\gamma \subseteq \mathit{preds}(P)$ and a $\overline{q_\gamma}$ as introduced in one of the previous steps ($\gamma$ contains possibly inverted predicates), we have the transition

$$\delta(\overline{q_\gamma}, n) = \bigvee_{a\in\gamma} (0, \overline{q_a}) \vee \bigvee_{not\ a\in\gamma} a \in n \tag{3.18}$$

where $a$ is unary or (possibly inverted) binary.

- For constraints $c : \leftarrow a(X)$, we have

$$\delta(q_c, n) = a \notin n \; . \tag{3.19}$$

A constraint $c$ is thus satisfied if $a$ is not in the current label of the node.

- For constraints $c_1 : \leftarrow f(X, Y)$ and $c_2 : \leftarrow f^{\mathrm{i}}(X, Y)$, we have

$$\delta(q_{c_1}, n) = \delta(q_{c_2}, n) = f \notin n \wedge f^{\mathrm{i}} \notin n \tag{3.20}$$

  A constraint $c_i$ is thus satisfied if neither $f$ nor $f^{\mathrm{i}}$ is in the current label of the node.

Note that we do not need qualifiers in the transition function definitions (3.9) and (3.15); we can rewrite them as boolean formulas.

### The States $Q$

Take the states $Q$ as introduced above. Denote with $Q^+$ the set of all states $q_a$ for unary and (possibly) inverted predicates $a$.

### The Acceptance Condition $\Omega$

Take $\Omega = (Q^+, Q)$. Then, an infinite path $\pi$ is accepting if $\mathrm{In}(\pi) \cap Q \neq \emptyset$ and $\mathrm{In}(\pi) \cap Q^+ = \emptyset$. Since the former is trivially satisfied for all paths, the latter condition boils down to forbidding the infinite occurrence of positive states. Intuitively, positive states $q_a$ were used to motivate the presence of predicates in a label by checking that there was some rule with a body that again could be motivated by positive states. Since, by the minimality of open answer sets, this must eventually end we forbid the infinite occurrence of positive states. E.g., a rule $a(X) \leftarrow a(X)$, would amount to a path with $q_a$ appearing infinitely often, which we disallow in accordance with the open answer set semantics where the above rule has an empty open answer set only.

**Theorem 3.38.** *Let $P$ be a CoLP and $p \in upreds(P)$. $p$ is satisfiable under IWA w.r.t. $P$ iff $\mathcal{L}(A_{p,P}) \neq \emptyset$.*

*Proof.* For the "only if" direction, assume $p$ is satisfiable under IWA w.r.t. $P$, then, by Theorem 3.36, $p$ is tree satisfiable under IWA w.r.t. $P$. By Definition 3.33 (pp. 80), there exists an open answer set under IWA $(U, M)$ such that $U$ is a tree with branching at most $k$, with $k$ the rank of $P$, and there is a labeling function $t : U \to 2^{preds(P)}$ such that

- $p \in t(\varepsilon)$ and $t(\varepsilon)$ does not contain (possibly inverted) predicates, and
- $z \cdot i \in U$, $i > 0$, iff there is some $f(z, z \cdot i) \in M$ where $f$ is possibly inverted, and
- for $y \in U$, $q \in upreds(P)$, $f \in bpreds(P)$,
  - $q(y) \in M$ iff $q \in t(y)$, and
  - $f(x, y) \in M$ iff $y = x \cdot i \wedge f \in t(y)$ or $x = y \cdot i \wedge f^{\mathrm{i}} \in t(x)$, where $f$ is possibly an inverted predicate.

The tree $U$ may be finite, however, a 2ATA demands for an infinite tree input. We thus take the infinite complete $k$-ary tree $U'$ and define $t' : U' \to 2^{preds(P)} \cup \{\{dummy\}\}$ as follows:

- for $x \in U$, $t'(x) \equiv t(x)$,
- for $x \in U' \setminus U$, $t'(x) \equiv \{dummy\}$.

Intuitively, we fill up all the holes in the tree $t$ and subsequently make it infinite; those new nodes are all labeled with the dummy label. Clearly, this is a well-behaved tree.

We then check that $t'$ is accepted by $A_{p,P}$ such that $\mathcal{L}(A_{p,P}) \neq \emptyset$. We construct a run $r$ on $t'$ by starting with a root $\varepsilon$ with $r(\varepsilon) = (\varepsilon, q_0)$ and subsequently defining the successors.

- We define one successor 1 for $\varepsilon$ such that $r(1) = (\varepsilon, q_1)$. Since $p \in t'(\varepsilon)$, this is in accordance with transition (3.5).
- Next, inductively, for every node $x$ in the run with $r(x) = (y, q_1)$, we distinguish between two cases:
  - If $t'(y) = \{dummy\}$, we do not define any successors for $x$: all paths passing through $x$ thus end (and are accepting since they are finite).
  - Otherwise, we introduce $|preds(P)| + |\{ \text{ constraints } \}| + k$. Looking at transition (3.6), we introduce $|preds(P)|$ successors to accommodate for all $a \in t(y)$ and all $a \notin t(y)$, $|\{ \text{ constraints } \}|$ successors for all the constraints and finally $k$ successors to recursively enter state $q_1$.
    In correspondence with the above introduction of successors $xi$, we define $r(xi) = (y, q_a)$ for $a \in t(y)$, $r(xi) = (y, \overline{q_a})$ for $a \notin t(y)$, $r(xi) = (y, q_c)$ for a constraint $c$, $r(xi_j) = (yj, q_1)$ for $k$ successors $yj$ of $y$.
    - For $r(xi) = (y, q_a)$, we have that $a \in t(y)$, and thus $a(y) \in M$ and $a(y) \in T^{\mathrm{i}^n}$ for some finite $n$. We then have that $a$ is free or there is a rule $a(y) \leftarrow body^+ \in P_U^M$ with $body^+ \subseteq T^{\mathrm{i}^{n-1}}$ originating from a unary rule:

      $$r : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), \psi \ .$$

      In the former case, we are done (and the path through $xi$ is finite and thus accepting). In the latter case, we define a successor $xi1$ of $xi$ and define $r(xi1) = (y, q_r)$. This is in accordance with transition (3.7).
      We have $body^+ \subseteq T^{\mathrm{i}^{n-1}}$, assume the $Y_i$ are grounded with $y_i$. Every $y_i$ is a successor or the predecessor of $y$. If $y_i = y \cdot j$, $1 \leq j \leq k$, take $i_{Y_i} = j$, if $y_i = y \cdot -1$, take $i_{Y_i} = 0$. Take $I$ the multi-set of such constructed $i_{Y_i}$, then $I$ satisfies $\psi$. Introduce $2 \times |I|$ successors $xi1j$ of $xi1$ for which $r$ can be defined in accordance with transition (3.9) with $I$ as defined.
      We can then define successors in accordance with transition (3.12) and fall again in a case where nodes are labeled with states $q_a$ but this time the corresponding $a$-atom will be in a lower $T^{\mathrm{i}^{n-1}}$. The negative predicates are checked immediately to be true and do not introduce any successors. Thus, the subtree of the run with label $(y, q_a)$ is a finite one such that all paths through $xi$ are accepting.

- For $r(xi) = (y, \overline{q_a})$, we can use a similar reasoning but this time the generated subtree need not to be finite. However, no positive states $q_b$ appear in labels of the subtree such that every path trough $xi$ is accepting.
- For $r(xi) = (y, \overline{q_c})$ and a constraint $c$, the transition can be immediately verified to be true (without the introduction of new successors), the paths through such $xi$ are thus finite and accepting.
- For $r(xi_j) = (yj, q_1)$ for $k$ successors $yj$ of $y$, we can repeat the above construction and case analysis. Since $q_1$ may occur infinitely, we have that all paths in the constructed run are accepting, making the run itself accepting.

For the "if" direction, assume $t : T \to 2^{preds(P)} \cup \{\{dummy\}\}$ is an infinite labeled $k$-ary tree that is accepted by $A_{p,P}$. Denote the corresponding run with $r$. Define $(U, M)$ with $U \equiv \{x | x \in T, t(x) \neq \{dummy\}\}$ and

$$M \equiv \{q(x) \mid q \in t(x) \cap upreds(P)\} \cup \{f(x, x \cdot i), f^{\mathsf{i}}(x \cdot i, x) \mid f \in t(x \cdot i) \cap bpreds(P)\} .$$

We have that $(U, M)$ is an open interpretation under IWA w.r.t. $P$. Since $r(\varepsilon) = (\varepsilon, q_0)$ and by the definition of a run and transition (3.5) which says that $\delta(q_0, t(\varepsilon)) = p \in t(\varepsilon) \wedge (0, q_1)$, we have that $p \in t(\varepsilon)$. By the definition of $M$, we then have that $p(\varepsilon) \in M$. It remains to show that $(U, M)$ is an open answer set under IWA of $P$.

- $M$ is a model under IWA of $P_U^M$. We check satisfiability of the different types of rules in a CoLP.
  - A rule in $P_U^M$ that originates from a free rule in $P$ is always satisfied.
  - Take a unary rule $r : a(x) \leftarrow \beta^+(x), \gamma_m^+(x, y_m), \delta_m^+(y_m) \in P_U^M$ originating from $s : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), Y_i \neq Y_j \in P$ with $\beta^-(x) \cap M = \gamma_m^-(x, y_m) \cap M = \delta_m^-(y_m) \cap M = \emptyset$ and $y_i \neq y_j$ for $Y_i \neq Y_j$.
    Assume body$(r) \subseteq M$ and assume, by contradiction, $a(x) \notin M$. Then $a \notin t(x)$. Since $x \in U$, we have that $t(x) \neq \{dummy\}$, and thus there always is a node in the run with label $(x, q_1)$. Since $a \notin t(x)$, there is a node in the run with label $(x, \overline{q_a})$. By transition (3.13), we have that $(x, \overline{q_s})$ is in the label of some node in the run. According to transition (3.15) there are two possibilities:
    - $(x, \overline{q_\beta})$ is in the label of some node in the run. Then, either there is some $b \in \beta$ such that $(x, \overline{q_b})$ in the run and thus $b \notin t(x)$ and $b(x) \notin M$, or there is a *not* $b \in \beta$ such that $b \in t(x)$ and thus $b(x) \in M$. Both lead to a contradiction with body$(r) \subseteq M$ and $\beta^-(x) \cap M = \emptyset$.
    - For all $I$ that satisfy the inequalities $\psi$ in $s$, we have one of the following:

1. There is a $m_{Y_m} \in I$ such that $(x \cdot m_{Y_m}, \overline{q_{\gamma_m}}')$ is in the label of the run.
2. There is a $m_{Y_m} \in I$ such that $(x \cdot m'_{Y_m}, \overline{q_{\delta_m}})$ is in the label of the run.

Assume an $Y_i$ in $s$ is grounded with $y_i$ from $r$, and take $i_{Y_i} \equiv 0$ if $y_i$ is a predecessor of $x$ or $i_{Y_i} \equiv j$ if $y_i$ is a successor $x \cdot j$ of $x$.[14] An $I$ consisting of those $i_{Y_i}$ satisfies $\psi$ such that one of the above must hold. One can see that both cases lead to a contradiction (similar to the above).

– Binary rules and constraints can be treated similarly.

- $M$ is a minimal model under IWA of $P_U^M$. Assume this is not true, then there is a model $N \subset M$ of $P_U^M$, and there is some $a(x) \in M \setminus N$ or $f(x,y) \in M \setminus N$.
  – Assume $a(x) \in M \setminus N$. Then, $a \in t(x)$ such that $(x, q_a)$ is the label of some node in the accepting run. Since positive states $q_b$ may not appear infinitely, we have that the subtree at the node with label $(x, q_a)$ is finite (the negative states cannot appear in this subtree by definition of $\delta$). One can then show, by induction on the depth of this tree, that for a node with label $(z, q_b)$ in this subtree $b(z) \in N$ for a unary $b$ and $b(z \cdot -1, z) \in N$ for a binary $b$. Consequently, $a(x) \in N$, which is a contradiction.
  – Assume $f(x,y) \in M \setminus N$. This case can be done similarly.

□

The non-emptiness problem for 2ATAs can be decided in exponential time in the number of states (Theorem 2.17, pp. 44), such that, with Theorem 3.38, we have an exponential upper bound (in the size of the program) for satisfiability checking under IWA w.r.t. CoLPs as well.

**Theorem 3.39.** *Satisfiability checking under IWA w.r.t. CoLPs is decidable and in* EXPTIME.

*Proof.* With Theorem 3.38, we have that $p$ is satisfiable w.r.t. a CoLP $P$ iff $\mathcal{L}(A_{p,P}) \neq \emptyset$. The latter can be decided in time exponential in the size of the number of states of $A_{p,P}$. One can see that the number of states of $A_{p,P}$ is polynomial in the size of $P$ such that the result follows. □

In Chapter 6, we will establish an EXPTIME lower bound for satisfiability checking under IWA w.r.t. CoLPs, by reducing satisfiability checking in the DL $\mathcal{SHIQ}$ to satisfiability checking under IWA w.r.t. CoLPs. Satisfiability checking w.r.t. CoLPs is thus EXPTIME-complete, which makes it more efficient than normal (closed world) answer set programming for arbitrary programs, which is NEXPTIME-complete if the head contains at most one positive literal, see [DEGV01].

[14] $y_i$ is always either a successor or the predecessor of $x$.

**Theorem 3.40.** *Consistency checking under IWA w.r.t. CoLPs is decidable and in* EXPTIME.

*Proof.* We can reduce consistency checking under IWA to satisfiability checking under IWA by Theorem 3.22 (pp. 73). Since $P \cup \{p(X) \vee not\ p(X) \leftarrow\}$ is a CoLP for a CoLP $P$, the result then follows from Theorem 3.39. □

**Theorem 3.41.** *Satisfiability checking and consistency checking w.r.t. CoLPs without inverted predicates is decidable and in* EXPTIME.

*Proof.* By Theorems 3.26 and 3.28, satisfiability checking and consistency checking coincides with their versions under IWA. Theorems 3.39 and 3.40 yield the desired result. □

A final note regarding the formal properties of CoLPs is that the syntax of CoLPs can be loosened up without loss of generality. Consider, for example, the following rule, expressing that a top film is a film that did well at the box office and received a good review of an expert magazine.

$$\begin{aligned} topFilm(Film) \leftarrow\ & film(Film), boxOffice(Film, Number), high(Number), \\ & goodReview(Film, Reviewer), \\ & writes(Reviewer, Magazine), expert(Magazine) \end{aligned}$$

In Figure 3.4, one sees that this rule has a tree structure if one maps variables to nodes in the tree. It is easy to rewrite such a tree rule to a pair of equivalent

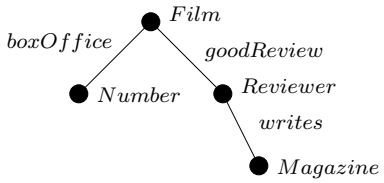


**Fig. 3.4.** Tree Rule

valid unary CoLP rules

$$\begin{aligned} topFilm(Film) \leftarrow\ & film(Film), boxOffice(Film, Number), high(Number), \\ & goodReview(Film, Reviewer), tmp(Reviewer) \end{aligned}$$

and

$$tmp(Reviewer) \leftarrow writes(Reviewer, Magazine), expert(Magazine)\ .$$

Intuitively, we recursively replace that part of the rule that goes *deeper* than one tree level, yielding CoLP rules in the end. Vice versa, such tree rules can be

seen as CoLP rules where the body is unfolded. In the following, we usually assume CoLP rules may have a tree structure if they can be equivalently rewritten as a set of CoLP rules in the sense of Definition 3.34. Consequently, we also allow for constraints $\leftarrow \beta$ where $\beta$ is a body as in a unary or binary CoLP rule. Such general constraints can be easily rewritten as the CoLP rules:

$$\begin{aligned} a(X) &\leftarrow \beta \\ &\leftarrow a(X) \end{aligned}$$

in the unary case, or as

$$\begin{aligned} f(X, Y) &\leftarrow \beta \\ &\leftarrow f(X, Y) \end{aligned}$$

in the binary case.

## 3.5 Application: Conceptual Modeling

Conceptual logic programming can be used as a language for conceptual modeling, hence its naming. We illustrate the translation of a particular *object-role modeling (ORM)* [Hal01] model to a CoLP.[15] The translated CoLPs can be used to detect and signal inconsistencies in the conceptual model, thus supporting a continuous quality assessment during the conceptual design phase. Advantages of using CoLP for conceptual modeling include modularity: rules can be added independently, e.g., to express complex constraints, while the consistency of the updated schema can be verified automatically.

*Object-role modeling* (ORM) is a mature conceptual modeling approach, comparable to *Entity Relationship Modeling (EER)* in its use. Its advantages include a rigorous methodology for building conceptual models, an easy to understand graphical notation, and a translation from conceptual ORM models to relational database models. Conceptual ORM models consist of object types and the roles they play, with in addition several constraints, such as mandatory or uniqueness constraints, enabling engineers to express a wide variety of knowledge. Instead of explaining ORM in its full detail, we highlight some basic features[16] of the graphical notation with the example in Figure 3.5. The boxes indicate the roles object types play. For instance, publications might have some co-author, and accordingly an author might be the co-author of a publication, as stated by "has co-author/is co-author of" below the corresponding role boxes. *Uniqueness constraints* are added as arrows

[15] Note that we do not claim a complete translation of ORM constructs to CoLP; we only use ORM, a standard modeling approach, to provide anecdotal evidence of the usefulness of CoLPs for conceptual modeling.

[16] Note that we do not consider so-called lexical object types and associated reference schemes; furthermore, we restrict ourselves to binary roles as roles of larger arity cannot be captured withing the CoLP framework.

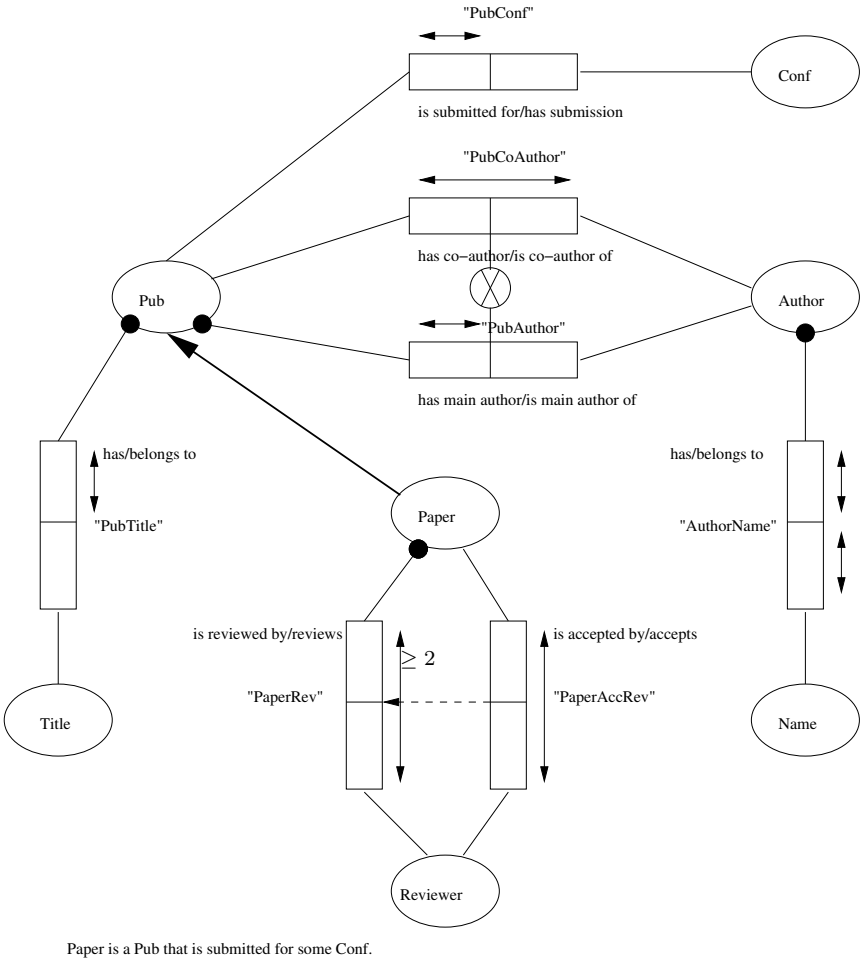


**Fig. 3.5.** ORM Example

over role boxes, e.g., there is at most one main author for each publication, but there may be more than one co-author. *Mandatory constraints* are indicated by big black dots, such that every publication must have a main author, while it does not need to have a co-author. Additionally, a *Paper* is a subtype of *Pub(lication)*, defined by "a *Paper* is a *Pub* that is submitted for some *Conf(erence)*". An *exclusion constraint*, a circle with a cross, indicates that no main author of a publication is also a co-author of that publication and vice versa. We have named not only roles but also the relationships they correspond to, *"PubConf"* is the relationship between *Pub* and *Conf* with the associated roles *is submitted for* and *has submission*. The example also shows a *subset constraint*, with a dashed arrow, saying that every paper accepted by

a reviewer must also have been reviewed by that same reviewer. Finally, we also have an *occurrence frequency* $\geq 2$ over the role *is reviewed by* indicating, in combination with the mandatory constraint, that every paper is reviewed by at least two reviewers.

To improve succinctness of the CoLP translation, we introduce abbreviations for certain CoLP rules that can be used to represent commonly occurring constructions.

| Rule type | Abbreviation |
|---|---|
| $a(X) \vee not\ a(X) \leftarrow$<br>$a(X,Y) \vee not\ a(X,Y) \leftarrow$ | **ftype** $a$<br>**frel** $a$ |
| **frel** $r$<br>$r_1(X) \leftarrow r(X,Y)$<br>$r_2(X) \leftarrow r^{\mathsf{i}}(X,Y)$<br>$\leftarrow r_1(X), not\ a_1(X)$<br>$\leftarrow r_2(X), not\ a_2(X)$ | **rel** $r(a_1\ r_1, a_2\ r_2)$ |
| $\leftarrow a(X), not\ b_1(X)$<br>$\ldots$<br>$\leftarrow a(X), not\ b_n(X)$ | **mandatory** $a(X)$ :<br>$b_1(X), \ldots, b_n(X)$ |
| $\leftarrow a(X,Y), not\ b_1(X,Y)$<br>$\ldots$<br>$\leftarrow a(X,Y), not\ b_n(X,Y)$ | **mandatory** $a(X, Y)$ :<br>$b_1(X,Y), \ldots, b_n(X,Y)$ |
| $\leftarrow a(X), not\ b_1(X), \ldots, not\ b_n(X)$ | **mandatory** $a(X)$ :<br>$b_1(X)$ or $\ldots$ or $b_n(X)$ |
| $\leftarrow a(X,Y),$<br>$\quad not\ b_1(X,Y), \ldots, not\ b_n(X,Y)$ | **mandatory** $a(X, Y)$ :<br>$b_1(X,Y)$ or $\ldots$ or $b_n(X,Y)$ |
| $\leftarrow a(X,Y), b(X,Y)$ | **impossible** $a(X, Y)$ and $b(X, Y)$ |
| $\leftarrow a(X), b(X)$ | **impossible** $a(X)$ and $b(X)$ |
| $\leftarrow f(X,Y_1), f(X,Y_2), Y_1 \neq Y_2$ | **functional** $f$ |
| $b(X) \leftarrow f(X,Y_1), \ldots,$<br>$\quad f(X,Y_n), Y_1 \neq Y_2, \ldots$<br>with $\leftarrow f(X,Y), not\ b(X)$<br>with $\leftarrow f(X,Y), b(X)$ | <br><br><br>**at-least** $f(X,n)$<br>**at-most** $f(X,(n-1))$ |

- **ftype** $a$ defines $a$ to be an object type, i.e., a unary predicate that may be populated (subject to further rules in the program).

- **frel** defines a relationship type; **rel** $r(a_1\ r_1, a_2\ r_2)$ indicates that $r$ is a relationship type with two associated roles $r_1$ and $r_2$ with respective domains $a_1$ and $a_2$.
- **mandatory** $a(X) : b_1(X), \ldots, b_n(X)$ can for example be used for an object type $a$ and roles $b_i$ to indicate that every object of type $a$ must play the roles $b_i$.
- **mandatory** $a(X) : b_1(X)$ or $\ldots$ or $b_n(X)$ is similar but now only one of the roles $b_i$ must be played.
- **impossible** provides a notational variant for constraints.
- **functional** $f$ asserts that a binary predicate $f$ is functional, i.e., if an $x$ plays the first role in $f$, then $x$ does not appear elsewhere playing the first role.
- The "**at-least** $f(X, n)$" and "**at-most** $f(X, n)$" constructions correspond to ORM occurrence frequencies "$\geq n$" and "$\leq n$" on the first role of $f$, and as such they mean that if there is an $f(x, y)$ then there are at least (resp. at most) $n$ $f(x, y_i)$ with different $y_i$ , i.e. if $x$ plays the first role in $f$, it plays it at least (resp. at most) $n$ times.

Using those abbreviations, the translation of the ORM model of Figure 3.5 to CoLP is straightforward. The result is shown in Table 3.2, where we renamed some of the role names, e.g., *has* is *hasTitle* or *hasName* depending on the accompanying object type (*Title* or *Name* respectively).

Taking a look at the rules of the CoLP in Table 3.2, one sees that in (1) we define the different object types in the ORM model. Secondly, we define the relationship types with their corresponding roles and associated object types. For example

$$\textbf{rel}\ \mathit{PubAuthor}(\mathit{Pub}\ \mathit{hasMain}, \mathit{Author}\ \mathit{isMain})$$

indicates that *PubAuthor* is a relationship with roles *hasMain* and *isMain* that are played respectively by *Pub* and *Author*. Specifying that object types (that are not related through subtyping) are mutually exclusive is done by constraints like in (3).

We then add the mandatory constraints in (4), and the implicit mandatory constraints as in (5), i.e., if an object type is attached to only one role it must play that role, and if an object type is attached to several roles but without an explicit dot indicating mandatoriness, objects of that type must play one of the attached roles (a disjunctive mandatory constraint). Next, we consider the uniqueness constraints, by declaring the appropriate relationships to be functional in (6), saying, for example, that an author has at most one name, and that no two authors have the same name. Together with the mandatory constraints this allows to identify authors with their name.

The exclusion constraint, the subset constraint, and subtyping can be expressed as in (7), (8), and (9), with the subtyping rules in (9) expressing that every paper is exactly a publication that is submitted for a conference. Finally,

**Table 3.2.** Translated CoLP from ORM Example

| | |
|---|---|
| **ftype** *Author*,**ftype** *Conf*,**ftype** *Paper*,**ftype** *Pub*,**ftype** *Name*,**ftype** *Title* | (1) |
| **rel** *PubConf*(*Pub isSubFor*, *Conf hasSub*) | (2) |
| **rel** *PubCoAuthor*(*Pub hasCo*, *Author isCo*) | |
| **rel** *PubAuthor*(*Pub hasMain*, *Author isMain*) | |
| **rel** *PubTitle*(*Pub hasTitle*, *Title TitleBelTo*) | |
| **rel** *PaperRev*(*Paper isRevBy*, *Reviewer Reviews*) | |
| **rel** *PaperAccRev*(*Paper isAccBy*, *Reviewer Accepts*) | |
| **rel** *AuthorName*(*Author hasName*, *Name NameBelTo*) | |
| **impossible** *Author*(*X*) and *Conf*(*X*), ... | (3) |
| **mandatory** *Pub*(*X*) : *hasMain*(*X*), *hasTitle*(*X*) | (4) |
| **mandatory** *Author*(*X*) : *hasName*(*X*) | |
| **mandatory** *Paper*(*X*) : *isRevBy*(*X*) | |
| **mandatory** *Title*(*X*) : *TitleBelTo*(*X*) | (5) |
| **mandatory** *Name*(*X*) : *NameBelTo*(*X*) | |
| **mandatory** *Reviewer*(*X*) : *Reviews*(*X*) or *Accepts*(*X*) | |
| **mandatory** *Conf*(*X*) : *hasSub*(*X*) | |
| **functional** *PubAuthor*, **functional** *PubTitle*, **functional** *PubConf*, **functional** *AuthorName*, **functional** *AuthorName*$^{\text{i}}$ | (6) |
| **impossible** *PubAuthor*(*X*, *Y*) and *PubCoAuthor*(*X*, *Y*) | (7) |
| **mandatory** *PaperAccRev*(*X*, *Y*) : *PaperRev*(*X*, *Y*) | (8) |
| **mandatory** *Paper*(*X*) : *isSubFor*(*X*) | (9) |
| **mandatory** *isSubFor*(*X*) : *Paper*(*X*) | |
| **at-least** *PaperRev*(*X*, *2*) | (10) |

the occurrence frequency, saying that a paper has at least two reviewers, can be written as in (10).

Creating an ORM model, one of the main questions that continually arises is "Can the model be populated?", or more specifically, whether it is possible to maintain information about authors that wrote a publication, or to keep track of publications submitted to a conference. For small conceptual models these may seem like trivial checks, however, when models become larger, a formalization of the ORM model and associated reasoning procedures becomes a necessity. For the example ORM model, Table 3.2 provides such a formalization.

Having a translation of an ORM model, one can use CoLP satisfiability checking to verify that the various object types can be populated, that other

derived properties do (not) hold etc. As an example consider Figure 3.6. The

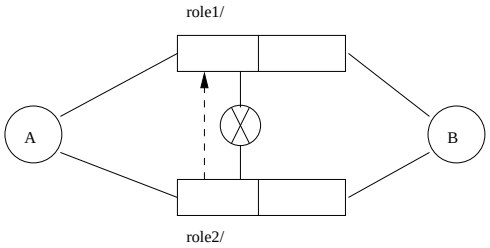


**Fig. 3.6.** No eXclusion with a Subset constraint

so-called Theorem NXS [Hal01] (No eXclusion with a Subset constraint) does not allow such constructions in a valid ORM model. The reason is that *role2* cannot be populated without the two constraints contradicting each other, i.e., on the one hand every object participating in *role2* must participate in *role1* by the subset constraint, but the exclusion constraint forbids exactly this.

If the modeler is not aware of Theorem NXS, or if the error is not as easy to spot, the CoLP translation can be used to detect the inconsistency. Indeed, the translated CoLP contains, among others, the two constraints

**mandatory** $role2(X) : role1(X)$ and **impossible** $role1(X)$ and $role2(X)$

and there exists no open answer set that contains a $role2(a)$, for an object $a$, or in other words, *role2* is not satisfiable w.r.t. the translated CoLP.

Note that consistency checking in CoLP, i.e., checking whether there exists an open answer set for the program, is less interesting in the context of conceptual modeling, where the main issue is whether roles can be populated or not. The latter corresponds to satisfiability checking of unary (role-)predicates in the CoLP framework.

Besides checking whether particular roles (or object types) can be populated, one can also query the conceptual model for the effect of possible future extensions. Assume the intention to add an object type "RevAuthor" corresponding to authors that are also reviewers. Before adding such object types, with all necessary constraints, one can simply add a rule $RevAuthor(X) \leftarrow Reviewer(X), Author(X)$ to the CoLP and check satisfiability of *RevAuthor* in order to see whether defining a reviewing author makes sense, i.e., whether, according to the current model, it is possible for someone to be both a reviewer and an author. CoLPs also allow for a modular extension of the conceptual model with additional constraints. For example, one may decide to include the business rule that every title of a publication should be unique. This is easily accomplished by making the inverted *PubTitle* functional by adding **functional** $PubTitle^{\mathrm{i}}$ to the CoLP.

## 3.6 Related Work

### 3.6.1 Domain Assumptions

The main difference between open answer set programming and normal answer set programming is the lack of a *domain closure axiom* in the former, i.e., our universe is an arbitrary non-empty countable superset of the *Herbrand Universe* (the set of constants in the program). Independently from the answer set semantics, definitions of universes different from Herbrand Universes have been investigated in literature. In [VS93], several axioms have been defined that constrain the allowed universes for a program. We repeat these axioms and explain how our concept of universe relates to them. The described axioms are defined for programs with *function symbols*. *Terms* in a program are then either constants, variables, or (recursively) of the form $f(t_1, \ldots, t_n)$ for terms $t_i$, $1 \leq i \leq n$, for an $n$-ary function symbol $f$. The other definitions (atoms, literals, ...) remain defined as before, with the modified definition of term.

In our overview of the axioms constraining the allowed universes of a program, we fix an example program with one constant $a$ and a unary function $g$. We then define a first-order formula as in [VS93]

$$\phi_H(x) \equiv (x = a) \vee \exists y \cdot (x = g(y) \wedge h(y)) \; . \tag{3.21}$$

The *domain closure axiom (dca)* is the second-order formula:

$$dca \equiv \forall R \cdot \phi_H[h/R] \subseteq R \Rightarrow \forall y \cdot R(y) \; . \tag{3.22}$$

This axiom enforces universes of the program to be a minimal closure of constants and function symbols in the program: if one has a set $R$ that is closed under constants and function symbol applications ($\phi_H[h/R] \subseteq R$) then every $y$ in the universe must be an element of that $R$. The universe is the minimal set that is closed under $\phi_H$, i.e., the ground terms that can be formed using the language of the program. Note that, if function symbols are present in the program, this always leads to an infinite universe. In the absence of function symbols, the domain closure axiom amounts to a universe that contains exactly the constants in the program. This differs from our definition of universe as we allow for *anonymous* elements, i.e., elements that are not constants.

One can augment a program with a rule [VRS91, VS93]

$$p_\#(g_\#(a_\#)) \leftarrow p_\#(g_\#(a_\#)) \tag{3.23}$$

where $p_\#$, $g_\#$, and $a_\#$ are new symbols not appearing in the original program. If $\phi_H$ is defined w.r.t. to the language of the *augmented program*, the domain closure axiom is denoted as dca#. Note that the added rule does not change the semantics, it only adds a predicate, function symbol, and constant to the language of the program in order to guarantee the presence of an infinite number of objects that are not named in the original program, and can thus

be considered anonymous. The difference with our definition of universe is that such an augmented rule always yields infinite universes, while in our case universes can be both finite and infinite. Similarly as the dca# is the assumption in [Kun87] that a countable infinite set of function symbols of each arity is present.

A *first-order approximation* of the domain closure axiom [VS93] is

$$dca_{fo} \equiv \forall x \cdot \phi_H[h/\mathbf{true}](x) \ . \tag{3.24}$$

Each element in the universe is thus either a constant or in the range of a function symbol. For the $\phi_H$ in Equation (3.21), we have that $\text{dca}_{fo}$ reduces to

$$(x = a) \vee \exists y \cdot (x = g(y)) \ .$$

$\text{dca}_{fo}$ thus allows for elements $g(x)$ where $x$ is a new anonymous element.

One can again define a variant $\text{dca}_{fo}\#$ by adding rule (3.23). In the absence of function symbols, $\text{dca}_{fo}$ enforces universes to be the set of constants in the program. The augmented variant again adds an infinite number of anonymous objects.

Finally, [VS93] introduces a *domain foundation axiom (dfa)*

$$dfa \equiv \forall R \cdot \phi_w[w/R] \subseteq R \Rightarrow \forall y \cdot R(y) \tag{3.25}$$

with

$$\phi_w(x) \equiv \forall y \cdot (x = g(y) \Rightarrow w(y)) \tag{3.26}$$

for the considered program. Intuitively, the universe may contain anything but the terms containing function symbols are finite. In the absence of function symbols, the universes that dfa enforces are exactly as our universes (in the assumption that constants are interpreted according to a unique name assumption, or, equivalently, as themselves).

### 3.6.2 *k*-Belief Sets

Specifically for the answer set semantics, [GP93] extends the language $\mathcal{L}_0$ of a program $P$ with an infinite sequence of new constants $c_1, \ldots, c_k, \ldots$ such that $\mathcal{L}_k$ is the expansion of $\mathcal{L}_0$ with $c_1, \ldots, c_k$. A pair $\langle k, B\rangle$ for a nonnegative integer $k$ and a set of ground literals $B$ in $\mathcal{L}_k$ is then a *$k$-belief set* of a program $P$ (without function symbols) iff $B$ is an answer set of $P_k$, where $P_k$ is the grounding of $P$ in the language $\mathcal{L}_k$. Our definition of open answer sets is more general in the sense that also infinite universes are allowed, while a $k$-belief set is always finite. Nonetheless, the other direction is valid: every $k$-belief set can be written as an open answer set.

**Theorem 3.42.** *Let $P$ be a program. Then, $\langle k, B\rangle$ is a $k$-belief set of $P$ iff $(cts(P) \cup \{c_1, \ldots, c_k\}, B)$ is an open answer set of $P$.*

*Proof.* $\langle k, B\rangle$ is a $k$-belief set of $P$ iff $B$ is an answer set of $P_{cts(P)\cup\{c_1,\ldots,c_k\}}$ iff $(cts(P) \cup \{c_1, \ldots, c_k\}, B)$ is an open answer set of $P$. □

Defining $k$-belief sets easily leads to undecidability as was argued for $k$-belief sets in [Sch93]. Interestingly, [Sch93] shows that reasoning becomes decidable again under the well-founded semantics . Since for stratified programs this semantics coincides with the answer set semantics, one has decidability of reasoning for $k$-belief sets of stratified programs. However, trying to extend the language of stratified programs with an extra stratum below all others, containing disjunctions of positive literals, leads to undecidability again [Sch93]. This construction, disjunctions with a stratified program on top, resembles the structure of CoLPs where we allow for special types of disjunctions (free rules) together with a set of rules that have a tree structure. However, in contrast with [Sch93], this gives decidable reasoning, and thus emphasizes the importance of the tree model property.

In [Sch95], an arbitrary infinite universe is assumed. Such answer set programming with an infinite universe can then define relations that are $\pi^1_1$-definable, i.e., relations that are definable by a formula $\forall\mathbf{P}\phi$ where $\phi$ is a first-order formula and $\mathbf{P}$ is a list of predicate variables, see, e.g., [EG97]. Every answer set w.r.t. such an infinite universe corresponds to an open answer set since we allow for both finite and infinite universes. The open answer set semantics thus generalizes both the approach in [GP93], where one extends the domain with a finite number of new constants, and the approach in [Sch95], where the domain is extended with an infinite number of new constants.

One may wonder whether one actually needs the capability of representing infinite domains; is a finite extension of the domain not enough? However, as illustrated in Example 3.7 (pp. 63), there are programs that have only infinite answer sets. Again the question remains: do we really need such infinity? Avoiding to answer this question, we "need" infinity in the sense that when one allows for certain constructs in a program, e.g., $\neq$, one can construct programs that have only infinite answer sets and, if one were to prohibit infinite extensions of the domain, one would get wrong answers regarding the satisfiability of predicates: under an open but finite answer set semantics the predicate *restore* in Example 3.7 is not satisfiable while it is satisfiable under the open and possibly infinite answer set semantics.

Similarly, the question rises whether one needs the finite extensions of the domain, is an infinite extension not enough? We claim one needs the possibility of finite and infinite answer sets. Take, for example, the program

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

If one would assume an infinite universe (and an infinite universe only), this program has no answer sets: there are an infinite number of constraints $\leftarrow \mathit{not}\ q(x)$ and no $q(x)$ can be deduced. On the other hand if we allow for arbitrary supersets of the constants in the program, we have that this program

has indeed an open answer set $(\{a\}, \{q(a)\})$. The latter is the desired behavior[17] as there are indeed cases (read *open answer sets*) where the expressed knowledge makes sense, i.e., in the absence of any anonymous elements.

### 3.6.3 Finitary Programs

Another approach to infinite reasoning is presented in [Bon04], where function symbols are included in the language. *Finitary programs* are identified as a class for which ground query answering is decidable, and lead to elegant formulations of, e.g., plans with unbounded planning length.

In [Bon04], program rules have the form $a \leftarrow \beta$ for an atom $a$ and $\beta$ a set of extended literals (not containing equality, inequality, or $\neg$) where function symbols are allowed. Call such programs *normal logic programs*. Terms are constants, variables, or of the form $f(t_1, \ldots, t_n)$ for $n$-ary function symbols $f$ where $t_1, \ldots, t_n$ in turn are *finite* terms. The *Herbrand Universe*, denoted $\mathcal{H}_P$, for a normal logic program is the set of ground terms that can be formed using the language of $P$; $\mathcal{B}_P$ is the set of ground atoms that can be formed using the language of $P$ (where the definition of terms takes into account function symbols). The grounding $gr(P)$ of $P$ is then w.r.t. the Herbrand Universe, i.e., $gr(P) = P_{\mathcal{H}_P}$.

An *atom dependency graph* of such a program is a directed graph having the atoms from $\mathcal{B}_P$ as nodes. There is a positive edge from a ground atom $a$ to a ground atom $b$ if there is a rule in $gr(P)$ with head $a$ and $b$ in the positive body. There is a negative edge from a ground atom $a$ to a ground atom $b$ if there is a rule in $gr(P)$ with head $a$ and $b$ in the negative body, i.e., *not* $b$ in the body. A ground atom $a$ *depends* on a ground atom $b$ if there is a path of (positive or negative) edges from $a$ to $b$ in the dependency graph. An *odd-cycle* in such a graph is a cycle with an odd number of negative edges. A ground atom is then *odd-cyclic* if it appears on an odd-cycle. A program is *finitely recursive* if each ground atom depends (negatively or positively) only on a finite number of ground atoms. A program is *finitary* if it is finitely recursive and there are only a finite number of odd-cycles in its dependency graph.

CoLPs and finitary programs are basically incomparable. Finitary programs that contain function symbols are not CoLPs since the language of the latter does not allow for function symbols, and vice versa, there are CoLPs that are not finitary.

*Example 3.43.* Take (part of) a CoLP $P$

$$a(X) \leftarrow f(X, Y), \mathit{not}\ b(Y)$$
$$b(X) \leftarrow a(X)$$

[17] Of course, this mostly depends on the situation at hand and is hard to defend formally, but in the context of conceptual reasoning we argue that this is indeed the desired behavior.

If we ground this program with an infinite universe[18], one gets an infinite number of ground rules

$$a(x) \leftarrow f(x,x), not\ b(x)$$
$$b(x) \leftarrow a(x)$$

and thus an infinite number of odd-cycles $a(x) \rightarrow b(x) \rightarrow a(x)$, where the first edge is a negative edge and the second edge is a positive one: the CoLP is not a finitary program.

A comparison between CoLPs and finitary programs is, as the previous example illustrates, quite artificial.

Concerning decidability, query answering w.r.t. a finitary program is decidable: queries can be answered by reasoning with a finite portion of $gr(P)$. We did not consider decidability of query answering for CoLPs since CoLPs do not contain constants such that it does not make sense to perform ground queries. However, unground query answering (i.e., satisfiability checking) w.r.t. finitary programs is only semi-decidable (and thus undecidable), where *semi-decidable* in this case means that one can countably enumerate all ground queries and then check whether this ground query holds. As was shown in Theorem 3.39, satisfiability checking w.r.t. CoLPs is decidable. For conceptual reasoning, where satisfiability checking of predicates is a key reasoning procedure, CoLPs seem to be advantageous over finitary programs, at least from a decidability viewpoint.

An additional difficulty for finitary programs is that checking whether a program is finitary is undecidable [Bon04], while checking whether a program is a CoLP is decidable; a CoLP is just a syntactically restricted finite set of rules, while the conditions for a finitary program impose restrictions on the infinite ground program.

### 3.6.4 Open Predicates

#### Open Logic Programming

*Open Logic Programming* as described in [VB97] is a framework that integrates classical first-order logic with logic programming. To this end, it allows for classical first-order reasoning for a designated set of predicates in the program, while retaining closed world reasoning for the other predicates. Intuitively, the semantics of a logic program with such a designated set of predicates, the *open predicates*, is given by taking a first-order interpretation for the open predicates, and, with the interpretation of those open predicates fixed, calculating the model of the logic program according to some pre-supposed

[18] One can also ground with a finite universe in the open answer set semantics, but in order to make a comparison between CoLPs and finitary programs, we ground with an infinite universe, as the Herbrand Universe is always infinite for finitary programs with function symbols.

logic programming semantics. Additionally, open logic programming allows for the specification of a set of first-order sentences that should be true in the obtained model.

**Definition 3.44 (cf. Definition 2.3.1 in [VB97]).** *Syntactically, an* open logic program *(OLP)* $T = \langle P, O, C \rangle$ *consists of*

- $P$*: A set of normal clauses (a normal logic program*[19]*).*
- $O$*: A set of open (abducible, undefined) predicates.*
- $C$*: A set of general first-order sentences.*

*Open predicates have no definition, i.e., they cannot occur in the head of any clauses.*

To make a comparison with our approach possible, we assume function symbols are not allowed. For *hierarchical* programs, i.e., programs where no predicate depends on itself in a predicate dependency graph[20], [VB97] gives the semantics by means of an extension of Clark's completion semantics [Cla87]. Instead of defining this formally, we take an example from [VB97] and explain the difference with Clark's traditional completion.

*Example 3.45 ([VB97]).* Take the open logic program $\langle P, O, C \rangle$, with $P$ the program

$$q(a, Y) \leftarrow p(Y)$$
$$r(X, Y) \leftarrow not\ q(X, Y)$$

and $O = \{p\}$, i.e., $p$ is an open predicate, and $C = \{\neg r(a, b)\}$. The semantics is given by the first-order theory

$$\forall X, Y \cdot q(X, Y) \iff p(Y) \wedge X = a$$
$$\forall X, Y \cdot r(X, Y) \iff \neg q(X, Y)$$
$$\neg r(a, b)$$
$$a \neq b$$

Intuitively, we provide rules that say exactly when something belongs to the extension of $q$ and $r$, we insert the theory $C$ such that models of the first two rules must also satisfy $C$, and, finally, we add $a \neq b$ to ensure that $a$ is interpreted differently from $b$ (first-order logic does not have the unique name assumption). The important part is that in Clark's completion semantics, we would also have a rule for $p$, i.e., in this case, $\forall X \cdot p(X) \iff$ **false**, since there are no rules with $p$ in the head. In an open setting, however, we do not include rules for the open predicates, such that we effectively get a first-order interpretation for them.

[19] One could allow for inequality or inequality, but for simplicity we do not.

[20] A *predicate dependency graph* is defined similarly as the atom dependency graph in Section 3.6.3, but with predicates instead of ground atoms, i.e., there is an edge from predicate $p$ to $q$ iff there is a rule with $p$ in the head and $q$ in the body.

Note that one can embed first-order logic in open logic programming. For an arbitrary first-order logic theory $T$, take the open logic program $\langle \emptyset, O, T \rangle$, with $O$ all the predicates in $T$, i.e., all predicates are open. As such, open logic programming is undecidable in the general case. Instead of the completion semantics, [VB97] also defines a version of the *well-founded semantics* [VRS91] for open logic programs with an associated sound (but incomplete in general) proof procedure SLDNFA (see also [DDS98]).

Adapting an answer set semantics for open logic programming, according to the intuition of open predicates in [VB97], can be done by using a translation to our open answer set semantics.

**Definition 3.46.** *Let $R = \langle P, O, C \rangle$ be an OLP. A pair $(U, M)$ is an* OLP answer set *of $R$ if*

- *$(U, M)$ is an open answer set of $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$, and*
- *$(U, M)$ is a first-order model of $C$.*

The open predicates give rise to free rules, indicating that one may choose their extension. The second condition ensures that every open answer set satisfies the sentences in $C$. We can then reuse our decidability results for the open answer set semantics.

**Theorem 3.47.** *Let $R = \langle P, O, \emptyset \rangle$ be an OLP with $P$ a CoLP without inverted predicates. Then, satisfiability checking and consistency checking w.r.t. $R$*[21] *is decidable in* EXPTIME.

*Proof.* This follows from Definition 3.46, the fact that $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ is a CoLP without inverted predicates for a CoLP $P$ without inverted predicates (since $P$ is part of an OLP), and Theorem 3.41 (pp. 95). □

### Finitary Open Logic Programs

Another approach to logic programming with open predicates can be found in [Bon03], which defines an *open program* as a tuple $\langle P, F, O \rangle$ where $P$ is a normal logic program, $F$ is a set of function symbols and constants not appearing in $P$, and $O$ is a set of predicate symbols, the open predicates. We define a variant of open programs where, instead of a set $F$ for each open program, we assume that there is a globally available infinite set of constants $Sk$ not appearing in $P$; an open program is then just a tuple $\langle P, O \rangle$ where $P$ is a normal logic program without function symbols. Intuitively, in the original definition $F$ is used to allow for the construction of an infinite set of anonymous elements, however, since we do not allow for function symbols and in order to

[21] Satisfiability checking and consistency checking for OLPs can be defined as one would except – replace "open answer set" by "OLP answer set".

enable a comparison with the open answer set semantics, we assume such an infinite set of anonymous elements is always given. A *completion* [Bon03] of an open program $\langle P, O\rangle$ is a normal logic program $P'$ (possibly infinite), such that

- $P' \supseteq P$,
- the constants of $P'$ occur in $P$ or in $Sk$,
- if $r \in P' \setminus P$, then the predicate in head$(r)$ is in $O$.

We call $M$ a $B$-answer set[22] of $\langle P, O\rangle$ if $M$ is an answer set of some completion of $\langle P, O\rangle$.

Intuitively, the choice of a $P'$ corresponds to the choice for a universe (one basically adds a subset of $Sk$ to the language of $P$) and a definition for the open predicates since the added rules have open predicates in their head.

**Theorem 3.48.** *Let $R = \langle P, O\rangle$ be an open program. Then, $M$ is a $B$-answer set of $R$ iff $(U, M)$ is an open answer set of $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ for some universe $U = cts(P) \cup X$ with $X \subseteq Sk$.*

*Proof.* Denote $\{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ with $Q$.

For the "only if" direction, assume $M$ is a $B$-answer set of $R$. Then there exists a completion $P'$ such that $M$ is an answer set of $P' = P \cup F$ with $F = P' \setminus P$. Define $U \equiv cts(P')$. Then $U = cts(P) \cup X$ with $X \subseteq Sk$ and $U$ is a universe for $P \cup Q$. We check that $(U, M)$ is an open answer set of $P \cup Q$.

- $M$ is a model of $(P \cup Q)^M_U$. We have that $M$ is a model of $P^M_U$ since $U = cts(P')$. The rules in $Q^M_U$ originate from free rules such that $M$ is a model of $Q^M_U$ too.
- $M$ is a minimal model of $(P \cup Q)^M_U$. Assume not, then there is a model $N \subset M$ of $(P \cup Q)^M_U$. We show that $N$ is a model of ${P'}^M_U$, which is a contradiction with the minimality of $M$. Indeed, $N$ is a model of $P^M_U$; it remains to show that $N$ is a model of $F^M_U$. Take $p(\mathbf{t}) \leftarrow \beta^+ \in F^M_U$ with $N \models \beta^+$. Then $M \models \beta^+$ such that $p(\mathbf{t}) \in M$, and $p(\mathbf{t}) \leftarrow\in Q^M_U$. Since $N$ is a model of the latter, we have that $p(\mathbf{t}) \in N$.

For the "if" direction, assume $(U, M)$ is an open answer set of $P \cup Q$ where $U = cts(P) \cup X$ with $X \subseteq Sk$. Then, $M$ is an answer set of $(P \cup Q)^M_U$. Define $P'$ as $P$ with the following rules added:

- $p(x, \ldots, x) \leftarrow u(x)$, for all $x \in U$, $p \in O$, and $u$ a new unary predicate not in $P$,
- $p(\mathbf{t}) \leftarrow$ for $p(\mathbf{t}) \in M$ and $p \in O$.

Intuitively, the first type of rules introduces the universe $U$ in $P'$ (the rules themselves are never applicable in $M$). The second type of rules corresponds to the rules in $Q^M_U$ (originating from free rules).

The program $P'$ is a completion of $\langle P, O\rangle$ such that it remains to check that $M$ is an answer set of $P'$. Note that $cts(P') = U$.

[22] $B$ for Bonatti.

- $M$ is a model of ${P'}_U^M$. We have that $M$ is a model of $P_U^M$. Take a $p(x,\ldots,x) \leftarrow u(x)$. Since $u(x) \notin M$, this rule is satisfied. Finally, for $p(\mathbf{t}) \leftarrow\in P$, we have, by definition of $P'$, that $p(\mathbf{t}) \in M$.
- $M$ is a minimal model of ${P'}_U^M$. Assume not, then there is a model $N \subset M$ of ${P'}_U^M$. We show that $N$ is a model of $(P \cup Q)_U^M$, which is a contradiction with the minimality of $M$. $N$ is a model of $P_U^M$. Take $p(\mathbf{t}) \leftarrow\in Q_U^M$. Then, $p(\mathbf{t}) \in M$ such that $p(\mathbf{t}) \leftarrow\in {P'}_U^M$ and $p(\mathbf{t}) \in N$.

□

One can reduce $B$-satisfiability checking w.r.t. open programs (i.e., with $B$-answer sets) to satisfiability checking in our setting.

**Theorem 3.49.** *Let $R = \langle P, O\rangle$ be an open program and $p$ a unary predicate in $R$. Then, $p$ is $B$-satisfiable w.r.t. $R$ iff $p$ is satisfiable w.r.t. $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow\ \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$.*

*Proof.* Denote $\{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow\ \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ with $Q$.

For the "only if" direction, assume $M$ is a $B$-answer set of $R$ such that there is some $p(x) \in M$. Then, by Theorem 3.48, there is an open answer set $(U, M)$ of $P \cup Q$ with $U = cts(P) \cup X$ and $X \subseteq Sk$, such that $p$ is satisfiable w.r.t. $P \cup Q$.

For the "if" direction, assume $(U, M)$ is an open answer set of $P \cup Q$ such that $p(x) \in M$. We cannot immediately apply Theorem 3.48 since we need the additional condition that $U = cts(P) \cup X$ with $X \subseteq Sk$. One can obtain this by mapping the elements of $U \setminus cts(P)$ to elements of $Sk$; we name the resulting universe $U'$. Since $U$ is a countable superset of $cts(P)$ and $Sk$ is a countable set, this can be easily done, and $(U', M)$ is an open answer set of $P \cup Q$. With Theorem 3.48 the result follows. □

A similar result holds for consistency checking.

**Theorem 3.50.** *Let $R = \langle P, O\rangle$ be an open program. $R$ is $B$-consistent*[23] *iff $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow\ \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ is consistent.*

*Proof.* Similar as the proof of Theorem 3.49. □

From our decidability results for CoLPs, we can then deduce some decidability results for open programs.

**Theorem 3.51.** *Let $R = \langle P, O\rangle$ be an open program with $P$ a CoLP without inverted predicates. Then, $B$-satisfiability checking and $B$-consistency checking w.r.t. $R$ is decidable in* EXPTIME.

*Proof.* This follows from Theorems 3.49 and 3.50, the fact that the program $P \cup \{p(\mathbf{X}) \vee not\ p(\mathbf{X}) \leftarrow\ \mid p \in O, p$ and $\mathbf{X}$ $n$-ary$\}$ is a CoLP without inverted predicates for a CoLP $P$ without inverted predicates (since $P$ is part of an open program), and Theorem 3.41. □

[23] An open program is $B$-consistent if there exists a $B$-answer set of $R$.

In [Bon03], a special class of open programs was identified. *Finitary open programs* extend the concept of *finitary*, as described in Section 3.6.3, for open programs. Keeping in mind that we consider not the original open programs with function symbols but with an infinite set of extra constants $Sk$, finitary open programs are open programs $\langle P, O\rangle$ where the ground $P_{Sk}$ is finitary. The same remarks apply then as in Section 3.6.3: detecting whether an open program is finitary is undecidable in general, ground query answering w.r.t. a finitary open program is decidable, but $B$-satisfiability checking is only semi-decidable.

### 3.6.5 ASP-EX

In [CI05], logic programs are extended with external predicates, capable of querying external sources of computation. The resulting framework is called ASP-EX. Since such external predicates may have infinite extensions, e.g., there are an infinite number of pairs $(x, y)$ such that $y$ is the square of $x$, [CI05] uses the open answer set semantics as their base semantics.

In the absence of external predicates, the open semantics in [CI05] coincides with our open answer set semantics. While we have decidability by imposing a tree structure on the rules, [CI05] shows decidability by imposing *safeness* of rules. A rule is *safe* if every variable in the rule appears in an atom in the positive body. Decidability is then guaranteed by the property that answer sets with as universe the constants of the program, i.e., the normal answers sets, coincide with the answer sets with as universe a superset of the program. Safeness thus ensures decidability as the open answer set semantics coincides with the normal answer set semantics for safe programs, however, the need for an open answer set semantics in safe programs (without external predicates though) is questionable.

Note that CoLPs are not safe in general, e.g., free rules are not safe since the body is empty while the head contains variables. Similarly, unary rules in a CoLP do not have to be safe either: $a(X) \leftarrow \mathit{not}\ b(X)$ has an empty positive body. Binary rules in a CoLP are safe since every positive body contains some $f(X, Y)$ for variables $X$ and $Y$ in the head.

However, despite the limited usefulness of an open domain semantics for safe programs without external predicates, in the presence of external predicates [CI05] detects interesting conditions on variables used in such external predicates for decidable reasoning, e.g., *weakly safe* programs.

### 3.6.6 $\omega$-Restricted Logic Programs

Another class of logic programs with function symbols are the *$\omega$-restricted programs* from [Syr01]. The Herbrand Universe of $\omega$-restricted programs is possibly infinite (in the presence of function symbols), however, answer sets are guaranteed to be finite, exactly by the structure of $\omega$-restricted programs. Informally, an $\omega$-restricted program consists of a stratified part and a part

that cannot be stratified (the *ω-stratum*). Rules are such that every variable in a rule is "guarded" by an atom of which the predicate is defined in a lower stratum. The answer sets of ω-restricted programs can then be computed by instantiating the strata from the bottom up.

For the predicate dependency graph of a normal logic program, we say that a path from predicate $p_1$ to a predicate $p_2$ is *negative* if there is a negative edge in it; otherwise, the path is *positive*. An *ω-stratification* of a program $P$ is then a function $\mathcal{S} : \mathit{preds}(P) \to \mathbb{N} \cup \{\omega\}$ such that, if there is a positive path from $p_1$ to $p_2$ in the predicate dependency graph of $P$, then $\mathcal{S}(p_1) \geq \mathcal{S}(p_2)$. In case of a negative path, we must have that $\mathcal{S}(p_1) > \mathcal{S}(p_2)$ or $\mathcal{S}(p_1) = \omega$. It is assumed that $n < \omega$ for all $n \in \mathbb{N}$. The *ω-valuation* of a rule $r : a \leftarrow \beta$ under an ω-stratification $\mathcal{S}$ is the function $\Omega(r, \mathcal{S}) = \mathcal{S}(\mathit{preds}(a))$[24]. The ω-valuation of a variable $X$ in a rule $r : a \leftarrow \beta$ under $\mathcal{S}$ is $\Omega(X, r, \mathcal{S}) = \min(\{\mathcal{S}(\mathit{preds}(l)) \mid l \in \beta^+ \wedge X \in \mathit{vars}(l)\} \cup \{\omega\})$. Finally, a rule is *ω-restricted* w.r.t. a ω-stratification $\mathcal{S}$ of $P$ iff $\forall X \in \mathit{vars}(r) \cdot \Omega(X, r, \mathcal{S}) < \Omega(r, \mathcal{S})$; a normal logic program is ω-restricted if its rules are ω-restricted w.r.t. an ω-stratification.

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

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

**Theorem 3.52.** *Let $P$ be an ω-restricted program and $U$ a universe for $P$. $(U, M)$ is an open answer set of $P$ iff $M$ is an answer set of $P$.*

*Proof.* We show this in 3 steps:

1. An answer set $M$ of $P_U$ does not contain atoms with anonymous elements (and thus $M$ is an interpretation of $P_{\mathcal{H}_P}$).
2. Rules in $P_U$ that contain anonymous elements are never applicable w.r.t. an interpretation $M$ of $P_{\mathcal{H}_P}$.
3. $M$ is an answer set of $P_U$ iff $M$ is an answer set of $P_{\mathcal{H}_P}$.

1. Assume $M$ includes a $p(\mathbf{t})$ which contains some $x \in U \setminus \mathcal{H}_P$. Thus there is a $p(\mathbf{t}) \leftarrow \beta \in P_U$ with $\beta^+ \subseteq M$ and $\beta^- \cap M = \emptyset$. Furthermore, this rule originates from a rule $r$ with a variable $X$ in its head and, by the ω-restrictedness of $P$, $\Omega(X, r, \mathcal{S}) < \Omega(r, S) = \mathcal{S}(p)$ for the ω-stratification of $P$, thus there must be a $q(\mathbf{s}) \in \beta^+ \subseteq M$ where $\mathbf{s}$ contains an anonymous element $x$ and $\mathcal{S}(q) < \mathcal{S}(p)$.
   Either $\mathcal{S}(p) = \omega$ or there is a $n < \omega$ such that $\mathcal{S}(p) = n$. In the latter case, we deduce a contradiction by induction on $n$.

[24] $\mathit{preds}(a)$ is the underlying predicate of $a$

If $n = 1$, we immediately have a contradiction as there is no lower stratum. By induction, assume we can deduce a contradiction for $\mathcal{S}(q) \leq n-1$ and $q(\mathbf{s}) \in M$ where $\mathbf{s}$ contains some anonymous element. If $\mathcal{S}(p) = n$, then we deduced above that there is a $q$ with $\mathcal{S}(q) < \mathcal{S}(p)$, in other words, $\mathcal{S}(q) \leq n-1$, which allows to deduce a contradiction by the induction hypothesis.

For $\mathcal{S}(p) = \omega$, we find a $q(\mathbf{s}) \in \beta^+ \subseteq M$ where $\mathbf{s}$ contains an anonymous element and $\mathcal{S}(q) < \mathcal{S}(p)$ and thus there is a $k < \omega$ such that $\mathcal{S}(q) = k$ which leads to a contradiction as in the previous case.

2. Assume $p(\mathbf{t}) \leftarrow \beta$ is a rule in $P_U$, applicable w.r.t. $M$, i.e., $\beta^+ \subseteq M$ and $\beta^- \cap M = \emptyset$, and it contains anonymous elements. The rule originates from some rule $r : \mathit{head} \leftarrow \mathit{body}$ and thus contains some variable $X$ that is grounded with an anonymous element $x$. Since $\Omega(X, r, \mathcal{S}) < \Omega(r, \mathcal{S})$, there is an atom containing $X$ in $\mathit{body}^+$ such that there is a $q(\mathbf{s}) \in \beta^+$ with $x \in \mathbf{s}$. But since $\beta^+ \subseteq M$, $q(\mathbf{s}) \in M$, a contradiction with $M$ being an interpretation of $P_{\mathcal{H}_P}$.
3. For the "only if" direction, assume $M$ is an answer set of $P_U$.
   - $M$ is a model of $P^M_{\mathcal{H}_P}$. Immediate, by 1., and since $P^M_{\mathcal{H}_P} \subseteq P^M_U$.
   - $M$ is a minimal model of $P^M_{\mathcal{H}_P}$. Assume not, then $N \subset M$ is some model of $P^M_{\mathcal{H}_P}$. Take an arbitrary $a \leftarrow \beta^+ \in P^M_U$ with $\beta^+ \subseteq N$, then $a \leftarrow \beta$ is applicable w.r.t. $N$ in $P_U$, such that, by 2., $a \leftarrow \beta$ does not contain anonymous elements, and thus $a \leftarrow \beta \in P_{\mathcal{H}_P}$ and $a \leftarrow \beta^+ \in P^M_{\mathcal{H}_P}$. Since $N$ is a model of $P^M_{\mathcal{H}_P}$, we have that $a \in N$. Thus $N$ is a model of $P^M_U$, which contradicts with the minimality of $M$.

   For the "if" direction, assume $M$ is an answer set of $P^M_{\mathcal{H}_P}$.
   - $M$ is a model of $P^M_U$. Take an arbitrary $a \leftarrow \beta^+ \in P^M_U$ with $\beta^+ \subseteq M$, then $a \leftarrow \beta$ is applicable w.r.t. $M$ in $P_U$, such that, by 2., $a \leftarrow \beta$ does not contain anonymous elements, and thus $a \leftarrow \beta \in P_{\mathcal{H}_P}$ and $a \leftarrow \beta^+ \in P^M_{\mathcal{H}_P}$. Since $M$ is a model of $P^M_{\mathcal{H}_P}$, we then have that $a \in M$. Thus $M$ is a model of $P^M_U$.
   - $M$ is a minimal model of $P^M_U$. Assume not, then $N \subset M$ is some model of $P^M_U$. But then $N$ is a model of $P^M_{\mathcal{H}_P}$, since $P^M_{\mathcal{H}_P} \subseteq P^M_U$. □

Since checking whether there exists an answer set of an $\omega$-restricted program is in general 2-NEXPTIME-complete [Syr01], we have, with Theorem 3.52, 2-NEXPTIME-completeness for consistency checking under the open answer set semantics for $\omega$-restricted programs.

**Theorem 3.53.** *Consistency checking w.r.t. $\omega$-restricted programs is* 2-NEXPTIME-*complete.*

*Proof.* Immediately with Theorem 3.52. □

Furthermore, since reasoning with $\omega$-restricted programs is implemented in the SMODELS reasoner [Sim], Theorem 3.52 implies an implementation of the open answer set semantics for $\omega$-restricted programs as well.

# 4

# Bounded Finite Model Property in Open Answer Set Programming

In Section 4.1, we introduce the *forest model property* and define a syntactically restricted class of programs, *forest logic programs (FoLPs)*, satisfying this property. We show in Section 4.2 that a particular type of FoLPs, local FoLPs, has the *bounded finite model property*, which enables a reduction to finite ASP. A type that can be reduced to local FoLPs are the *acyclic FoLPs* from Section 4.3. Section 4.4 identifies an upper bound for the complexity of reasoning. Finally, in Section 4.5, we extend FoLPs with an arbitrary finite set of rules that can only be grounded with constants present in the program, resulting in EFoLPs, and we show that properties such as the forest model property and the bounded finite model property are valid for suitably restricted classes of EFoLPs.

## 4.1 Forest Model Property

As seen in the previous chapter, the so-called *tree model property* proves to be a critical factor in showing decidability of satisfiability checking. A generalization of this property is the *forest model property*: if there is an open answer set that makes a predicate satisfiable, then there is an open answer set that has the form of a set of trees, a forest. A similar property arises for DLs that include nominals, e.g., $\mathcal{SHOQ}(\mathbf{D})$ [HS01].

*Example 4.1.* Consider the program $P$ representing the knowledge that a company can be trusted for doing business with if it has the ISO 9000 quality certificate and at least two different trustworthy companies are doing business with it:

$$\begin{aligned} trust(C) \leftarrow\ & t\_bus(C, C_1), t\_bus(C, C_2), C_1 \neq C_2, qual(C, iso9000) \\ \leftarrow\ & t\_bus(C, D), not\ trust(D) \end{aligned}$$

with $t\_bus$ and $qual$ free predicates, and $iso9000$ a constant. An open answer set, e.g., $(U, M)$ with $U \equiv \{x_1, x_2, \ldots\}$ and

$$M \equiv \{trust(x_1), t\_bus(x_1, x_2), t\_bus(x_1, x_3), \\ qual(x_1, iso9000), trust(x_2), \ldots\}$$

is such that for every trusted company $x_i$ in $M$, i.e., $trust(x_i) \in M$, there must be $t\_bus(x_i, x_j)$, $t\_bus(x_i, x_k)$ and $trust(x_j)$, $trust(x_k)$ with $x_j \neq x_k$; additionally, every trusted company has the *iso9000* quality label. This particular open answer set has a forest shape, as can be seen from Fig. 4.1: we call it a *forest model*. The forest in Fig. 4.1 consists of two trees, one with root $x_1$ and

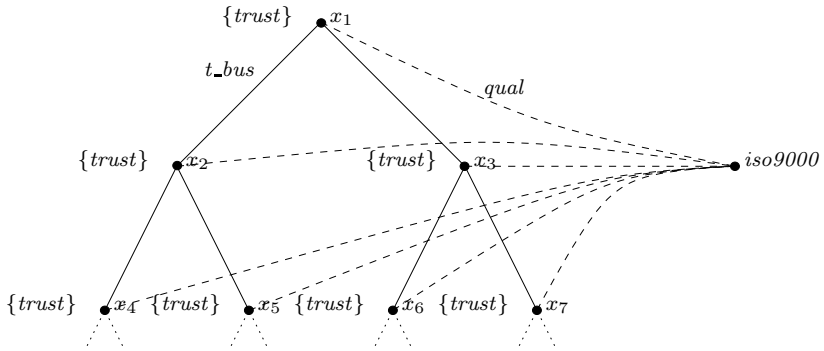


**Fig. 4.1.** Forest Model

one, a single node tree, with root *iso9000*. This will be a general feature of programs with the forest model property: they have open answer sets that can be rewritten as sets of trees where each constant is identified with the root of its own tree and there is (possibly) an additional tree with an anonymous root. The labels of a node $x$ in a tree, e.g., $\{trust\}$ for $x_2$, encode which literals are in the corresponding open answer set, e.g., $trust(x_2) \in M$. The labeled edges indicate relations between domain elements. The dashed arrows, describing relations between anonymous domain elements $x \in U \setminus cts(P)$ and constants, appear to be violating the forest structure; their labels can, however, be stored in the label of the starting node, e.g., $qual(x_2, iso9000)$ can be kept in the label of $x_2$ as $qual^{iso9000}$. Since there are only a finite number of constants, the number of different labels in a forest is still finite. To be formally correct, the forest should not have any labeled edges; we solve this by keeping the label on an edge from $x$ to $y$ in the label of $y$, and assume that binary predicates in labels refer to edge labels from the predecessor node to the current node, e.g., for $t\_bus(x_1, x_2)$ we keep $t\_bus$ in the label of $x_2$.

**Definition 4.2.** *Let $P$ be a program. A $p \in upreds(P)$ is* forest satisfiable *w.r.t. $P$ if there is an open answer set $(U, M)$ of $P$ and there is a forest $\mathsf{F} \equiv$*

$\{t_\varepsilon\} \cup \{t_a \mid a \in cts(P)\}$ *where the* $t_x : U_x \to 2^{preds(P) \cup \{f^a \mid a \in cts(P) \wedge f \in bpreds(P)\}}$ *are labeled trees*[1] *with bounded arity such that*

- $U = \cup_{t_x \in \mathsf{F}} U_x$, *and*[2]
- $p \in t_\varepsilon(\varepsilon)$, *where* $\varepsilon$ *is the root of* $U_\varepsilon$, *and*
- $z \cdot i \in U_x$, $i > 0$, *iff there is some* $f(z, z \cdot i) \in M$, $z \in U_x$, *and*
- *for* $y \in U_x$, $q \in upreds(P)$, $f \in bpreds(P)$, *we have that*
  - $q(y) \in M$ *iff* $q \in t_x(y)$, *and*
  - $f(y, u) \in M$ *iff* $(u = y \cdot i \in U_x \wedge f \in t_x(u)) \vee (u \in cts(P) \wedge f^u \in t_x(y))$.

*We call such a* $(U, M)$ *a* forest model *and a program* $P$ *has the* forest model property *if the following property holds:*

> *If* $p \in upreds(P)$ *is satisfiable w.r.t.* $P$ *then* $p$ *is forest satisfiable w.r.t.* $P$.

*The* label *of a node* $z \in U_x$ *is* $\mathcal{L}(z) \equiv \{q \mid q \in t_x(z), q \in upreds(P)\}$.[3]

This definition is very similar to the definition of tree satisfiability under IWA (Definition 3.33, pp. 80), except that it generalizes it to take into account forests instead of trees. This generalization allows for the introduction of constants in the CoLPs of the previous chapter. We do not take into account, however, the inverted predicates (i.e., we do not define *forest satisfiable under the IWA*) as we want to look for a fragment that has a *bounded finite model property* later on. In the presence of inverted predicates, one can show that there are infinity programs (Example 3.21, pp. 73).

*Example 4.3.* The forest model of Example 4.1, drawn according to Definition 4.2, is then as in Fig. 4.2.

In effect, a forest model can be seen as a collection of trees, with arbitrary connections from elements to constants. As a consequence, the connections between constants, i.e., the roots of the trees, may form an arbitrary graph.

A particular class of programs with this forest model property are *forest logic programs* (FoLPs).

**Definition 4.4.** *A* forest logic program (FoLP) *is a program with only unary and binary predicates, and such that a rule is of one of the following types,*

---

[1] We assume that the root of each $t_x$ is identified with a constant, unless $x = \varepsilon$. We further allow for $t_\varepsilon$ to be an element of $\{t_a \mid a \in cts(P)\}$, i.e., the forest contains trees for which the roots are identified with constants and possibly, but not necessarily, an extra tree with unidentified root node.

[2] Note that $U$ is thus a "flat" structure, consisting of the nodes in the trees $U_x$. Since the $U_x$'s are possibly non-disjoint, $U$ would thus be a multi-set. However, we assume that the elements in $U_x$ are – by convention – prefixed with $x$, effectively making them disjoint. E.g., for $U_\varepsilon = \{\varepsilon, 1\}$ and $U_a = \{a, a1, a2\}$, we have that $U = \{\varepsilon, 1, a, a1, a2\}$.

[3] $\mathcal{L}(z)$ is not equal to $t_x(z)$ as we only record the unary predicates in the former.

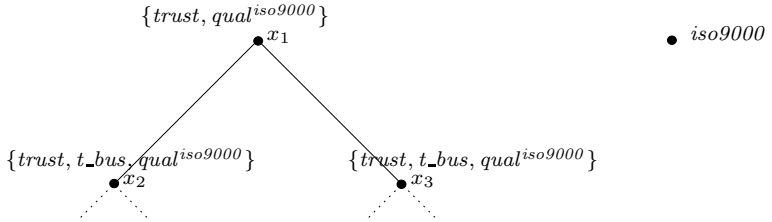


**Fig. 4.2.** Formal Forest Model

- free rules $a(s) \vee not\ a(s) \leftarrow$ *or* $f(s,t) \vee not\ f(s,t) \leftarrow$ *, where* $s$ *and* $t$ *are terms such that if* $s$ *and* $t$ *are both variables, they are different*[4]*,*
- unary rules

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

  *for terms* $s$ *and* $t_m$*,* $1 \leq m \leq k$ *(again, if both* $s$ *and* $t_m$ *are variables, they are different; similarly for* $t_i$ *and* $t_j$*), where*
  1. $\psi \subseteq \bigcup_{1 \leq i \neq j \leq k} \{t_i \neq t_j\}$ *and* $\{=, \neq\} \cap \gamma_m = \emptyset$ *for* $1 \leq m \leq k$*,*
  2. $\forall t_i \in vars(r) \cdot \gamma_i^+ \neq \emptyset$*, i.e., for variables* $t_i$ *there is a positive atom that connects* $s$ *and* $t_i$*,*
- binary rules $f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t)$ *with* $\{=, \neq\} \cap \gamma = \emptyset$ *and* $\gamma^+ \neq \emptyset$ *if* $t$ *is a variable (*$s$ *and* $t$ *are different if both are variables),*
- constraints $\leftarrow a(s)$ *or* $\leftarrow f(s,t)$*, (*$s$ *and* $t$ *are different if both are variables),*

The conditions in FoLPs are essentially the same as for CoLPs (Definition 3.34, pp. 80), i.e., allowing for constants in the program does not yield extra conditions on the rules. Indeed, the conditions

$$\forall t_i \in vars(r) \cdot \gamma_i^+ \neq \emptyset$$

for unary rules and

$$\gamma^+ \neq \emptyset \text{ if } t \text{ is a variable}$$

for binary rules apply only to variables.

Intuitively, the syntactical restrictions on the rules in a FoLP are now designed to ensure the forest model property, while maintaining a sufficient degree of expressiveness, e.g., to simulate expressive DLs, see Chapter 6. Recall (pp. 81) that a rule

$$q(X) \leftarrow not\ f(X, Y), not\ q(Y)$$

was not a valid CoLP rule (and hence it is not a valid FoLP rule), since it is impossible to make a tree out of an open answer set that satisfies $q$ (there

[4] A rule $f(X,X) \vee not\ f(X,X) \leftarrow$ is not allowed.

is no edge to connect the two different elements ). Hence, we enforce that, for variables, there is always a positive literal connecting such an $X$ and $Y$. However, if instead of $Y$ we have a constant $a$, this restriction is no longer necessary. Take

$$q(X) \leftarrow not\ f(X, a), not\ q(a)$$

We have that $(\{x, a\}, \{q(x)\})$ is an open answer set, which leads to a forest with trees $t_\varepsilon$, where $t_\varepsilon(\varepsilon) = \{q\}$, and $t_a$ where the root is identified with $a$. Thus, we no longer need a connection between $x$ and $a$ since $a$ is actually (identified with) the root of its own tree.

In the same spirit we have for binary rules that $f(X, Y) \leftarrow v(X)$ is not allowed, since this may impose connections between $x$ and $y$ without $y$ being a successor of $x$. $f(X, a) \leftarrow v(X)$ for a constant $a$ on the other hand is allowed, since this imposes connections between some $x$ and $a$ (which is again the root of its own tree). One can encode such connections in the label of $x$ with $f^a$ such that the forest structure is not broken.

Like for CoLPs, we can ease the syntactical restrictions on FoLPs by allowing for more general bodies, e.g., by unfolding them, resulting in bodies with a tree like structure. More complex constraints $\leftarrow \beta$ can be simulated by a unary rule $a(s) \leftarrow \beta$ and a constraint $\leftarrow a(s)$.

**Theorem 4.5.** *Let $P$ be a CoLP without inverted predicates. Then, $P$ is a FoLP.*

*Proof.* Immediately from Definitions 3.34 and 4.4. □

Since a FoLP may contain constants, but a CoLP may not, the other direction does not hold. However, FoLPs without constants are CoLPs.

**Theorem 4.6.** *Let $P$ be a FoLP without constants. Then, $P$ is a CoLP.*

*Proof.* Immediately from Definitions 3.34 and 4.4. □

Consequently, the set of CoLPs without inverted predicates coincides with the set of FoLPs without constants.

We modify the definitions of liveness as follows. A unary rule

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

in a FoLP is a *live* rule if there is a $\gamma_m \neq \emptyset$ and $t_m$ is a variable (the latter condition being extra compared to CoLPs). A unary predicate $a$ is live if there is a live rule $r$ with $a$ in head($r$) and $a$ is not free. We denote the set of live predicates for a FoLP $P$ again with $live(P)$. A *degree* for the liveliness of a FoLP rule $r$, i.e., how many new individuals might need to be introduced to make the head true, is

$$degree(r) \equiv |\{m \mid \gamma_m \neq \emptyset, t_m \in vars(r)\}| \,. \tag{4.1}$$

The *degree of a live predicate* $a$ in $P$ is

$$degree(a) \equiv \max\{degree(r) \mid a \in \text{head}(r)\} \ . \tag{4.2}$$

The *rank of a FoLP* $P$ is the sum of the degrees of the live predicates in $P$:

$$\sum_{a \in live(P)} degree(a) \ . \tag{4.3}$$

FoLPs indeed have the forest model property.

**Theorem 4.7.** *Forest logic programs have the forest model property.*

*Proof.* Take a FoLP $P$ and $p \in upreds(P)$ s.t. $p$ is satisfiable, i.e., there exists an open answer set $(U, M)$ with $p(u) \in M$. Let $n$ be the rank of $P$.

We first define mappings $\theta_x : \{x\} \cdot \{1, \ldots, n\}^* \to U$ from a complete $n$-ary tree to the domain $U$, with $x \in K$ where

$$K \equiv \begin{cases} cts(P) & \text{if } u \in cts(P) \\ \{\varepsilon\} \cup cts(P) & \text{otherwise} \end{cases}$$

Intuitively, we assume there are trees with the roots identified with the constants, and, in case $u$ is not a constant, there is an additional tree with anonymous root (not identified with a constant). Each $\theta_x$ then associates some of the nodes in the trees with elements in the domain.

Initially, assume each $\theta_x$ is undefined for the whole tree $\{x\} \cdot \{1, \ldots, n\}^*$. If $\theta_x$ is defined on some node, we will call the node *defined*. Each $\theta_x$ is constructed as follows:

- Define $\theta_x(x) \equiv x$ if $x \in cts(P)$ and $\theta_x(x) = u$ otherwise, i.e., if $x = \varepsilon$.
- Assume that we have considered, as in [Var98], every node in $\{x\} \cdot \{1, \ldots, n\}^k$, for some $k$, as well as every successor node of the defined $z' \in \text{fr}(\{x\} \cdot \{1, \ldots, n\}^k)$ until[5] $z \cdot m$ for some defined $z \in \text{fr}(\{x\} \cdot \{1, \ldots, n\}^k)$. Consequently, we have considered the nodes $z \cdot 1, \ldots, z \cdot m$.
  Since $\theta_x$ is defined on $z$, we have that $\theta_x(z) \in U$. For every $q(\theta_x(z)) \in M$, there is, by Theorem 3.13 (pp. 66), some $l < \infty$ s.t. $q(\theta_x(z)) \in T^l$. By definition of the immediate consequence operator, we have that there is a rule
  $$r_{q(\theta_x(z))} : q(\theta_x(z)) \leftarrow \beta^+[] \in P_U^M$$
  with $M \models \beta^+[]$, originating from $r : q(s) \vee \alpha \leftarrow \beta \in P$ such that
  - $M \models \alpha^-[]$,
  - $M \models not\ \beta^-[]$,

[5] By saying "until", we assume that there is an ordering from left to right in the graphical representation of the tree.

and $T^{l-1} \models \beta^+[]$. If $r$ is not live, we do nothing. Else, the body of $r_{q(\theta_x(z))}$ is of the form

$$\gamma^+(\theta_x(z)), \bigcup_i \gamma_i^+(\theta_x(z), y_i), \bigcup_i \delta_i^+(y_i)$$

with at least one $\gamma_i^+ \neq \emptyset$ for $y_i$ not a constant. Without loss of generality, we can assume that for all $i$, where $y_i$ is not a constant, $\gamma_i^+ \neq \emptyset$. For the $y_i$ that are not constants we then do the following: if there is a $z \cdot j \in \{z \cdot -1, z \cdot 1, \ldots, z \cdot m, \ldots, z \cdot (m+i-1))\}$ with $\theta(z \cdot j) = y_i$ then $\theta_x$ remains undefined on $z \cdot (m+i)$, otherwise $\theta(z \cdot (m+i)) \equiv y_i$. Intuitively, if $\theta$ is already defined on a neighbor of $z$ as equal to $y_i$, there is no need to define $\theta$ on another successor as equal to $y_i$.

For each $\theta_x$, define a corresponding labeled tree

$$t_x : \mathrm{dom}(\theta_x) \to 2^{preds(P) \cup \{f^a \mid a \in cts(P) \wedge f \in bpreds(P)\}} ,$$

where $\mathrm{dom}(\theta_x)$ are those elements for which $\theta_x$ is defined, by

- $t_x(x) \equiv \{q \mid q(\theta_x(x)) \in M\} \cup \{f^a \mid f(\theta_x(x), a) \in M, a \in cts(P)\}$,
- $t_x(z \cdot i) \equiv \{q \mid q(\theta_x(z \cdot i)) \in M\} \cup \{f \mid f(\theta_x(z), \theta_x(z \cdot i)) \in M\} \cup \{f^a \mid f(\theta_x(z \cdot i), a) \in M, a \in cts(P)\}$.

Define the open interpretation $(V, N)$ such that $V \equiv \cup_x \mathrm{dom}(\theta_x)$ and

$$\begin{aligned} N \equiv & \{q(z) \mid q \in t_x(z) \cap upreds(P), z \in \mathrm{dom}(\theta_x)\} \\ & \cup \{f(z, z \cdot i) \mid f \in t_x(z \cdot i) \cap bpreds(P), \{z, z \cdot i\} \subseteq \mathrm{dom}(\theta_x)\} \\ & \cup \{f(z, a) \mid f^a \in t_x(z), z \in \mathrm{dom}(\theta_x)\} . \end{aligned}$$

Similarly as in the proof of Theorem 3.36, one can then check that $(V, N)$ is indeed a forest model of $P$ according to Definition 4.2. □

## 4.2 Bounded Finite Model Property

Satisfiability checking w.r.t. the CoLPs in Chapter 3 was shown to be decidable by a reduction to two-way alternating tree automata. However, the definition of FoLPs includes constants, which are not allowed in CoLPs, such that the automata reduction cannot be readily applied. Moreover, while automata provide an elegant characterization, there are few implementations available, e.g., [HS03] implements a specific type, looping alternating automata, using a translation to description logics.

An alternative approach is to identify a particular class of FoLPs, *local FoLPs*, that allow for a reduction to normal (finite) answer set programming by a so-called *bounded finite model property*. This property enables the transformation of an (infinite) open answer set into a finite one, and, more specifically, it establishes a bound on the number of domain elements that are needed for such a construction.

Infinite forest models can be turned into finite structures as follows: cut every path in the forest from the moment there are duplicate labels and copy the connections of the first node in such a duplicate pair to the second node of the pair. Intuitively, when we reach a node that is in a state we already encountered, we proceed as that previous state, instead of going further down the tree. This *cutting* is similar to the blocking technique for DL tableaux [BCM+03], but the minimality of (open) answer sets makes it non-trivial in the sense that cutting yields finite structures that are not guaranteed to be (open) answer sets. We will identify a class of FoLPs, *local* FoLPs, for which the cutting of infinite forest models does result in finite open answer sets.

*Example 4.8.* Considering the forest model in Fig. 4.1, we can cut everything below $x_2$ and $x_3$ since they have the same label as $x_1$. Furthermore, since $t\_bus(x_1, x_2)$, $t\_bus(x_1, x_3)$, and $qual(x_1, iso9000)$, we have that $t\_bus(x_i, x_2)$, $t\_bus(x_i, x_3)$, and $qual(x_i, iso9000)$ for $i = 2$ and $i = 3$, resulting in the finite open answer set depicted in Fig. 4.3.

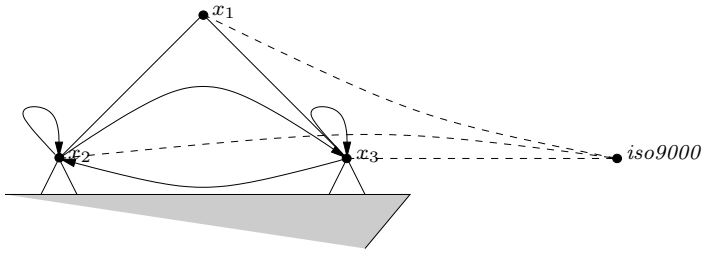


**Fig. 4.3.** Bounded Finite Model

**Definition 4.9.** *A program $P$ has the* bounded finite model property *if the following holds:*

> *If $p \in upreds(P)$ is satisfiable w.r.t. $P$ then there is a finite open answer set $(U, M)$ of $P$ and a nonnegative integer $k$, defined only in function of $P$, such that $p(x) \in M$ and $|U| < k$.*

The bounded finite model property is similar to the *small model property* found in the temporal logic CTL [Eme90] where a CTL formula is satisfiable iff it is satisfiable by a model that has a number of states at most exponential in the length of the formula.

Cutting the (infinite) forest at nodes with duplicate labels, as illustrated above, does not necessarily yield a finite answer set.

*Example 4.10.* Consider the program

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

A forest model of this program, depicted in Figure 4.4, is

$$\{a(\varepsilon), f(\varepsilon, 1), a(1), f(1, 11), a(11), b(11)\}\,.$$

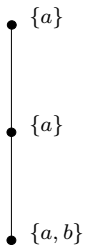


**Fig. 4.4.** Example Open Answer Set

Since $\varepsilon$ and 1 have the same label, i.e., $\mathcal{L}(\varepsilon) = \mathcal{L}(1)$, we cut the tree at 1 and copy the connections from $\varepsilon$ ($f(\varepsilon, 1)$) to 1 such that $f(1, 1)$ holds in the new structure; this is depicted in Figure 4.5.

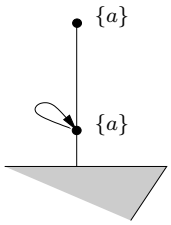


**Fig. 4.5.** Example Cutting

In the resulting structure $\{a(\varepsilon), f(\varepsilon, 1), a(1), f(1, 1)\}$, neither $a(\varepsilon)$ nor $a(1)$ is (minimally) motivated, as $b(11)$ is no longer present. The resulting structure is thus not minimal.

Intuitively, we want a FoLP where atoms in forest models are *locally* motivated such that upon cutting a forest, the motivation for literals higher up in the forest is not cut away – as it was in the above example. We can obtain this by enforcing $\delta_m^+$ for variables $t_m$ to be empty in rules

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

Such *local FoLPs* can motivate an $a(s)$ ($f(s,t)$) in an open answer set, by descending at most one level in the tree, where one can locally prove $a(s)$ ($f(s,t)$), i.e., without the need to go further down the tree. Of course, in the level below $s$ one may need to check more literals which could amount to going further down the tree, but whilst doing this, one does not need to remember which literals need to be proved above in the tree. In a way a local FoLP has limited memory: it only remembers the previous (predecessor) state.

**Definition 4.11.** *A* local FoLP *is a FoLP where unary rules*

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

*are such that* $\delta_m^+ = \emptyset$ *if* $t_m$ *is a variable,* $1 \leq m \leq k$*, and binary rules*

$$f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t)$$

*are such that* $\delta^+ = \emptyset$ *if* $t$ *is a variable.*

Note that the restrictions in the definition of local FoLPs can be loosened up by allowing for predicates $b$ in $\delta_m^+$ for a variable $t_m$ if $b(X) \vee not\ b(X) \leftarrow$ is in the program; and similarly for $\delta$ in binary rules. Call such FoLPs *semi-local*.

**Definition 4.12.** *A* semi-local FoLP *is a FoLP where unary rules*

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

*are such that* $\delta_m^+ \subseteq \{b \mid b(X) \vee not\ b(X) \leftarrow \in P\}$ *if* $t_m$ *is a variable,* $1 \leq m \leq k$*, and binary rules*

$$f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t)$$

*are such that* $\delta^+ \subseteq \{b \mid b(X) \vee not\ b(X) \leftarrow \in P\}$ *if* $t$ *is a variable.*

*Example 4.13.* The program from Example 4.1 is a local FoLP while the program from Example 4.10 is neither local nor semi-local.

One can indeed replace such $b \in \delta_m^+$ by a double negation. Formally, for a FoLP $P$, we define $\phi(P)$ as the program where each unary rule

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta_m(t_m), \psi$$

is replaced by

$$r : a(s) \leftarrow \beta(s), \bigcup_{1 \leq m \leq k} \gamma_m(s, t_m), \bigcup_{1 \leq m \leq k} \delta'_m(t_m), \psi$$

where

$$\delta'_m \equiv \begin{cases} \delta_m & \text{if } t_m \in cts(P) \\ not\ \delta_m^- \cup \{not\ b' \mid b \in \delta_m^+\} & \text{otherwise} \end{cases}$$

and rules

$$b'(X) \leftarrow not\ b(X)$$

are added for each $b \in \delta_m^+$; and similarly for binary rules. The result of this transformation is indeed a local FoLP.

*Example 4.14.* Take the semi-local FoLP

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

Then, its equivalent local version is

$$\begin{aligned} q(X) &\leftarrow f(X, Y), not\ r'(Y), s(a) \\ r'(Y) &\leftarrow not\ r(Y) \\ r(Z) \vee not\ r(Z) &\leftarrow \end{aligned}$$

**Theorem 4.15.** *Let $P$ be a semi-local FoLP and $p \in upreds(P)$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. the local FoLP $\phi(P)$. Furthermore, the size of $\phi(P)$ is linear in the size of $P$.*

*Proof.* The added rule $b'(X) \leftarrow not\ b(X)$ is a valid local FoLP rule, and since the modified rules replace exactly the positive literals that violate locality by naf-literals, $\phi(P)$ is indeed a local FoLP. Moreover, this translation is linear: the modified rules include *not* while we add a linear number of new rules $b'(X) \leftarrow not\ b(X)$.

For the "only if" direction, assume $p$ is satisfiable w.r.t. $P$, i.e., there is an open answer set $(U, M)$ of $P$ such that $p(y) \in M$. One can show that $(U, M')$ with $M' \equiv M \cup \{b'(x) \mid b(x) \notin M, b' \in \phi(P)\}$ is an open answer set of $\phi(P)$ with $p(y) \in M'$.

- $M'$ is a model of $\phi(P)_U^{M'}$. Indeed, free rules and constraints can be seen to be satisfied by $M'$. Take a rule $b'(x) \leftarrow\in \phi(P)_U^{M'}$ originating from $b'(X) \leftarrow not\ b(X) \in \phi(P)$, such that $b(x) \notin M'$, and thus $b(x) \notin M$ such that, by definition of $M'$, $b'(x) \in M'$.

  Take a unary rule $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s, t_m)[], \delta'^+_m(t_m)[] \in \phi(P)_U^{M'}$ originating from $r : a(s)[] \leftarrow \beta(s)[], \gamma_m(s, t_m)[], \delta'_m(t_m)[], \psi[] \in \phi(P)_U$. Assume $\beta^+(s)[] \cup \gamma_m^+(s, t_m)[] \cup \delta'^+_m(t_m)[] \subseteq M'$. We have that $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s, t_m)[], \delta_m^+(t_m)[] \in P_U^M$. Indeed, take a $b(t_m)[] \in \delta_m^-(t_m)[]$, then $b \in \delta'^-_m$ such that $b(t_m)[] \notin M'$ and $b(t_m)[] \notin M$. Also $\delta_m^+(t_m)[] \subseteq M$. If $t_m$ is a constant then $\delta_m^+ = \delta'^+_m$. Otherwise – $t_m$ is a variable – take $b(t_m)[] \in \delta_m^+(t_m)[]$, then $b'(t_m)[] \in \delta'^-_m(t_m)[]$ such that $b'(t_m)[] \notin M'$ and, by definition of $M'$, $b(t_m)[] \in M$, such that, in general, $\delta_m^+(t_m)[] \subseteq M$.

Thus, $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta_m^+(t_m)[] \in P_U^M$ is applicable in $M$ and $a(s)[] \in M$ such that $a(s)[] \in M'$.
Binary rules can be done similarly.

- $M'$ is a minimal model of $\phi(P)_U^{M'}$. Assume not, i.e., there is a model $N' \subset M'$ of $\phi(P)_U^{M'}$. Define $N \equiv N' \setminus \{b'(x)\}$. Then, $N \subset M$, and we can show that $N$ is a model of $P_U^M$, which is a contradiction with the minimality of $M$.

For the "if" direction, assume $p$ is satisfiable w.r.t. $\phi(P)$, i.e., there is an open answer set $(U, M)$ of $\phi(P)$ such that $p(y) \in M$. Define $M' \equiv M \setminus \{b'(x)\}$, then $(U, M')$ is an open answer set of $P$ and $p(y) \in M'$.

- $M'$ is a model of $P_U^{M'}$. Free rules and constraints can be seen to be satisfied by $M'$.
Take a unary rule $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta_m^+(t_m)[] \in P_U^{M'}$ originating from $r : a(s)[] \leftarrow \beta(s)[], \gamma_m(s,t_m)[], \delta_m[], \psi[] \in P_U$. Assume $\beta^+(s)[] \cup \gamma_m^+(s,t_m)[] \cup \delta_m^+(t_m)[] \subseteq M'$. We have that
$$a(s)[] \leftarrow \beta(s)[], \gamma_m(s,t_m)[], \delta'_m(t_m)[], \psi[] \in \phi(P)_U \ .$$
Take a *not* $b(t_m)[] \in \delta'_m(t_m)[]$, then *not* $b(t_m)[] \in \delta_m(t_m)[]$ and thus $b(t_m)[] \notin M'$ such that $b(t_m)[] \notin M$. Take a *not* $b'(t_m)[] \in \delta'_m(t_m)[]$, then $b(t_m)[] \in \delta_m(t_m)[]$ and thus $b(t_m)[] \in M'$ and $b(t_m)[] \in M$, such that $b'(t_m)[] \leftarrow\notin \phi(P)_U^M$. Since $M$ is an open answer set, and thus minimal, we have that $b'(t_m)[] \notin M$. And thus, $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta'^+_m(t_m)[] \in \phi(P)_U^M$ is applicable in $M$ and $a(s)[] \in M$ such that $a(s)[] \in M'$.
Binary rules can be done similarly.
- $M'$ is a minimal model of $P_U^{M'}$. Assume not, then there is a model $N' \subset M'$ of $P_U^{M'}$. Define $N \equiv N' \cup \{b'(x) \in M\}$. Then, $N \subset M$. We show that $N$ is a model of $\phi(P)_U^M$, which is a contradiction with the minimality of $M$.
Free rules and constraints can be checked.
Take a rule $b'(x) \leftarrow\in \phi(P)_U^M$, then $b'(x) \in M$, such that, by definition of $N$, $b'(x) \in N$.
Take a unary rule $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta'^+_m(t_m)[] \in \phi(P)_U^M$ originating from $r : a(s)[] \leftarrow \beta(s)[], \gamma_m(s,t_m)[], \delta'_m[], \psi[] \in \phi(P)_U$. Assume $\beta^+(s)[] \cup \gamma_m^+(s,t_m)[] \cup \delta'^+_m(t_m)[] \subseteq N$. We have that $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta_m^+(t_m)[] \in P_U^{M'}$. Indeed, take a $b(t_m)[] \in \delta_m^-(t_m)[]$, then $b \in \delta'^-_m$ such that $b(t_m)[] \notin M$ and $b(t_m)[] \notin M'$. Take now a $b(t_m) \in \delta_m^+(t_m)[]$. If $t_m$ is a constant then $b(t_m)[] \in \delta'^+_m(t_m)[]$ such that $b(t_m) \in N'$. Otherwise, $t_m$ is a variable and *not* $b'(t_m)[] \in \delta'_m(t_m)[]$ such that $b'(t_m)[] \notin M$ and thus, by minimality of $M$, $b(t_m)[] \in M$ and $b(t_m)[] \in M'$. Since there is a free rule (by semi-locality) $b(X)[] \vee$ *not* $b(X)[] \leftarrow\in \phi(P)$ with $X[] = t_m[]$, we have that $b(t_m)[] \vee$ *not* $b(t_m)[] \leftarrow\in P_U$ and $b(t_m)[] \leftarrow\in P_U^{M'}$, and, since $N'$ is a model of $P_U^{M'}$, we have that $b(t_m) \in N'$. Thus, in general, $\delta_m^+(t_m)[] \subseteq N'$. Thus, $a(s)[] \leftarrow \beta^+(s)[], \gamma_m^+(s,t_m)[], \delta_m^+(t_m)[] \in P_U^{M'}$ is applicable in $N'$ such that $a(s)[] \in N'$ and $a(s)[] \in N$.

Binary rules can be done similarly.

□

**Theorem 4.16.** *Let $P$ be a local FoLP. Then, $P$ has the bounded finite model property.*

*Proof.* Assume $p$ is satisfiable w.r.t. $P$. Since $P$ has the forest model property, there is a forest model $(U, M)$ with $p(\varepsilon) \in M$. Let $U$ be the set of nodes from trees $U_x$ which have roots $x$; thus $\varepsilon$ is one of those $x$'s. Let $m$ be the number of different labels in the forest model. Note that $m$ is at most $2^u$ where $u \equiv |\mathit{upreds}(P)|$. For a path $\mathcal{P}$ of length at least $m+2$ in some $U_x$, define $z_\mathcal{P} \in U_x$ as the minimal node in $U_x$ (w.r.t. the prefix relation $<$) s.t.

$$\exists y < z_\mathcal{P} \cdot y \notin \mathit{cts}(P) \wedge \mathcal{L}(y) = \mathcal{L}(z_\mathcal{P}) \ .$$

Denote this unique $y$ with $\overline{z}_\mathcal{P}$. Note that for paths $\mathcal{P}$ of length at least $m+2$, $z_\mathcal{P}$ and $\overline{z}_\mathcal{P}$ are always defined: we have a finite number $m$ of different labels, such that, for every path $\mathcal{P}$ of length $m+1$, there are two nodes with the same label, moreover, in the worst case we only need a path of length $m+2$ to make sure that $\overline{z}_\mathcal{P}$ is not a constant.

Intuitively, $z_\mathcal{P}$ repeats the label of $\overline{z}_\mathcal{P}$ and will function as a cutting point. The connections from $z_\mathcal{P}$ will be copies of the connections from $\overline{z}_\mathcal{P}$ and the atoms at $z_\mathcal{P}$ will be motivated by the same rules that motivate those atoms at $\overline{z}_\mathcal{P}$. The latter also explains why neither $z_\mathcal{P}$ nor $\overline{z}_\mathcal{P}$ are allowed to be constants. Constants may be introduced by rules containing no variables in the head, which, consequently, cannot be used to motivate the presence of literals at anonymous nodes: it might be that a rule $t(a) \leftarrow$ introduces $t$ in the label of some constant $a$, however, such a rule cannot be used to motivate the presence of $t$ lower in the tree. Below the root, we would not have this problem as $t$ would be motivated there by a rule with head $t(X)$, which can be matched against any lower node.

Define $U'_x$ as follows

$$U'_x \equiv \{z \in U_x \mid (z \in \mathcal{P} \wedge |\mathcal{P}| > m+1) \Rightarrow z \leq z_\mathcal{P}\} \ ,$$

i.e., cut the tree $U_x$ at $z_\mathcal{P}$ for every path $\mathcal{P}$ in $U_x$ that has length at least $m+2$. Let $U' \equiv \cup\{U'_x \mid U_x \in U\}$. Define

$$\begin{aligned} M' \equiv \{&q(z) \mid z \in U', q(z) \in M\} \\ &\cup \{f(z,y) \mid z \in \mathcal{P} \wedge |\mathcal{P}| > m+1 \Rightarrow z < z_\mathcal{P}, f(z,y) \in M\} \\ &\cup \{f(z_\mathcal{P}, y) \mid |\mathcal{P}| > m+1, f(\overline{z}_\mathcal{P}, y) \in M\} \ . \end{aligned}$$

Intuitively, copy the connections from the first node of the duplicate pair to the second node of the pair.

From Theorem 4.7, we have that the branching of a $U_x$ is at most the rank $n$ of $P$, such that the number of nodes in $U'_x$ is at most $\sum_{i=0}^{m+1} n^i$. Since $m$ is

at most $2^u$ for $u \equiv |upreds(P)|$, we have that the number of elements in $U_x$ is at most $\sum_{i=0}^{2^u+1} n^i$ (and this expression is defined in function of $P$ only, not in function of the forest model $(U, M)$).

We have that $U'$ contains the nodes of at most $c+1$ trees $U'_x$, where $c \equiv |cts(P)|$, such that the cardinality of $U'$ is at most

$$k \equiv (c+1) \sum_{i=0}^{2^u+1} n^i \,, \tag{4.4}$$

where $k$ is calculated in function of $P$ only.

Note that $p(\varepsilon) \in M'$, such that it remains to show that $(U', M')$ is an open answer set of $P$.

- $M'$ is a model of $P_{U'}^{M'}$. Free rules and constraints from $P_{U'}^{M'}$ can be easily seen to be satisfied by $M'$.
  Take a unary rule $r : a(x) \leftarrow \alpha^+(x), \gamma_i^+(x, y_i), \beta_i^+(y_i) \in P_{U'}^{M'}$ orig. from $a(s) \leftarrow \alpha(s), \gamma_i(s, t_i), \beta_i(t_i), t_i \neq t_j \in P$, and $y_i \neq y_j$ for $t_i \neq t_j$. Take body$(r) \subseteq M'$.
  - If $x = z_\mathcal{P}$ for some path $\mathcal{P}$ in some $U_x$ of length at least $m+2$, then $\gamma_i^+(\overline{z}_\mathcal{P}, y_i) \subseteq M$, $\beta_i^+(y_i) \subseteq M$ and $\alpha^+(\overline{z}_\mathcal{P}) \subseteq M$. Furthermore, $\gamma_i{}^-(\overline{z}_\mathcal{P}, y_i) \cap M = \alpha^-(\overline{z}_\mathcal{P}) \cap M = \beta_i{}^-(y_i) \cap M = \emptyset$, such that $a(\overline{z}_\mathcal{P}) \leftarrow \alpha^+(\overline{z}_\mathcal{P}), \gamma_i^+(\overline{z}_\mathcal{P}, y_i), \beta_i^+(y_i) \in P_U^M$, and since the body is true in $M$, we have that $a(\overline{z}_\mathcal{P}) \in M$ such that $a(z_\mathcal{P}) \in M$.
  - Otherwise ($x$ does not lie on such a path $\mathcal{P}$, or if it does, then $x < z_\mathcal{P}$). Then, $a(x) \leftarrow \alpha^+(x), \gamma_i^+(x, y_i), \beta_i^+(y_i) \in P_U^M$ such that $a(x) \in M$ and thus $a(x) \in M'$.

  Binary rules can be done similarly .
- $M'$ is a minimal model of $P_{U'}^{M'}$. One can prove this by subsequently showing the following:
  1. if $b(x) \in M', x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$, then $b(x) \in T_{M'}^{k_1}$[6], and if $g(x, y) \in M', x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$, then $g(x, y) \in T_{M'}^{k_2}$, for finite $k_1$, $k_2$.
  2. if $b(z_\mathcal{P}) \in M'$, then $b(z_\mathcal{P}) \in T_{M'}^{k_1}$ and if $g(z_\mathcal{P}, y) \in M'$, then $g(z_\mathcal{P}, y) \in T_{M'}^{k_2}$, for finite $k_1$ and $k_2$.
  3. if $b(x) \in M'$, then $b(x) \in T_{M'}^{k_1}$, and if $g(x, y) \in M'$, then $g(x, y) \in T_{M'}^{k_2}$, for finite $k_1$, $k_2$.
  1. *if $b(x) \in M', x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$, then $b(x) \in T_{M'}^{k_1}$, and if $g(x, y) \in M', x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$ then $g(x, y) \in T_{M'}^{k}$, for finite $k_1$, $k_2$.*
     Assume $b(x) \in M'$ and $g(x, y) \in M'$. Then, $b(x) \in M$ and $g(x, y) \in M$ such that $b(x) \in T_M^{n_1}$ and $g(x, y) \in T_M^{n_2}$ for $n_1$ and $n_2$ finite. We prove it by induction on $n_1$ and $n_2$.
     - BASE CASES.
       - $n_1 = 1$. Then, $b(x) \leftarrow \in P_U^M$, which

[6] Note that we subscript the immediate consequence operator with $M'$ instead of superscripting it with $(U, M')$ as on pp. 66; this to avoid cluttering up notation.

- orig. from $b(s) \vee not\ b(s) \leftarrow\ \in P$ such that $b(x) \leftarrow\in P^{M'}_{U'}$ and thus $b(x) \in T^1_{M'}$, or
- orig. from $r : b(x) \leftarrow \beta(x), \gamma_m(x, t_m[]), \delta_m(t_m[]), \psi[] \in P_U$ with $\text{body}(r)^+ = \emptyset$. Each $t_m[]$ must be a constant (otherwise, by the definition of FoLPs, $\gamma^+_m \neq \emptyset$), and thus $x \in U'$ and each $t_m[] \in U'$ such that $r \in P_{U'}$. Furthermore, $\beta^-(x) \cap M = \gamma^-_m(x, t_m[]) \cap M = \delta^-_m(t_m[]) \cap M = \emptyset$, such that $\beta^-(x) \cap M' = \gamma^-_m(x, t_m[]) \cap M' = \delta^-_m(t_m[]) \cap M' = \emptyset$. Then, $b(x) \leftarrow\in P^{M'}_{U'}$ s.t. $b(x) \in T^1_{M'}$.

· $n_2 = 1$. This can be done similarly.

– INDUCTION HYPOTHESIS. Assume that for $b(x) \in M'$, $b(x) \in T^{n_1-1}_M$, $x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$, and $g(x, y) \in M'$, $g(x, y) \in T^{n_2-1}_M$, $x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$, for $n_1$ and $n_2$ finite, then $b(x) \in T^{l_1}_{M'}$ and $g(x, y) \in T^{l_2}_{M'}$, for finite $l_1$ and $l_2$.

– INDUCTION. Take $b(x) \in M'$, $b(x) \in T^{n_1}_M$, $x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$ and $g(x, y) \in M'$, $g(x, y) \in T^{n_2}_M$, $x \in \mathcal{P} \Rightarrow x < z_\mathcal{P}$.

· $b(x) \in T^{n_1}_M$, then $r : b(x) \leftarrow \alpha^+(x), \gamma^+_i(x, y_i), \beta^+_i(y_i) \in P^M_U$ with $\text{body}(r) \subseteq T^{n_1-1}_M$, originating from

$$b(s) \leftarrow \alpha(s), \gamma_i(s, t_i), \beta_i(t_i), t_i \neq t_j \in P^{7}\ ,$$

$y_i \neq y_j$ for $t_i \neq t_j$, and $\alpha^-(x) \cap M = \gamma^-_i(x, y_i) \cap M = \beta^-_i(y_i) \cap M = \emptyset$.

Either $y_i$ is a constant or there is a $g(x, y_i) \in \gamma^+_i(x, y_i)$ such that $g(x, y_i) \in M$. In both cases, we have that $y_i \in U'$, such that $b(x) \leftarrow \alpha(x), \gamma_i(x, y_i), \beta_i(y_i), y_i \neq y_j \in P_{U'}$. Moreover, $\alpha^-(x) \cap M' = \gamma^-_i(x, y_i) \cap M' = \beta^-_i(y_i) \cap M' = \emptyset$, such that $b(x) \leftarrow \alpha^+(x), \gamma^+_i(x, y_i), \beta^+_i(y_i) \in P^{M'}_{U'}$.

We have that $\alpha^+(x) \subseteq T^{k_1-1}_M \cap M'$ and $x < z_\mathcal{P}$ such that, by induction, $\alpha^+(x) \subseteq T^k_{M'}$ for some finite $k$. The same applies for $\gamma^+_i(x, y_i)$. If $\beta^+_i(y_i) \neq \emptyset$, we have that $y_i$ must be a constant[8] such that if $y_i \in \mathcal{P}$, then $y_i < z_\mathcal{P}$, and we can, again by induction, show that $\beta^+_i(y_i) \subseteq T^k_{M'}$ for some finite $k$.

We have that the body of the latter rule is in some $T^k_{M'}$, $k < \infty$, such that $b(x) \in T^{k_1}_{M'}$ for some finite $k_1$.

· Take $g(x, y) \in T^{n_2}_M$, this can be done similarly.

2. $b(z_\mathcal{P}) \in M' \Rightarrow b(z_\mathcal{P}) \in T^{k_1}_{M'}$ *and* $g(z_\mathcal{P}, y) \in M' \Rightarrow g(z_\mathcal{P}, y) \in T^{k_2}_{M'}$ *for finite* $k_1$ *and* $k_2$.

Assume $b(z_\mathcal{P}) \in M'$ and $g(z_\mathcal{P}, y) \in M'$. Then, $b(\overline{z}_\mathcal{P}) \in M'$ since $\mathcal{L}(\overline{z}_\mathcal{P}) = \mathcal{L}(z_\mathcal{P})$ and $g(\overline{z}_\mathcal{P}, y) \in M'$ by definition of $M'$. We have that $b(\overline{z}_\mathcal{P}) \in M$ and $g(\overline{z}_\mathcal{P}, y) \in M$ such that $b(\overline{z}_\mathcal{P}) \in T^{n_1}_M$ and $g(\overline{z}_\mathcal{P}, y) \in T^{n_2}_M$ for finite $n_1$ and $n_2$. We prove it by induction on $n_1$ and $n_2$.

[7] Note that ${\beta_i}^+(y_i) = \emptyset$ if $t_i$ is a variable, by definition of local FoLPs.

[8] This is where we use the locality.

- BASE CASES.
  - $n_1 = 1$. Then, $b(\overline{z}_\mathcal{P}) \leftarrow \ \in P_U^M$,
    - orig. from $b(s) \vee not\ b(s) \leftarrow \ \in P$. Since $\overline{z}_\mathcal{P}$ is not a constant, we have that $s$ is a variable, and thus $b(z_\mathcal{P}) \leftarrow\in P_{U'}^{M'}$ and thus $b(z_\mathcal{P}) \in T_{M'}^1$.
    - orig. from $r : b(s) \leftarrow \beta(s), \gamma_m(s, t_m), \delta_m(t_m) \in P$. Each $t_m$ must be a constant (otherwise, by the definition of FoLPs, $\gamma_m^+ \neq \emptyset$). Again, since $\overline{z}_\mathcal{P}$ is not a constant, $s$ must be a variable. With $\overline{z}_\mathcal{P} \in U'$ and $t_m[] \in U'$, we have then $b(\overline{z}_\mathcal{P}) \leftarrow \beta(\overline{z}_\mathcal{P}), \gamma_m(\overline{z}_\mathcal{P}, t_m[]), \delta_m(t_m[]) \in P_{U'}$. Furthermore, $\beta^-(\overline{z}_\mathcal{P}) \cap M = \gamma_m^-(\overline{z}_\mathcal{P}, t_m[]) \cap M = \delta_m^-(t_m[]) \cap M = \emptyset$, such that $\beta^-(z_\mathcal{P}) \cap M' = \gamma_m^-(z_\mathcal{P}, t_m[]) \cap M' = \delta_m^-(t_m[]) \cap M' = \emptyset$. Then, $b(z_\mathcal{P}) \leftarrow\in P_{U'}^{M'}$ s.t. $b(z_\mathcal{P}) \in T_{M'}^1$.
  - $n_2 = 1$. This can be done similarly.
- INDUCTION HYPOTHESIS. Assume that for $b(\overline{z}_\mathcal{P}) \in T_M^{n_1 - 1}$ and $g(\overline{z}_\mathcal{P}, y) \in T_M^{n_2 - 1}$ for $n_1$ and $n_2$ finite, $b(z_\mathcal{P}) \in T_{M'}^{l_1}$ and $g(z_\mathcal{P}, y) \in T_{M'}^{l_2}$, for finite $l_1$ and $l_2$.
- INDUCTION. Take $b(\overline{z}_\mathcal{P}) \in T_M^{n_1}$ and $g(\overline{z}_\mathcal{P}, y) \in T_M^{n_2}$.
  - $b(\overline{z}_\mathcal{P}) \in T_M^{n_1}$, then $r : b(\overline{z}_\mathcal{P}) \leftarrow \alpha^+(\overline{z}_\mathcal{P}), \gamma_i^+(\overline{z}_\mathcal{P}, y_i), \beta_i^+(y_i) \in P_U^M$ with $\text{body}(r) \subseteq T_M^{n_1 - 1}$, originating from

$$b(s) \leftarrow \alpha(s), \gamma_i(s, t_i), \beta_i(t_i), t_i \neq t_j \in P\ ,$$

    $y_i \neq y_j$ for $t_i \neq t_j$, and $\alpha(\overline{z}_\mathcal{P})^- \cap M = \gamma_i^-(\overline{z}_\mathcal{P}, y_i) \cap M = \beta_i^-(y_i) \cap M = \emptyset$. We have that $\overline{z}_\mathcal{P}$ is not a constant such that $s$ is a variable. We have that $z_\mathcal{P} \in U'$ and each $y_i \in U'$ ($y_i$ is constant or there is a $f(\overline{z}_\mathcal{P}, y_i) \in M$ such that $y_i$ is a successor of $\overline{z}_\mathcal{P}$ and thus in $U'$). Thus, $b(z_\mathcal{P}) \leftarrow \alpha(z_\mathcal{P}), \gamma_i(z_\mathcal{P}, y_i), \beta_i(y_i), y_i \neq y_j \in P_{U'}$. By induction, we have that $\alpha^+(z_\mathcal{P}) \cup \gamma_i^+(z_\mathcal{P}, y_i) \cup \beta_i^+(y_i) \subseteq T_{M'}^{l_1}$. Indeed (for $\beta_i$), if $\beta_i^+ \neq \emptyset$, then $y_i$ is a constant (by locality) such that $y_i \in \mathcal{P} \Rightarrow y_i < z_\mathcal{P}$, and thus, by 1., $\beta_i^+(y_i) \subseteq T_{M'}^k$ for some finite $k$. Furthermore, $\alpha^-(z_\mathcal{P}) \cap M' = \gamma_i^-(z_\mathcal{P}, y_i) \cap M' = \beta_i^-(y_i) \cap M' = \emptyset$ such that $b(z_\mathcal{P}) \leftarrow \alpha^+(z_\mathcal{P}), \gamma_i^+(z_\mathcal{P}, y_i), \beta_i^+(y_i) \in P_{U'}^{M'}$ and thus $b(z_\mathcal{P}) \in T_{M'}^k$ for some finite $k$.
  - Take $g(\overline{z}_\mathcal{P}, y) \in T_M^{n_2}$, this can be done similarly.

3. *if* $b(x) \in M'$, *then* $b(x) \in T_{M'}^{k_1}$, *and if* $f(x, y) \in M'$ *then* $f(x, y) \in T_{M'}^{k_1}$, *for finite* $k_1$, $k_2$. This follows immediately by 1. and 2.

Assume $M'$ is not minimal, then there is a model $N' \subset M'$ of $P_{U'}^{M'}$ such that there is a $a(x) \in M' \backslash N'$ or $f(x, y) \in M' \backslash N'$. From 3., we have that $a(x) \in T_{M'}^k$ and likewise for $f(x, y)$, such that, since $N'$ is a model of $P_{U'}^{M'}$, $a(x) \in N'$ (and $f(x, y) \in N'$), which is a contradiction.

□

Satisfiability checking w.r.t. local FoLPs can then be done by standard answer set solvers. Intuitively, we introduce a large enough number of constants, such that a bounded finite model can be mapped to these constants.

**Theorem 4.17.** *Let $P$ be a local FoLP. Then, $p \in upreds(P)$ is satisfiable w.r.t. $P$ iff there is a $0 \leq h \leq k$ and an answer set $M$ of $\psi_h(P)$, where $k$ is*

$$k \equiv (c+1) \sum_{i=0}^{2^u+1} n^i \ , \tag{4.5}$$

*with $c \equiv |cts(P)|$, $u \equiv |upreds(P)|$, and $n$ the rank of $P$, and*

$$\psi_h(P) \equiv P \cup \{ \leftarrow not\ \alpha\} \ , \tag{4.6}$$

*where $\alpha \equiv \{p(a) \mid a \in \{x_i \mid 1 \leq i \leq h\} \cup cts(P)\}$.*

*Proof.* For the "only if" direction, assume $p$ is satisfiable w.r.t. $P$, such that, by Theorem 4.16, there is an open answer set $(U', M')$ of $P$, with $|U'| \leq k$ and a $p(u) \in M'$. Define $h \equiv |U'| - |cts(P)|$, i.e., the number of anonymous elements in $U'$: since $cts(P) \subseteq U'$, we have that $0 \leq h \leq k$. Define a bijection $F : U' \rightarrow cts(\psi_h(P))$ such that $F(a) = a$ for $a \in cts(P)$, which is always possible since $cts(P) \subseteq cts(\psi_h(P))$ and $|U'| = h + |cts(P)| = |cts(\psi_h(P))|$. Take

$$M \equiv \{a(F(x)) \mid a(x) \in M'\} \cup \{f(F(x), F(y)) \mid f(x,y) \in M'\} \ .$$

Intuitively, we identify $U'$ with the constants in $\psi_h(P)$ making sure the original constants in $P$ are mapped to the same constants in $\psi_h(P)$. One can show that $M$ is an answer set of $\psi_h(P)$.

- $M$ is a model of $gr(\psi_h(P))^M$. Free rules and constraints can be easily checked for satisfaction. Take a unary rule
  $a(x) \leftarrow \alpha^+(x), {\gamma_i}^+(x, y_i), {\beta_i}^+(y_i) \in gr(\psi_h(P))^M$, originating from $a(s) \leftarrow \alpha(s), \gamma_i(s, t_i), \beta_i(t_i), t_i \neq t_j \in \psi(P)$ (and thus also in $P$), such that $\alpha^-(x) \cap M = \gamma_i^-(x, y_i) \cap M = \beta_i^-(y_i) \cap M = \emptyset$ and $y_i \neq y_j$ for $t_i \neq t_j$. Assume that $\alpha^+(x) \cup \gamma_i^+(x, y_i) \cup \beta_i^+(y_i) \subseteq M$.
  We have that for each $y \in cts(\psi_h(P))$ there is a $y' \in U'$ such that $F(y') = y$, and thus $a(x') \leftarrow \alpha(x'), \gamma_i(x', y_i'), \beta_i(y_i'), y_i' \neq y_j' \in P_{U'}$ (note that $F(y') = y = y'$ for constants $y$). Furthermore $\alpha^-(x') \cap M' = \gamma_i^-(x', y_i') \cap M' = \beta_i^-(y_i') \cap M = \emptyset$ and $y_i' \neq y_j'$. Indeed, take $b(x') \in \alpha^-(x') \cap M'$, then $b(F(x')) \in M$, by definition of $M$, and, since $F(x') = x$, $b(x) \in M$ for $b \in \alpha^-$, a contradiction. The other cases can be done similarly; that $y_i' \neq y_j'$ follows since $F$ is a bijection. Thus $a(x') \leftarrow \alpha^+(x'), {\gamma_i}^+(x', y_i'), {\beta_i}^+(y_i') \in P_{U'}^{M'}$.
  Moreover, $\alpha^+(x') \cup {\gamma_i}^+(x', y_i') \cup {\beta_i}^+(y_i') \subseteq M'$ such that $a(x') \in M'$ and thus $a(F(x')) \in M$, by definition of $M$, such that $a(x) \in M$ since $F(x') = x$.

Binary rules can be checked similarly. Since there is a $p(u) \in M'$, we have that $p(F(u)) \in M$ and $\leftarrow not\ \alpha \notin gr(\psi_h(P))^M$.

- $M$ is a minimal model of $gr(\psi_h(P))^M$. Assume not, then there is a model $N \subset M$ of $gr(\psi_h(P))^M$. Define

$$N' \equiv \{a(x) \mid a(F(x)) \in N\} \cup \{f(x,y) \mid f(F(x),F(y)) \in N\}\ .$$

Then, $N' \subset M'$ and one can show that $N'$ is a model of $P_{U'}^{M'}$, which is a contradiction.

For the "if" direction, assume there exists an answer set $M$ of $\psi_h(P)$ for $0 \leq h \leq k$. Define $U' \equiv cts(\psi_h(P))$. One can show that $(U', M)$ is an open answer set of $P$. Since $\leftarrow not\ \alpha \in \psi_h(P)$ and $\alpha \neq \emptyset$ (otherwise $\psi_h(P)$ would have no answer set), there must be at least one $p(b) \in M$ for $b \in cts(\psi_h(P))$. □

Local FoLPs may contain negation as failure in the head (with free rules) such that $\psi_h(P)$ may as well, which is not allowed by standard answer set solvers such as DLV or SMODELS. One can, however, remove negation as failure from the heads in $\psi_h(P)$ as in [IS98].

With Theorem 4.17, one can then subsequently check satisfiability by letting $h$ range from 0 to $k$ and checking whether $\psi_h(P)$ has an answer set; the latter can be done with standard answer solvers.

Note that CoLPs without inverted predicates are FoLPs (Theorem 4.5), such that *local* CoLPs have the bounded finite model property as well (and satisfiability checking w.r.t. such CoLPs can also be reduced to finite answer set programming).

**Definition 4.18.** *A* local CoLP *is a CoLP where unary rules*

$$r : a(X) \leftarrow \beta(X), \bigcup_{1 \leq m \leq k} \gamma_m(X, Y_m), \bigcup_{1 \leq m \leq k} \delta_m(Y_m), \psi$$

*are such that* $\delta_m^+ = \emptyset$, $1 \leq m \leq k$, *and binary rules*

$$f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$$

*are such that* $\delta^+ = \emptyset$.

**Theorem 4.19.** *Let $P$ be a local CoLP without inverted predicates. Then, $P$ has the bounded finite model property.*

*Proof.* By Theorem 4.5, $P$ is a FoLP. Furthermore, $P$ is local such that the result follows from Theorem 4.16. □

## 4.3 Acyclic Programs

We identify a class of programs, *acyclic programs*, such that satisfiability checking of acyclic FoLPs can be reduced to satisfiability checking w.r.t. local FoLPs. Acyclic programs will be further used in Chapter 6.

Formally, a *positive predicate dependency graph* $PDG(P)$ for a program $P$ is defined by edges between (non-equality) predicates $a$ and $b$ such that $a \to b$ iff there is a rule $\alpha \leftarrow \beta \in P$ such that $a$ is a predicate from $\alpha^+$ and $b$ is a predicate from $\beta^+$.

**Definition 4.20.** *Let $P$ be a program. $P$ is* (positively) acyclic*, if $PDG(P)$ does not contain cycles.*

Acyclic programs are programs that do not allow recursion through positive literals. A distinction with stratified programs [Bar03] or with the hierarchical programs from pp. 107 is that recursion through negated literals is still allowed.

A useful property of acyclic programs is that they can be rewritten such that there appear no positive unary literals in the body anymore; one replaces them by a double negation.

Formally, for a program $P$, we define $\kappa(P)$ as the program $P$ where rules $r : \alpha \leftarrow \beta, \gamma$ with $\beta$ the unary atoms of body$(r)$, are replaced by

$$\alpha \leftarrow not\ \beta', \gamma$$

and rules

$$b'(X) \leftarrow not\ b(X)$$

are added for all $b'(s) \in \beta'$ where $\beta' \equiv \{b'(s) \mid b(s) \in \beta\}$.[9]

*Example 4.21.* Take the program $P$

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

The positive dependency graph of this program is $\{a \to b, a \to f\}$ such that $P$ is acyclic. The translation $\kappa(P)$ is then

$$\begin{aligned} a(X) &\leftarrow not\ b'(X), f(X, Y), not\ c(Y) \\ b'(X) &\leftarrow not\ b(X) \\ b(X) \vee not\ b(X) &\leftarrow \\ f(X, Y) \vee not\ f(X, Y) &\leftarrow \end{aligned}$$

which has, among others, the open answer set
$(\{x, y\}, \{a(x), b(x), f(x, y), b'(y)\})$, corresponding to an open answer set
$(\{x, y\}, \{a(x), b(x), f(x, y)\})$ of $P$.

[9] Note that $\kappa(P)$ is very similar to the translation $\phi(P)$ for semi-local FoLPs (pp. 125), only now we replace *every* unary atom in the bodies of $P$.

**Theorem 4.22.** *Let $P$ be a program and $p \in upreds(P)$. If $p$ is satisfiable w.r.t. $P$, then $p$ is satisfiable w.r.t. $\kappa(P)$.*

*Proof.* Assume $(U, M)$ is an open answer set of $P$ such that $p(\mathbf{x}) \in M$. Define $M' \equiv M \cup \{b'(x) \mid b(x) \notin M, b' \in \kappa(P)\}$. Then, one can show that $(U, M')$ is indeed an open answer set of $\kappa(P)$ that satisfies $p$. □

The other direction is in general not valid.

*Example 4.23.* Consider the program $P$

$$a(X) \leftarrow a(X)$$

This is not an acyclic program and $\kappa(P)$ is the program

$$a(X) \leftarrow not\ a'(X)$$
$$a'(X) \leftarrow not\ a(X)$$

with an open answer set $(\{x\}, \{a(x)\})$, which does not correspond to any open answer set of $P$.

For acyclic programs, however, $P$ and $\kappa(P)$ are equivalent w.r.t. satisfiability checking.

**Theorem 4.24.** *Let $P$ be an acyclic program and $p \in upreds(P)$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. $\kappa(P)$.*

*Proof.* The "only if" direction follows from Theorem 4.22.

For the "if" direction, assume $p$ is satisfiable w.r.t. $\kappa(P)$, i.e., there is an open answer set $(U, M)$ of $\kappa(P)$ such that $p(y) \in M$. Define $M' \equiv M \backslash \{b'(x)\}$, then $(U, M')$ is an open answer set of $P$ and $p(y) \in M'$.

- $M'$ is a model of $P_U^{M'}$. This is again along the lines of the proof of Theorem 4.15.
- $M'$ is a minimal model of $P_U^{M'}$. Assume not, then there is a model $N' \subset M'$ of $P_U^{M'}$.
  We prove that $M' \subseteq N'$. Take $l \in M'$, we prove that $l \in N'$ by induction on the maximum depth[10] of $PDG(P)$ of the predicate $preds(l)$ in $l$, which is possible since $P$ is acyclic and $PDG(P)$ is finite.
  - $preds(l)$ has depth 0 in $PDG(P)$. Then, all rules $\alpha \leftarrow \beta \in P$ with $preds(l)$ in $\alpha^+$ are such that $\beta$ is a set of of naf-atoms and/or equality atoms. Consequently $\alpha \leftarrow \beta \in \kappa(P)$ and it does not contain any newly added $b'(s)$'s. Since $l \in M$ we have that there is a $l \leftarrow \beta^+[] \in \kappa(P)_U^M$ with $\emptyset \models \beta^+[]$[11] and $M \models \alpha^-[] \cup not\ \beta^-[]$ originating from $\alpha[] \leftarrow \beta[] \in \kappa(P)_U$. Then $\alpha[] \leftarrow \beta[] \in P_U$ and $M' \models \alpha^-[] \cup not\ \beta^-[]$ such that $l \leftarrow \beta^+[] \in P_U^{M'}$ with $\emptyset \models \beta^+[]$. Since $N'$ is a model of $P_U^{M'}$, $l \in N'$.

[10] A predicate $p$ has depth 0 if it has no successors in $PDG(P)$ and depth $n$ if the maximum depth of its successors in $PDG(P)$ is $n-1$.

[11] $\beta^+$ contains, if anything, only equality atoms.

– Assume it is proved for literals $l$ with depth of $preds(l)$ at most $n-1$ (IH).
– Take $l$ with depth of $preds(l)$ at most $n$. Then all rules $\alpha \leftarrow \beta, \gamma$ in $P$ with $preds(l)$ in $\alpha^+$, where $\beta$ is a set of unary atoms, and $\gamma$ the rest, are such that the predicates in $\beta$ and $\gamma^+$ have a depth of at most $n-1$. We have that $\alpha \leftarrow not\ \beta', \gamma \in \kappa(P)$ and $b'(X) \leftarrow not\ b(X) \in \kappa(P)$ for $b \in preds(\beta)$.
Since $l \in M$, we have that there is a $l \leftarrow \gamma^+[] \in \kappa(P)^M_U$, with $M \models \gamma^+[]$, $M \models \alpha^-[] \cup not\ \gamma^-[] \cup not\ \beta'[]$.
We have that $l \leftarrow \beta[], \gamma^+[] \in P^{M'}_U$ with the body true in $N'$. Indeed, we have that $\alpha \leftarrow \beta, \gamma \in P$, and $M' \models \alpha^-[] \cup not\ \gamma^-[]$. Furthermore, $N' \models \beta[]$ and $N' \models \gamma^+[]$, by the fact that $M \models \beta[]$ and $M \models \gamma^+[]$ and the induction hypothesis. The former can be seen by noting that $M \models not\ \beta'[]$. Take then a $b(x) \in \beta[]$, then $b'(x) \in \beta'[]$ such that $b'(x) \notin M$ and thus, by $b'(X) \leftarrow not\ b(X) \in \kappa(P)$, we have that $b(x) \in M$.
Since $N'$ is a model of $P^{M'}_U$, we then have that $l \in N'$. Thus $M' \subseteq N'$, a contradiction with $N' \subset M'$.

□

**Theorem 4.25.** *Let $P$ be an acyclic FoLP. Then, $\kappa(P)$ is a local FoLP that has a size linear in the size of $P$.*

*Proof.* The added rule $b'(X) \leftarrow not\ b(X)$ is a valid local FoLP rule, and since in the modified rules all unary atoms are replaced by their naf variants, $\kappa(P)$ is a local FoLP. Moreover, this translation is linear: the modified rules include *not* while we add a linear number of new rules $b'(X) \leftarrow not\ b(X)$. □

Together with Theorem 4.24, the latter theorem allows to reduce satisfiability checking of acyclic FoLPs to local FoLPs, and thus, by Theorem 4.17, to finite answer set programming.

## 4.4 Complexity

Let $P$ be a local FoLP. We verify the complexity of checking whether there exists an answer set $M$ of $\psi_h(P)$ for some $0 \leq h \leq k$ where $k$ and $\psi_h(P)$ are as in Equations (4.5) and (4.6) respectively. We distinguish between two cases:

- If FoLP rules have a degree bounded by $m$, independent of a particular FoLP, then the size of $gr(\psi_h(P))$ is polynomial in the size of $\psi_h(P)$, since every rule in $\psi_h(P)$ introduces at most $\mathcal{O}(|cts(\psi_h(P))|^{m+1})$ rules in $gr(\psi_h(P))$. Indeed, each FoLP rule then contains at most $m+1$ variables, each of which can be instantiated with a constant from $\psi_h(P)$. Since checking whether there exists an answer set $M$ of $\psi_h(P)$ is in NP in the size of

$gr(\psi_h(P))$ [DEGV01, Bar03], we have that checking whether there exists an answer set $M$ of $\psi_h(P)$ is in NP in the size of $\psi_h(P)$ as well.

- If the degree is not bounded, we use a result from [EFF$^+$04] to state that checking whether $M$ is an answer of $\psi_h(P)$ is in $\Sigma_2^{\mathrm{P}}$ w.r.t. the size of $\psi_h(P)$.[12] Indeed, the arities of predicates in $\psi_h(P)$ are bounded by 2 since FoLPs allow only for unary and binary predicates.

Thus, for a fixed $h$, checking whether $\psi_h(P)$ has an answer set is in NP for a FoLP with bounded degree and in $\Sigma_2^{\mathrm{P}}$ in general.

Satisfiability checking of a predicate w.r.t. $P$ can then be done by starting with $h = 0$ and checking whether $\psi_h(P)$ has an answer set. If this is the case, we are done (by Theorem 4.17), otherwise, we repeat the check for $h = 1$, and so on. If finally $h = k$ has been checked, i.e., $\psi_h(P)$ had no answer sets, one can conclude, by Theorem 4.17, that the predicate is not satisfiable. This procedure thus involves at most $k + 1$ calls to an NP oracle for FoLPs with bounded degree or to an $\Sigma_2^{\mathrm{P}}$ oracle in general.

We have that

$$k = (c+1)\sum_{i=0}^{2^u+1} n^i = (c+1)\frac{(1-n^{2^u+2})}{(1-n)}\ ,$$

with $u = |upreds(P)|$, $c = |cts(P)|$, and $n$ the rank of $P$ such that $k$ is double exponential in the size of $P$ and the above procedure to check satisfiability runs in 2-EXPTIME$^{\mathrm{NP}}$ for local FoLPs with bounded degree or in 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ for arbitrary local FoLPs.

**Theorem 4.26.** *Satisfiability checking w.r.t. local FoLPs is in* 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ *for FoLP rules with unbounded degree or in* 2-EXPTIME$^{\mathrm{NP}}$ *otherwise.*

*Proof.* From the above exposition. □

**Theorem 4.27.** *Satisfiability checking w.r.t. semi-local FoLPs is in* 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ *for FoLP rules with unbounded degree or in* 2-EXPTIME$^{\mathrm{NP}}$ *otherwise.*

*Proof.* With Theorem 4.15, we can translate a semi-local FoLP to an equivalent local FoLP that has a size linear in the size of original program. The result follows from Theorem 4.26. □

**Theorem 4.28.** *Satisfiability checking w.r.t. acyclic FoLPs is in* 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ *for FoLP rules with unbounded degree or in* 2-EXPTIME$^{\mathrm{NP}}$ *otherwise.*

*Proof.* With Theorem 4.24, we can translate an acyclic FoLP to an equivalent local FoLP that has a size linear in the size of original program. The result follows from Theorem 4.26. □

[12] Recall that $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$.

## 4.5 Extended Forest Logic Programs

Consider a FoLP defining when one cheats one's spouse, i.e., if one is married to someone that is different than the person one is dating. We have a specialized rule saying that when one is cheating one's spouse with the spouse's friend Jane. Furthermore, a constraint ensures that cheaters date cheaters.

$$\begin{aligned} cheats(X) &\leftarrow marr(X, Y_1), dates(X, Y_2), Y_1 \neq Y_2 \\ cheats\_j(X) &\leftarrow marr(X, Y), friend(Y, jane), dates(X, jane), Y \neq jane \\ &\leftarrow cheats(X), dates(X, Y), not\ marr(X, Y), not\ cheats(Y) \end{aligned}$$

where *marr*, *friend*, and *dates* are free predicates.[13] An (infinite) open answer set of this program that satisfies *cheats_j* is

$$\begin{aligned} M = \{&cheats(x), cheats\_j(x), dates(x, jane), \\ &marr(x, x1), friend(x1, jane), \\ &cheats(jane), marr(jane, jane1), dates(jane, jane2), \\ &cheats(jane2), marr(jane2, jane21), dates(jane2, jane22), \\ &cheats(jane22), \ldots\} \ , \end{aligned}$$

depicted in Fig. 4.6.

One sees that $x$ cheats his spouse with Jane since $x$ dates Jane but is married to $x1$. Furthermore, by the constraint, we must have that Jane is also a cheater, and thus, by minimality of open answer sets, Jane is married to some *jane*1 and dates *jane*2, who in turn must be cheating, resulting in an infinite answer set.

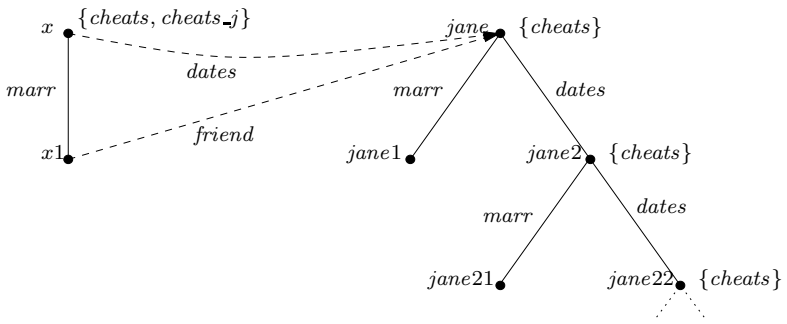


**Fig. 4.6.** Forest Model

In many cases, there is interesting knowledge that cannot be captured within the rather strict tree format of FoLP rules. For example, in addition,

[13] Note that the second rule is, strictly speaking, not a FoLP rule. However, we can easily rewrite it as two FoLP rules.

we may have a rule representing that if Leo is married to Jane, Jane is dating Felix, and Leo himself is not cheating, then Leo dislikes Felix:

$$dislikes(leo, felix) \leftarrow marr(leo, jane), dates(jane, felix), not\ \ cheats(leo)$$

This ground rule does not have a tree structure, but relates the three constants in an arbitrary graph-like manner. We extend FoLPs by allowing for a component with arbitrary program rules that may only be grounded with the combined program's constants.

**Definition 4.29.** *An* extended forest logic program (EFoLP) $P$ *is a pair* $(Q, R)$ *where* $Q$ *is a FoLP and* $R$ *is a finite program where predicates are unary or binary. We denote* $Q$ *with* $folp(P)$ *and* $R$ *with* $e(P)$*. An* EFoLP answer set *of* $(Q, R)$ *is an open answer set of* $Q \cup R_{cts(Q \cup R)}$*. Satisfiability checking, consistency checking, and query answering w.r.t. EFoLPs are modified accordingly.*

We will often speak of open answer sets of an EFoLP $(Q, R)$ instead of an EFoLP answer set. Additionally, we may also call a program $P$ an EFoLP if $P$ can be written as $Q \cup R$ with $Q$ a FoLP and $R$ a finite *ground* program with unary and/or binary predicates.

Note that $e(P)$ can be a full-fledged program, i.e., with negation as failure. Moreover, predicates in $e(P)$ may be defined (i.e., appear in the head of rules) in the FoLP $folp(P)$, as is the case for *marr*, *dates* and *cheats*. Vice versa, we may have predicates appearing in the FoLP part that are defined in the ground rule part, e.g., *dislikes* could appear in the FoLP part as a $dislikes(X, Y)$ literal.

Naively, one could try to reduce reasoning with an EFoLP $(Q, R)$ to FoLP reasoning by first calculating an answer set of the ground program part $R_{cts(Q \cup R)}$ and then replacing the part by the facts induced by this answer set, resulting in a FoLP. However, this would be wrong due to the influence the FoLP part plays in the ground part. E.g., the empty set is the only answer set of the above *dislikes* rule, and thus one would never have that somebody dislikes someone, which is clearly not true in combination with the FoLP from the cheating example since it provides definitions for the body predicates of the rule.

EFoLPs still have the forest model property, since, intuitively, graph-like connections between constants are allowed in a forest, which is all the $e(P)$ part of an EFoLP $P$ can ever introduce.

**Theorem 4.30.** *Extended forest logic programs have the forest model property.*

*Proof.* Take an EFoLP $P = (Q, R)$, where $Q$ is a FoLP and $R$ is an arbitrary program. Let $p \in upreds(P)$ s.t. $p$ is satisfiable, i.e., there exists an open answer set $(U, M)$ of $P$ with $p(u) \in M$. Let $n$ be the rank of $P$, i.e., the rank of $Q$ (we discard $R$ in calculating the rank of $P$ as, semantically, $R$ is identified with the ground $R_{cts(Q \cup R)}$, of which rules can be considered non-live).

We then construct the $\theta_x$ as in the proof of Theorem 4.7 (pp. 120). If the selected $r$ is in $R$, we treat it as if it were non-live. The rest of the proof is entirely analogous to the proof of Theorem 4.7. □

The forest model of the cheats example is depicted in Fig. 4.7. The cutting of

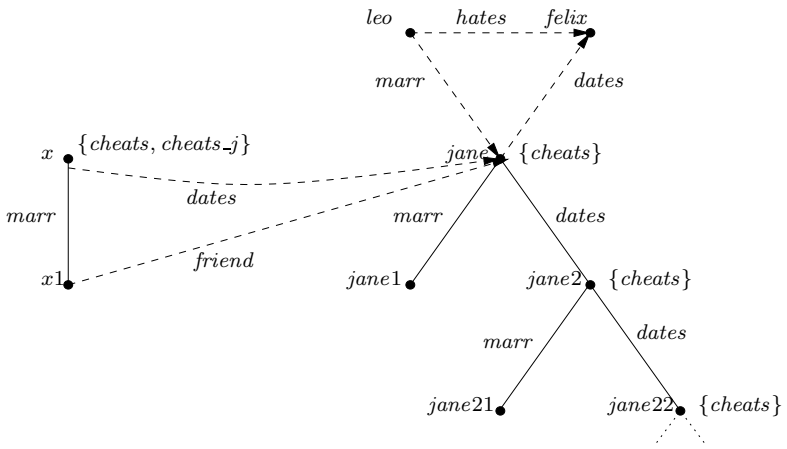


**Fig. 4.7.** Forest Model of the EFoLP

infinite open answer sets to finite structures, as described in Section 4.2, can again not be applied to arbitrary EFoLPs since the finite structures would not necessarily be answer sets. We define *local EFoLPs* as consisting of a local FoLP and an arbitrary program with unary or binary predicates.

**Definition 4.31.** *A* local EFoLP $P$ *is an EFoLP where* $folp(P)$ *is a local FoLP.*

Local EFoLPs then have the desired bounded finite model property.

**Theorem 4.32.** *Let $P$ be a local EFoLP. Then, $P$ has the bounded finite model property.*

*Proof.* Let $P = (Q, R)$ be an EFoLP. The proof is along the lines of the proof of Theorem 4.32 where $k$ is again at most

$$(c+1)\sum_{i=0}^{2^u+1} n^i\ , \tag{4.7}$$

with $c$ the number of constants in $Q \cup R$, $u$ the number of unary predicates in $Q \cup R$ and $n$ the rank of $Q$.

Note that constants are always in the cut open answer set $(U', M')$ (higher up in the trees than either $\overline{z}_{\mathcal{P}}$ or $z_{\mathcal{P}}$) such that the ground part $e(P)$ does not yield any difficulties. □

Thanks to this property we can reduce reasoning with EFoLPs to normal answer set programming by introducing a sufficiently large number of new constants $x_i$.

**Theorem 4.33.** *Let $P$ be a local EFoLP $(Q, R)$. Then, $p \in upreds(P)$ is satisfiable w.r.t. $P$ iff there is a $0 \leq h \leq k$ and an answer set $M$ of $\psi_h(Q \cup R_{cts(Q \cup R)})$, where $k$ is as in Equation (4.7) and $\psi_h$ is as in Theorem 4.17.*

*Proof.* The proof is analogous to the proof of Theorem 4.17. □

Theorem 4.33 allows to reduce satisfiability checking w.r.t. local EFoLPs to normal finite answer set programming. The opposite direction holds as well for programs with unary and binary predicates only.

**Theorem 4.34.** *Let $P$ be a program with unary and binary predicates only. Then, there is an answer set $M$ of a program $P$ containing a $p(a)$ with $a \in cts(P)$ iff $p$ is satisfiable w.r.t. to the local EFoLP $(\emptyset, P)$*[14]*.*

*Proof.* The pair $(\emptyset, P)$ is indeed a local EFoLP. Furthermore, $M$ is an answer set of $P$ iff $M$ is an answer set of $P_{cts(P)}$ iff $(cts(P), M)$ is an open answer set of $P_{cts(P)}$ iff $(cts(P), M)$ is an open answer set of $(\emptyset, P)$. This proves the "only if" direction.

For the "if" direction, take an open answer set $(U, M)$ of $P_{cts(P)}$, then $M$ is an answer set of $(P_{cts(P)})_U = P_{cts(P)}$. □

**Theorem 4.35.** *Satisfiability checking w.r.t. local EFoLPs is in* 2-EXPTIME$^{\text{NEXPTIME}}$.

*Proof.* Let $P = (Q, R)$ be a local EFoLP. Checking whether there exists an answer set $M$ of $\psi_h(P')$ for some $0 \leq h \leq k$ with $P' \equiv Q \cup R_{cts(Q \cup R)}$ and where $k$ and $\psi_h(P')$ are as in Theorem 4.33, amounts to checking whether there exists an answer set $M$ of $gr(\psi_h(P'))$. By [DEGV01, Bar03] and the disjunction-freeness of the GL-reduct of $gr(\psi_h(P'))$ we have that the latter can be decided by a non-deterministic Turing Machine in time polynomial in the size of $gr(\psi_h(P'))$. In determining the size of $gr(\psi_h(P'))$, one sees that the size of $gr(\psi_h(P'))$ is exponential in the size of $Q$ if the degree of $Q$ is unbounded and polynomial in the size of $Q$ if the degree of $Q$ is bounded. Moreover, the size of $gr(\psi_h(P'))$ is exponential in the size of $R$. Thus the size of $gr(\psi_h(P'))$ is (at most) exponential in the size of $P$ and, for a fixed $h$, checking whether $\psi_h(P')$ has an answer set is in NEXPTIME.

Satisfiability checking of a predicate w.r.t. $P = (Q, R)$ can then again be done by starting with $h = 0$ and checking whether $\psi_h(P')$ has an answer set. If this is the case, we are done (by Theorem 4.33), otherwise, we repeat the check for $h = 1$, and so on. This procedure involves at most $k + 1$ calls to an NEXPTIME oracle.

Since $k$ is double exponential in the size of $P$ the above procedure to check satisfiability runs in 2-EXPTIME$^{\text{NEXPTIME}}$. □

[14] Or equivalently, the local EFoLP $\emptyset \cup P_{cts(P)}$.

We define *semi-local* EFoLPs and *acyclic* EFoLPs for which satisfiability checking can be reduced to satisfiability checking w.r.t. local EFoLPs.

**Definition 4.36.** *An EFoLP $P = (Q, R)$ is* semi-local *if $Q$ is semi-local.*

One can translate a semi-local EFoLP $(Q, R)$ to a local EFoLP $(\phi(Q), R)$ where $\phi$ is defined as on pp. 124.

**Theorem 4.37.** *Let $P = (Q, R)$ be a semi-local EFoLP and $p \in upreds(P)$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. the local EFoLP $(\phi(Q), R)$. Furthermore, the size of $(\phi(Q), R)$ is linear in the size of $P$.*

*Proof.* The proof is analogous to the proof of Theorem 4.15. □

Complexity upper bounds for semi-local EFoLPs can then be obtained from the upper bounds for local EFoLPs.

**Theorem 4.38.** *Satisfiability checking w.r.t. semi-local EFoLPs is in* 2-EXPTIME$^{\text{NEXPTIME}}$.

A similar extension of acyclic FoLPs to the EFoLP case does not work, i.e., an EFoLP $(Q, R)$ where $Q$ is an acyclic FoLP can not be equivalently rewritten as the local EFoLP $(\kappa(Q), R)$, where $\kappa$ is as on pp. 133.

*Example 4.39.* Take the EFoLP $(Q, R)$ with $Q$ the rule $p(X) \leftarrow q(X)$ and $R$ the rule $q(a) \leftarrow p(a)$. Then $Q$ is acyclic and $\kappa(Q)$ is

$$p(X) \leftarrow not\ q'(X)$$
$$q'(X) \leftarrow not\ q(X)$$

Then, $(\{a\}, \{p(a), q(a)\})$ is an open answer set of the EFoLP $(\kappa(Q), R)$ but the only open answer set of $(Q, R)$ with universe $\{a\}$ is $(\{a\}, \emptyset)$.

If $Q \cup R$ were acyclic then $(Q, R)$ would be equivalent to $(\kappa(Q), \kappa(R))$.

**Theorem 4.40.** *Let $P = (Q, R)$ be an EFoLP such that $Q \cup R$ is acyclic and $p \in upreds(P)$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. the local EFoLP $(\kappa(Q), \kappa(R))$.*

*Proof.* We have that $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. $Q \cup R_{cts(Q \cup R)}$. Since the latter is acyclic, we have, with Theorem 4.24, that $p$ is satisfiable w.r.t. $Q \cup R_{cts(Q \cup R)}$ iff $p$ is satisfiable w.r.t. $\kappa(Q \cup R_{cts(Q \cup R)}) = \kappa(Q) \cup \kappa(R)_{cts(Q \cup R)}$ iff $p$ is satisfiable w.r.t. $(\kappa(Q), \kappa(R))$. Since $Q$ is an acyclic FoLP ($Q \cup R$ is acyclic), $\kappa(Q)$ is a local FoLP (Theorem 4.25) and thus $(\kappa(Q), \kappa(R))$ is a local EFoLP. □

**Theorem 4.41.** *Satisfiability checking w.r.t. EFoLPs $(Q, R)$ where $Q \cup R$ is acyclic is in* 2-EXPTIME$^{\text{NEXPTIME}}$.

*Proof.* By the reduction in Theorem 4.40 and Theorem 4.35. □

Another class of EFoLPs, one that will prove useful in Chapter 6, are the *free acyclic* EFoLPs.

**Definition 4.42.** *An EFoLP $P = (Q, R)$ is* free acyclic *if $Q$ is acyclic and*

$$\forall \alpha \leftarrow \beta \in R, \alpha^+ = \{q(\mathbf{s})\} \cdot q(\mathbf{X}) \vee \mathit{not}\ q(\mathbf{X}) \leftarrow\in Q \cup R\ .$$

*where $\mathbf{X} = X$ if $q$ is unary and $\mathbf{X} = (X, Y)$ if $q$ is binary.*

An EFoLP is thus free acyclic if its FoLP part is acyclic and for each positive atom in a head of a rule in $R$ there is a free rule. One can then safely replace unary atoms in $Q$ by their double negation.

**Theorem 4.43.** *Let $P = (Q, R)$ be a free acyclic EFoLP and $p \in \mathit{upreds}(P)$. Then, $p$ is satisfiable w.r.t. $P$ iff $p$ is satisfiable w.r.t. the local EFoLP $(\kappa(Q), R)$.*

*Proof.* For the "only if" direction, assume $p$ is satisfiable w.r.t. $P$, i.e., there is an open answer set $(U, M)$ of $Q \cup R_{cts(Q \cup R)}$ such that $p(y) \in M$. One can show, along the lines of the proof of Theorem 4.24, that $(U, M')$ with $M' \equiv M \cup \{b'(x) \mid b(x) \notin M, b' \in \kappa(P)\}$ is an open answer set of $(\kappa(Q), R)$.

For the "if" direction, assume $p$ is satisfiable w.r.t. $(\kappa(Q), R)$, i.e., there is an open answer set $(U, M)$ of $\kappa(Q) \cup R_{cts(Q \cup R)}$ such that $p(y) \in M$. Define $M' \equiv M \backslash \{b'(x)\}$, then $(U, M')$ is an open answer set of $(Q, R)$ and $p(y) \in M'$.

- $M'$ is a model of $(Q \cup R_{cts(Q \cup R)})_U^{M'}$. This is again along the lines of the proof of Theorem 4.15.
- $M'$ is a minimal model of $(Q \cup R_{cts(Q \cup R)})_U^{M'}$. Assume not, then there is a model $N' \subset M'$ of $(Q \cup R_{cts(Q \cup R)})_U^{M'}$.
  We prove that $M' \subseteq N'$. Take $l \in M'$, we prove that $l \in N'$ by induction on the maximum depth[15] of $PDG(Q)$ of the predicate $\mathit{preds}(l)$ in $l$, which is possible since $Q$ is acyclic and $PDG(Q)$ is finite[16].
  - $\mathit{preds}(l)$ has depth 0 in $PDG(Q)$. Then, all rules $\alpha \leftarrow \beta \in Q$ with $\mathit{preds}(l)$ in $\alpha^+$ are such that $\beta$ is a set of of naf-atoms and/or equality atoms. Consequently such $\alpha \leftarrow \beta \in \kappa(Q)$ and it does not contain any newly added $b'(s)$'s.
    Since $l \in M$ we have that there is a $l \leftarrow \beta^+[] \in (\kappa(Q) \cup R_{cts(Q \cup R)})_U^M$ with $M \models \beta^+[]$ and $M \models \alpha^-[] \cup \mathit{not}\ \beta^-[]$ originating from $\alpha[] \leftarrow \beta[] \in \kappa(Q)_U$ or $\alpha[] \leftarrow \beta[] \in R_{cts(Q \cup R)}$.
    In the former case, $\emptyset \models \beta^+[]$[17], $\alpha[] \leftarrow \beta[] \in Q_U$ and $M' \models \alpha^-[] \cup \mathit{not}\ \beta^-[]$ such that $l \leftarrow \beta^+[] \in Q_U^{M'}$ with $\emptyset \models \beta^+[]$. Since $N'$ is a model of $Q_U^{M'}$, $l \in N'$.

[15] A predicate $p$ has depth 0 if it has no successors in $PDG(Q)$ and depth $n$ if the maximum depth of its successors in $PDG(Q)$ is $n - 1$.

[16] We assume that if $l \notin \mathit{preds}(Q)$, then the depth of $l$ is 0.

[17] $\beta^+$ contains, if anything, only equality atoms since the depth of $\mathit{preds}(l)$ is 0.

In the latter case, there is a free rule $L \vee not\ L \leftarrow\in Q \cup R$ such that $L[] = l$ and thus $l \vee not\ l \leftarrow\in (Q \cup R_{cts(Q \cup R)})_U$. Since $l \in M'$, we have that $l \leftarrow\in (Q \cup R_{cts(Q \cup R)})_U^{M'}$ and $l \in N'$.

- Assume it is proved for literals $l$ with depth of $preds(l)$ at most $n-1$ (IH).
- Take $l$ with depth of $preds(l)$ at most $n$. Then all rules $\alpha \leftarrow \beta, \gamma$ in $Q$ with $preds(l)$ in $\alpha^+$, where $\beta$ is a set of unary atoms, and $\gamma$ the rest, are such that the predicates in $\beta$ and $\gamma^+$ have a depth of at most $n-1$. We have that such $\alpha \leftarrow not\ \beta', \gamma \in \kappa(Q)$ and $b'(X) \leftarrow not\ b(X) \in \kappa(Q)$ for $b \in preds(\beta)$.
  Since $l \in M$, we have that there is a $l \leftarrow \gamma^+[] \in (\kappa(Q) \cup R_{cts(Q \cup R)})_U^M$ with $M \models \gamma^+[]$, and $l \leftarrow \gamma^+[] \in \kappa(Q)_U^M$ or $l \leftarrow \gamma^+[] \in R^M_{cts(Q \cup R)}$.
  In the former case, $l \leftarrow \gamma^+[]$ originates from a $\alpha \leftarrow not\ \beta', \gamma \in \kappa(Q)$ and thus $M \models \alpha^-[] \cup not\ \gamma^-[] \cup not\ \beta'[]$.
  We have that $l \leftarrow \beta[], \gamma^+[] \in (Q \cup R_{cts(Q \cup R)})_U^{M'}$ with the body true in $N'$. Indeed, we have that $\alpha \leftarrow \beta, \gamma \in Q$, and $M' \models \alpha^-[] \cup not\ \gamma^-[]$. Furthermore, $N' \models \beta[]$ and $N' \models \gamma^+[]$, by the fact that $M \models \beta[]$ and $M \models \gamma^+[]$ and the induction hypothesis. The former can be seen by noting that $M \models not\ \beta'[]$. Take then a $b(x) \in \beta[]$, then $b'(x) \in \beta'[]$ such that $b'(x) \notin M$ and thus, by $b'(X) \leftarrow not\ b(X) \in \kappa(Q)$, we have that $b(x) \in M$. Since $N'$ is a model of $(Q \cup R_{cts(Q \cup R)})_U^{M'}$, we then have that $l \in N'$.
  In the latter case, $l \leftarrow \gamma^+[] \in R^M_{cts(Q \cup R)}$, there is a free rule $L \vee not\ L \leftarrow\in Q \cup R$ such that $L[] = l$ and thus $l \vee not\ l \leftarrow\in (Q \cup R_{cts(Q \cup R)})_U$. Since $l \in M'$, we have that $l \leftarrow\in (Q \cup R_{cts(Q \cup R)})_U^{M'}$ and $l \in N'$.

Thus $M' \subseteq N'$, a contradiction with $N' \subset M'$.

□

**Theorem 4.44.** *Satisfiability checking w.r.t. free acyclic EFoLPs is in* 2-EXPTIME$^{\text{NEXPTIME}}$.

*Proof.* By the reduction in Theorem 4.43 and Theorem 4.35. □

# 5

# Guarded Open Answer Set Programming

In Section 5.1, we reduce satisfiability checking w.r.t. arbitrary logic programs to satisfiability checking of alternation-free fixed point logic formulas. We identify in Section 5.2 syntactical classes of programs for which this FPL translation falls into the decidable logic $\mu$GF or $\mu$LGF, i.e., guarded or loosely guarded fixed point logic.

In Section 5.3, we introduce so-called generalized literals and modify the translation to FPL in Section 5.4. Section 5.5 mirrors Section 5.2 and identifies classes of programs with generalized literals that can be mapped to guarded FPL. Finally, in Section 5.6, we relate the obtained languages under the open answer set semantics to Datalog LITE which has a least fixed point model semantics.

## 5.1 Open Answer Set Programming via Fixed Point Logic

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 5.1.** *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 $in(Y) \equiv \cup\{Y \neq a \mid a \in preds(P) \cup \{0\}\}$, i.e., a set of inequalities between the variable $Y$ and the predicates in $P$ as well as a new constant 0. For a sequence of variables $\mathbf{Y}$, 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 5.2.* 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 5.3.** *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 $p(\mathbf{x}, \mathbf{0}, q) \in M'$.*

*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 $U' = U \cup preds(P) \cup \{0\}$ 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 $U \equiv U' \backslash (preds(P) \cup \{0\})$ and $M \equiv \{q(\mathbf{x}) \mid p(\mathbf{x}, \mathbf{0}, q) \in M' \wedge \mathbf{x} \cap (preds(P) \cup \{0\}) = \emptyset\}$.

By Theorem 3.11 (pp. 65), we can assume that $q$ is a non-free predicate (and we assume this throughout the rest of the chapter). 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 $(P_p)^{M'}_{U'}$ such that $M' \models in(\mathbf{Y})[]$ for $\mathbf{Y}$ the variables in the corresponding ungrounded rule $r$. Thus $\mathbf{x} \cap (preds(P) \cup \{0\}) = \emptyset$, such that $q(\mathbf{x}) \in M$, by definition of $M$.

Remains to show that $(U, M)$ is an open answer set of $P$.

- $M$ is a model of $P^M_U$. This can be easily done.
- $M$ is a minimal model of $P^M_U$. Assume not, then there is a model $N \subset M$ of $P^M_U$. Define

$$N' \equiv \{p(\mathbf{x}, \mathbf{0}, q) \mid q(\mathbf{x}) \in N\} \\ \cup \{p(\mathbf{x}, \mathbf{0}, q) \mid p(\mathbf{x}, \mathbf{0}, q) \in M' \wedge \mathbf{x} \cap (\mathit{preds}(P) \cup \{0\}) \neq \emptyset\} .$$

Clearly $N' \subset M'$; one can show that $N'$ is a model of $(P_p)^{M'}_{U'}$, which leads to a contradiction with the minimality of $M'$.

□

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

**Theorem 5.4.** *Let $P$ be a program and $p$ a predicate not in $P$. The size 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 5.3 and 5.4, 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 5.8.

In [CH82], a similar motivation drives the reduction of Horn clauses[1] 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, [CH82] needs an additional transformation that unfolds bodies of clauses.

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

$$a \neq b \tag{5.1}$$

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

$$\exists X \cdot \mathbf{true} . \tag{5.2}$$

Besides these technical requirements matching FOL interpretations with open interpretations, $\mathtt{comp}(P)$ contains the formulas in $\mathtt{fix}(P) \equiv \mathtt{sat}(P) \cup \mathtt{gl}(P) \cup \mathtt{fpf}(P)$, which can be intuitively categorized as follows:

- $\mathtt{sat}(P)$ ensures that a model of $\mathtt{fix}(P)$ satisfies all rules in $P$,

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

- $\texttt{gl}(P)$ 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 5.5.** *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*

$$\forall \mathbf{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{5.3}$$

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

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

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg \beta^- \tag{5.4}$$

*for rules $r : \alpha \leftarrow \beta \in P$[2] with variables $\mathbf{Y}$,*

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

$$\forall \mathbf{X} \cdot p(\mathbf{X}) \Rightarrow [\text{LFP } \ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{X}) \tag{5.5}$$

*with*

$$\phi(W, \mathbf{X}) \equiv W(\mathbf{X}) \vee \bigvee_{r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P} E(r) \tag{5.6}$$

*and*

$$E(r) \equiv \exists \mathbf{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p|W] \wedge r(\mathbf{Y}) \tag{5.7}$$

*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 $\beta^+$ with $p$ replaced by $W$.*

*The* completion *of $P$ is $\texttt{comp}(P) \equiv \texttt{fix}(P) \cup \{(5.1), (5.2)\}$.*

The predicate $W$ appears only positively in $\phi(W, \mathbf{X})$ such that the fixed point formula in (5.5) is well-defined. By the first disjunct in (5.6), we have that applying the operator $\phi^{(U,M)}$ (see pp. 57) to an arbitrary set $S \subseteq U^n$ does not lose information from $S$.

**Theorem 5.6.** *Let $P$ be a p-program and $(U, M)$ an interpretation with $S \subseteq U^n$. Then*

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

[2] We assume that rules are uniquely named.

*Proof.* Take $\mathbf{x} \in S$, then $(U, M), W \rightarrow S \models W(\mathbf{x})$, such that, by (5.6), $(U, M), W \rightarrow S \models \phi(W, \mathbf{x})$. Thus, by (2.2), we have that $\mathbf{x} \in \phi^{(U,M)}(S)$. □

*Example 5.7.* 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 5.8.** *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*

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

*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 $\texttt{comp}(P)$.

- $(U, M')$ **is a model of** (5.1). Immediate, since the domain $U$ of the FOL interpretation is also the universe of the open interpretation.
- $(U, M')$ **is a model of** (5.2). Immediate, $U$ is non-empty by the definition of universes.
- $(U, M')$ **is a model of** $\texttt{sat}(P)$. Take $y_1, \ldots, y_d \in U$ s.t. $\bigwedge \beta[Y_1|y_1, \ldots, Y_d|y_d]$ is true in $(U, M')$. Ground $\beta$ in the rule $\alpha \leftarrow \beta$ accordingly; we have that $M \models \mathit{not}\ \beta^-[]$. If $M \not\models \alpha^-[]$, there is a $\mathit{not}\ l \in \alpha[]$ such that $M \models \mathit{not}\ l$, and thus $\bigvee \alpha[]$ is true in $(U, M')$.
  If $M \models \alpha^-[]$, we have that $\alpha^+[] \leftarrow \beta^+[] \in P_U^M$ and thus there exists a $l \in \alpha^+[]$ with $M \models l$ such that $\bigvee \alpha[]$ is true in $(U, M')$.
- $(U, M')$ **is a model of** $\texttt{gl}(P)$. Take $\mathbf{y} = y_1, \ldots, y_d$ and assume $r(\mathbf{y})$ is in $M'$. By definition of $M'$, we have that $r(\mathbf{y}) \in R$, and thus $\mathit{vars}(r) = \mathbf{Y}, M \models \alpha^-[\mathbf{Y}|\mathbf{y}], M \models \mathit{not}\ \beta^-[\mathbf{Y}|\mathbf{y}]$ for $r : \alpha \leftarrow \beta$, such that $M' \models \bigwedge \alpha^-[]$ and $M' \models \bigwedge \neg\beta^-[]$.
  For the other direction, take $\mathbf{y} = y_1, \ldots, y_d$ and assume $\bigwedge \alpha^-[\mathbf{Y}|\mathbf{y}]$ and $\bigwedge \neg\beta^-[\mathbf{Y}|\mathbf{y}]$ are true in $M'$, consequently, immediately by the definition of $M'$, $r(\mathbf{y}) \in R \subseteq M'$.

$(U, M')$ **is a model of** $\texttt{fpf}(P)$. 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.* $\mathbf{x} \in \phi^{(U,M')} \uparrow n$, $n < \infty$.

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

- $\emptyset \models \beta^+[]$[3],
- $M \models \alpha^-[]$, and
- $M \models not\ \beta^-[]$.

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 $(U, M'), W \rightarrow \emptyset \models \phi(W, \mathbf{x})$, 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
- $M \models not\ \beta^-[]$.

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 $\beta$ 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 $(U, M'), W \rightarrow \phi^{(U,M')} \uparrow n-1 \models E(r)[\mathbf{X}|\mathbf{x}]$, such that $(U, M'), W \rightarrow \phi^{(U,M')} \uparrow n-1 \models \phi(W, \mathbf{x})$. 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 $\alpha$ 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 (5.5) is satisfied.

$\boxed{\Leftarrow}$ For the "if" direction, assume $(U, M')$ is a model of $\texttt{comp}(P)$. 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 (5.1) and (5.2), 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. $\overline{M} = \mathrm{LFP}(\phi^{(U,M')})$.

[3] $\beta^+$ may contain equalities but no regular atoms.

- $\overline{M} = \phi^{(U,M')}(\overline{M})$.
  - $\overline{M} \subseteq \phi^{(U,M')}(\overline{M})$. Immediate, with Theorem 5.6.
  - $\overline{M} \supseteq \phi^{(U,M')}(\overline{M})$. Assume $\mathbf{x} \in \phi^{(U,M')}(\overline{M})$. Then by (2.2), we have that $(U, M'), W \to \overline{M} \models \phi(W, \mathbf{x})$. Thus, by (5.6), 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 $\mathbf{x} \in \overline{M}$.
- $\overline{M}$ 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 $\overline{N} = \{\mathbf{x} \mid p(\mathbf{x}) \in N\}$, we show that $\overline{N}$ is a fixed point of $\phi^{(U,M')}$, i.e., $\overline{N} = \phi^{(U,M')}(\overline{N})$.
   - $\overline{N} \subseteq \phi^{(U,M')}(\overline{N})$. Immediate, with Theorem 5.6.
   - $\overline{N} \supseteq \phi^{(U,M')}(\overline{N})$. Assume $\mathbf{x} \in \phi^{(U,M')}(\overline{N})$. Then by (2.2), we have that $(U, M'), W \to \overline{N} \models \phi(W, \mathbf{x})$. Thus, by (5.6), we have either that $\mathbf{x} \in \overline{N}$, which means we are done, or there is a $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ such that $(U, M'), W \to \overline{N} \models E(r)[\mathbf{X}|\mathbf{x}]$.
     Then, there exist $[\mathbf{Y}|\mathbf{y}]$ with
     - $\mathbf{x} = \mathbf{t}[]$,
     - $(U, M'), W \to \overline{N} \models \beta^+[p|W][]$, such that $N \models \beta^+[]$,
     - $r(\mathbf{y}) \in M'$, from which, since $M'$ is a model of $\mathtt{gl}(P)$, we have that $M' \models \bigwedge \alpha^-[]$ and thus $M' \models \bigwedge \neg\beta^-[]$, and thus $M \models \alpha^-[]$ and $M \models \mathit{not}\ \beta^-[]$.

     Thus $p(\mathbf{x}) \leftarrow \beta^+[] \in P_U^M$ with the body true in $N$, such that, since $N$ is a model of $P_U^M$, we have that $p(\mathbf{x}) \in N$, and $\mathbf{x} \in \overline{N}$.

Thus $\overline{N}$ is a fixed point of $\phi^{(U,M')}$. 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 5.9.* For a universe $U = \{x\}$ we have the unique open answer set $(U, \emptyset)$ of $P$ in Example 5.7. 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 5.5 differs from Clark's completion [Cla87] both in the presence of the fixed point construct in (5.5) 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 5.7 is the open ASP version of the classical example $p \leftarrow p$ [LL03]. 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 $P$.

*Example 5.10.* Take the program $P$

$$r_1 : p(X, a) \leftarrow \mathit{not}\ p(X, b), X \neq a, X \neq b$$
$$r_2 : p(X, b) \leftarrow \mathit{not}\ p(X, a), X \neq a, X \neq b$$

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

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

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 $\mathrm{LFP}(\phi^{(U,M_3')}) = \emptyset$, such that $\mathtt{fpf}(P)$ is not satisfied by $M_3'$. Furthermore, $\mathrm{LFP}(\phi^{(U,M_1')}) = \{(x,a)\}$ and $\mathrm{LFP}(\phi^{(U,M_2')}) = \{(x,b)\}$. 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 $P$.

**Theorem 5.11.** *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 (5.1) 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 5.12.** *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 $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{comp}(P_p)$ is satisfiable. Moreover, this reduction is polynomial in the size of $P$.*

*Proof.* Assume $q$ is satisfiable w.r.t. $P$. By Theorem 5.3, we have that $p(\mathbf{x}, \mathbf{0}, q)$ is in an open answer set of $P_p$, such that, with Theorem 5.8, $p(\mathbf{x}, \mathbf{0}, q)$ 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 5.8, such that $(U, M)$ is an open answer set of $P_p$ and $p(\mathbf{x}, \mathbf{0}, q) \in M$. From Theorem 5.3, we then have that $q$ is satisfiable w.r.t. $P$.

By Theorem 5.11, 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 5.4, the size of $\bigwedge \mathtt{comp}(P_p)$ is polynomial in the size of $P$. □

## 5.2 Guarded Open Answer Set Programming

We repeat the definitions of the *loosely guarded fragment* [Ben97] of first-order logic as in [GW99]: *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* $\psi(\mathbf{X},\mathbf{Y})$[4] *is in LGF, and* $\alpha(\mathbf{X},\mathbf{Y}) = \alpha_1 \wedge \ldots \wedge \alpha_m$ *is a conjunction of atoms, then the formulas*

$$\exists \mathbf{Y} \cdot \alpha(\mathbf{X},\mathbf{Y}) \wedge \psi(\mathbf{X},\mathbf{Y})$$
$$\forall \mathbf{Y} \cdot \alpha(\mathbf{X},\mathbf{Y}) \Rightarrow \psi(\mathbf{X},\mathbf{Y})$$

*belong to LGF (and* $\alpha(\mathbf{X},\mathbf{Y})$ *is the* guard *of the formula), provided that* $free(\psi) \subseteq free(\alpha) = \mathbf{X} \cup \mathbf{Y}$ *and for every quantified variable* $Y \in \mathbf{Y}$ *and every variable* $Z \in \mathbf{X} \cup \mathbf{Y}$ *there is at least one atom* $\alpha_j$ *that contains both* $Y$ *and* $Z$ *(where* $free(\psi)$ *are the free variables of* $\psi$*).*

The *loosely guarded fixed point logic* $\mu$LGF is LGF extended with fixed point formulas (2.1) where $\psi(W,\mathbf{X})$ is a $\mu$LGF 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* $\mu$GF is GF extended with fixed point formulas where $\psi(W,\mathbf{X})$ is a $\mu$GF formula such that $W$ does not appear in guards.

*Example 5.13.* The infinity axiom in Example 2.27 (pp. 58) is a $\mu$GF formula where all the formulas are guarded by $F(X,Y)$.

*Example 5.14 ([GW99]).* Take the formula

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

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 5.15.** *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* $\gamma_b$*; we call* $\gamma_b$ *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* $\gamma_h$*;* $\gamma_h$ *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 5.16.* The rule in Example 5.7 is loosely guarded but not fully loosely guarded. The program in Example 5.10 is neither fully loosely guarded nor loosely guarded. A rule

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

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

[4] Recall that $\psi(\mathbf{X},\mathbf{Y})$ denotes a formula whose free variables are all among $\mathbf{X} \cup \mathbf{Y}$ ([ANB98], pp. 236).

**Definition 5.17.** *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 [GHO02] 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 5.18.* 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 $P^{\mathrm{f}}$, where $P^{\mathrm{f}}$ 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 $P^{\mathrm{f}}$ is indeed a fully (loosely) guarded program.

**Theorem 5.19.** *Let $P$ a (L)GP. Then, $P^{\mathrm{f}}$ 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 $\gamma_b$ is a head guard of $r$ and $r$ is fully (loosely) guarded. □

A rule is vacuously satisfied if the body of a rule in $P^{\mathrm{f}}$ 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 $P^{\mathrm{f}}$ reduces to its corresponding rule in $P$.

**Theorem 5.20.** *Let $P$ be a program. An open interpretation $(U, M)$ of $P$ is an open answer set of $P$ iff $(U, M)$ is an open answer set of $P^{\mathrm{f}}$.*

*Proof.* For the "only if" direction, assume $(U, M)$ is an open answer set of $P$.

- $M$ is a model of $(P^{\mathrm{f}})^M_U$. Take a rule $(\alpha \cup \mathit{not}\ \beta^+)[]^+ \leftarrow \beta[]^+ \in (P^{\mathrm{f}})^M_U$ with
  - $M \models (\alpha \cup \mathit{not}\ \beta^+)[]^-$,
  - $M \models \mathit{not}\ \beta[]^-$,

  originating from $\alpha \cup \mathit{not}\ \beta^+ \leftarrow \beta \in P^{\mathrm{f}}$ with $\alpha \leftarrow \beta \in P$. Furthermore,
  - $M \models \alpha[]^-$,
  - $M \models \mathit{not}\ \beta[]^-$.

  Thus $\alpha[]^+ \leftarrow \beta[]^+ \in P^M_U$. Take $M \models \beta[]^+$, then $\exists l \in \alpha^+[] \cdot M \models l$, and thus $l \in (\alpha \cup \mathit{not}\ \beta^+)[]^+$.

- $M$ is a minimal model of $(P^{\mathrm{f}})^M_U$. Assume not, then there is a model $N \subset M$ of $(P^{\mathrm{f}})^M_U$. We show that $N$ is a model of $P^M_U$, which leads to a contradiction with the minimality of $M$. Take a rule $\alpha[]^+ \leftarrow \beta[]^+ \in P^M_U$ with
  - $M \models \alpha[]^-$,
  - $M \models not\ \beta[]^-$.

  originating from $\alpha \leftarrow \beta \in P$. Take $N \models \beta[]^+$ (then $M \models \beta[]^+$). For the corresponding $\alpha \cup not\ \beta^+ \leftarrow \beta$ in $P^{\mathrm{f}}$, we have that
  - $M \models (\alpha \cup not\ \beta^+)[]^-$. Indeed, $M \models \alpha[]^-$ and $M \models (not\ \beta^+[])^- = \beta^+$.
  - $M \models not\ \beta[]^-$,

  such that $(\alpha \cup not\ \beta^+)[]^+ \leftarrow \beta[]^+ \in (P^{\mathrm{f}})^M_U$. Since $N$ is a model we have that $\exists l \in (\alpha \cup not\ \beta^+)[]^+ = \alpha[]^+ \cdot N \models l$.

  For the "if" direction, assume $(U, M)$ is an open answer set of $P^{\mathrm{f}}$.

- $M$ is a model of $P^M_U$. This can be done similarly as the above case where $N$ was shown to be a model.
- $M$ is a minimal model of $P^M_U$. Assume not, then there is a model $N \subset M$ of $P^M_U$. One can again show that $N$ is a model of $(P^{\mathrm{f}})^M_U$, which leads to a contradiction with the minimality of $M$.

□

Since we only copy (a part of) the bodies to the heads, the size of $P^{\mathrm{f}}$ only increases linearly in the size of $P$.

**Theorem 5.21.** *Let $P$ be a program. The size of $P^{\mathrm{f}}$ is linear in the size of $P$.*

*Proof.* Immediate. □

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

**Theorem 5.22.** *Let $P$ be a program. 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 $\gamma$. 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 $\gamma$ containing $X$ and $Y$.

□

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

- Formula (5.2) is replaced by

$$\exists X \cdot X = X \,, \tag{5.8}$$

  such that it is guarded by $X = X$.
- Formula (5.3) is removed if $r : \alpha \leftarrow \beta$ is free or otherwise replaced by

$$\forall \mathbf{Y} \cdot \bigwedge \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \gamma_b) \vee \bigvee \beta^- \,, \tag{5.9}$$

  where $\gamma_b$ 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 $\texttt{comp}(P)$.
- Formula (5.4) is replaced by the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg\beta^- \tag{5.10}$$

  and

$$\forall \mathbf{Y} \cdot \bigwedge \gamma_h \Rightarrow r(\mathbf{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \setminus \gamma_h) \tag{5.11}$$

  where $\gamma_h$ 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 (5.5), replace $E(r)$ by

$$E'(r) \equiv \bigwedge_{t_i \notin \mathbf{Y}} X_i = t_i \wedge \exists \mathbf{Z} \cdot (\bigwedge \beta^+[p|W] \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y}|X_i] \,, \tag{5.12}$$

  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 5.23.* For the fully guarded $p$-program $P$ containing a rule

$$p(X) \vee \mathit{not}\ p(X) \leftarrow p(X)$$

with body and head guard $\{p(X)\}$, one has that $\texttt{sat}(P) = \{\forall X \cdot p(X) \Rightarrow p(X) \vee \neg p(X)\}$, $\texttt{gl}(P) = \{\forall X \cdot r(X) \Leftrightarrow p(X)\}$ and the formula $\phi(W, X_1)$

in $\mathtt{fpf}(P)$ 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 5.24.** *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.* Clearly $(U, M) \models (5.2)$ iff $(U, M) \models (5.8)$. Assume formula (5.3) is replaced by (5.9) (and thus $\alpha \leftarrow \beta$ is non-free). Since the latter is logically equivalent with the former, we have $(U, M) \models (5.3)$ iff $(U, M) \models (5.9)$. Moreover if $\alpha \leftarrow \beta$ is free, i.e., of the form $q(\mathbf{t}) \vee \mathit{not}\ q(\mathbf{t}) \leftarrow$ we have that $(U, M) \models (5.3)$ iff $(U, M) \models \forall Y \cdot \mathbf{true} \Rightarrow q(\mathbf{t}) \vee \neg q(\mathbf{t})$, which is always satisfied. It is easy to see that $(U, M) \models (5.4)$ iff $(U, M) \models (5.10)$ and $(U, M) \models (5.11)$.

Finally, we show that for any substitution $[\mathbf{X}|\mathbf{x}]$,

$$(U, M) \models E(r)[\mathbf{X}|\mathbf{x}] \iff (U, M) \models E'(r)[\mathbf{X}|\mathbf{x}] \ .$$

Assume $(U, M) \models E(r)[\mathbf{X}|\mathbf{x}]$. We can move out the $X_i = t_i$ where $t_i$ is a constant, such that $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge (\exists \mathbf{Y} \cdot \bigwedge_{t_j \in \mathbf{Y}} X_j = t_j \wedge B)[\mathbf{X}|\mathbf{x}]$ with $A \equiv \bigwedge_{t_i \notin \mathbf{Y}} X_i = t_i$ and $B \equiv \bigwedge \beta^+[p|W] \wedge r(\mathbf{Y})$. Thus, there exists a $[\mathbf{Y}|\mathbf{y}]$ such that $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge (\bigwedge_{t_j \in \mathbf{Y}} X_j = t_j \wedge B)[\mathbf{X}|\mathbf{x}, \mathbf{Y}|\mathbf{y}]$ which is well-defined since $\mathbf{X}$ and $\mathbf{Y}$ are disjoint. By $\bigwedge_{t_j \in \mathbf{Y}} X_j = t_j$ we have that $t_j \in \mathbf{Y}$ is grounded by $x_j$, and thus, we can first substitute every $t_j \in \mathbf{Y}$ by $X_j$, and since the mapping of the $t_j \in \mathbf{Y}$ is taken care of by $[\mathbf{X}|\mathbf{x}]$, we can restrict ourselves to $\mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}$ for $[\mathbf{Y}|\mathbf{y}]$. Thus, $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge B[t_j \in \mathbf{Y}|X_j][\mathbf{X}|\mathbf{x}, \mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}|\mathbf{z}]$ where every $z_k = y_k$ for $Y_k \in \mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}$. And thus $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge B[t_j \in \mathbf{Y}|X_j][\mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}|\mathbf{z}][\mathbf{X}|\mathbf{x}]$ such that, with $\mathbf{Z} = \mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}$, we have that $(U, M) \models E'(r)[\mathbf{X}|\mathbf{x}]$.

For the other direction, assume $(U, M) \models E'(r)[\mathbf{X}|\mathbf{x}]$. Then $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge B[t_i \in \mathbf{Y}|X_i][\mathbf{Z}|\mathbf{z}][\mathbf{X}|x]$. Since $t_i$ gets substituted by $X_i$ and $X_i$ is grounded with $x_i$ we have that $\bigwedge_{t_i \in \mathbf{Y}} X_i = t_i$ is true w.r.t. to the latter []. We have that $\mathbf{Z} = \mathbf{Y} \setminus \{t_j \mid t_j \in \mathbf{Y}\}$ such that $[\mathbf{Y}|\mathbf{y}] \equiv [t_i \in \mathbf{Y}|x_i][\mathbf{Z}|\mathbf{z}]$ is well-defined. We then have that $(U, M) \models A[\mathbf{X}|\mathbf{x}] \wedge (\bigwedge_{t_j \in \mathbf{Y}} X_j = t_j \wedge B)[\mathbf{Y}|\mathbf{y}][\mathbf{X}|\mathbf{x}]$. And thus, $(U, M) \models \exists \mathbf{Y} \cdot (A \wedge B)[\mathbf{X}|\mathbf{x}]$, such that $(U, M) \models E(r)[\mathbf{X}|\mathbf{x}]$. □

**Theorem 5.25.** *Let $P$ be a fully (loosely) guarded p-program. Then, the formula $\bigwedge \mathtt{gcomp}(P)$ is a $\mu$(L)GF formula.*

*Proof.* We first show that $[\text{LFP } W\mathbf{X}.\phi'(W, \mathbf{X})](\mathbf{X})$ 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 $\phi'$.

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

- Formula (5.8) is guarded with guard $X = X$.
- Formula (5.9) corresponds with a non-free rule $\alpha \leftarrow \beta$ with a body guard $\gamma_b$; thus $vars(\alpha \leftarrow \beta) \subseteq vars(\gamma_b)$.
  - $\text{free}(\bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \gamma_b) \vee \bigvee \beta^-) \subseteq \mathbf{Y} = vars(\alpha \leftarrow \beta) = vars(\gamma_b) = \text{free}(\bigwedge \gamma_b)$.
  - Take two variables $Y_i$ and $Y_j$ from $\mathbf{Y}$, then $Y_i \in vars(\alpha \leftarrow \beta)$ and $Y_j \in vars(\alpha \leftarrow \beta)$, such that $Y_i$ and $Y_j$ are in an atom from $\gamma_b$.
- Formula (5.10) is guarded with guard $r(\mathbf{Y})$.
- Formula (5.11):
  - For a non-free rule $\alpha \leftarrow \beta$ with a head guard $\gamma_h$. Can be done similarly as formula (5.9).
  - 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 (5.11) is of the form $\forall \mathbf{Y} \cdot q(\mathbf{t}) \Rightarrow r(\mathbf{Y})$.
    - $\text{free}(r(\mathbf{Y})) = \mathbf{Y} = vars(\alpha \leftarrow \beta) = vars(q(\mathbf{t})) = \text{free}(\bigwedge \gamma_h)$.
    - Take two variables $Y_i$ and $Y_j$ from $\mathbf{Y}$, then $Y_i \in vars(\alpha \leftarrow \beta)$ and $Y_j \in vars(\alpha \leftarrow \beta)$, such that $Y_i$ and $Y_j$ are in $vars(q(\mathbf{t})) = \text{free}(\gamma_h)$.
- For the last case, we need to show that $\phi'(\mathbf{X})$ is a $\mu$LGF 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 $\mathtt{gcomp}(P)$ is just a logical rewriting of $\mathtt{comp}(P)$ its size is linear in the size of $\mathtt{comp}(P)$.

**Theorem 5.26.** *Let $P$ be a fully (loosely) guarded p-program. The size of $\mathtt{gcomp}(P)$ is linear in the size of $\mathtt{comp}(P)$.*

*Proof.* The size of formula (5.8) is linear in the size of (5.2). Formula (5.9) is just a shuffling of (5.3). Every formula (5.4) 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 5.27.** *Let $P$ be a (L)GP and $q$ an $n$-ary predicate in $P$. $q$ is satisfiable w.r.t. $P$ iff $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable. Moreover, this reduction is polynomial in the size of $P$.*

*Proof.* By Theorem 5.19 and 5.22, we have that $(P^{\mathrm{f}})_p$ is a fully (loosely) guarded $p$-program, thus the formula $\bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is defined. By Theorem 5.20, we have that $q$ is satisfiable w.r.t. $P$ iff $q$ is satisfiable w.r.t. $P^{\mathrm{f}}$. By Theorem 5.12, we have that $q$ is satisfiable w.r.t. $P^{\mathrm{f}}$ iff $p(\mathbf{X}, \mathbf{0}, q) \wedge \mathtt{comp}((P^{\mathrm{f}})_p)$ is satisfiable. Finally, Theorem 5.24 yields that $q$ is satisfiable w.r.t. $P$ iff $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is satisfiable.

Theorem 5.21, Theorem 5.12, and Theorem 5.26 yield that this reduction is polynomial. □

For a (L)GP $P$, we have, by Theorem 5.25, that $\bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is a $\mu$(L)GF formula such that the formula $p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is as well. Since satisfiability checking for $\mu$(L)GF is 2-EXPTIME-complete (Theorem [1.1] in [GW99]), satisfiability checking w.r.t. $P$ is in 2-EXPTIME.

**Theorem 5.28.** *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 $P^{\mathrm{g}}$ 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 5.29.* 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 $P^{\mathrm{g}}$ 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 5.30.** *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 $P^{\mathrm{g}}$.*

*Proof.* Define $(U \equiv cts(P), M' \equiv M \cup \{g(a,b) \mid a,b \in cts(P)\})$.
$\boxed{\Rightarrow}$ For the "only if" direction, assume $M$ is an answer set of $P$, i.e., $M$ is a minimal model of $P^M_{cts(P)}$. We have that $U \neq \emptyset$ by the assumption that $P$ contains at least one constant, thus $U$ is a universe for $P^{\mathrm{g}}$.

- $M'$ is a model of ${P^{\mathrm{g}}}^{M'}_U$.
  - Take a rule $\alpha^+[] \leftarrow \gamma[], \beta^+[] \in {P^{\mathrm{g}}}^{M'}_U$ with $M' \models \gamma[] \cup \beta^+[]$ originating from $\alpha \leftarrow \gamma, \beta \in P^{\mathrm{g}}$ where $\gamma = \{g(X,Y) \mid X,Y \in vars(P)\}$ and $M' \models not\ \beta^-[]$ and $M' \models \alpha^-[]$.
    Then $\alpha \leftarrow \beta \in P$ and $M \models not\ \beta^-[]$ and $M \models \alpha^-[]$, such that $\alpha^+[] \leftarrow \beta^+[] \in P^M_U$. With $M \models \beta^+[]$, $U = cts(P)$, and $M$ a model of $P^M_U$, we then have that $\exists l \in \alpha^+[] \cdot M \models l$, and thus $M' \models l$.
  - Take a rule $g(a,b) \leftarrow\ \in {P^{\mathrm{g}}}^{M'}_U$. We have that, by definition of $M'$, $g(a,b) \in M'$.
- $M'$ is a minimal model of ${P^{\mathrm{g}}}^{M'}_U$. Assume not, then there is a model $N' \subset M'$ of ${P^{\mathrm{g}}}^{M'}_U$. Define $N \equiv N' \setminus \{g(a,b) \mid a,b \in cts(P)\}$.
  - $N \subset M$. This follows from $g(a,b) \in N'$ iff $g(a,b) \in M'$: $M'$ contains a $g(a,b)$ for all $a,b \in cts(P)$, and so does $N'$ by the rules $g(a,b) \leftarrow\ \in {P^{\mathrm{g}}}^{M'}_U$ and $N'$ being a model of ${P^{\mathrm{g}}}^{M'}_U$.
  - $N$ is a model of $P^M_{cts(P)}$, which is a contradiction with the minimality of $M$. Take a rule $\alpha^+[] \leftarrow \beta^+[] \in P^M_{cts(P)}$ with $M \models \beta^+[]$ originating from $\alpha \leftarrow \beta \in P$ and $M \models not\ \beta^-[]$ and $M \models \alpha^-[]$. Then $\alpha \leftarrow \gamma, \beta \in P$ with $\gamma = \{g(X,Y) \mid X,Y \in vars(\alpha \leftarrow \beta)\}$. Since $M' \models not\ \beta^-[]$ and $M' \models \alpha^-[]$ and $[]$ is a grounding in $U$, we have that $\alpha^+[] \leftarrow \gamma[], \beta^+[] \in {P^{\mathrm{g}}}^{M'}_U$.
    Since $N \models \beta^+[]$, $N' \models \beta^+[]$. Take $g(a,b) \in \gamma[]$. Since $g(a,b) \leftarrow\ \in {P^{\mathrm{g}}}^{M'}_U$ and $N'$ is a model, $N' \models g(a,b)$. And thus $N' \models \gamma[]$. Thus $\exists l \in \alpha^+[] \cdot N' \models l$, and thus $N \models l$.

  $\boxed{\Leftarrow}$ For the "if" direction, assume $(U, M')$ is an open answer set of $P^{\mathrm{g}}$.
  - $M$ is a model of $P^M_{cts(P)}$. This can be done similarly as the above case where $N$ was shown to be a model.
  - $M$ is a minimal model of $P^M_{cts(P)}$. Assume not, then there is a model $N \subset M$ of $P^M_{cts(P)}$. Define $N' \equiv N \cup \{g(a,b) \mid a,b \in cts(P)\}$. Then $N' \subset M'$ and one can again show that $N'$ is a model of ${P^{\mathrm{g}}}^{M'}_U$, which leads to a contradiction with the minimality of $M'$.

□

**Theorem 5.31.** *Let $P$ be a program. The size of $P^{\mathrm{g}}$ 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 $P^{\mathrm{g}}$. 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, $P^{\mathrm{g}}$ is loosely guarded.

**Theorem 5.32.** *Let $P$ be a program. $P^{\mathrm{g}}$ is a LGP.*

*Proof.* Immediate. □

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.

**Lemma 5.33.** *Let $P$ be a LGP with an open interpretation $(U, M)$ and $U' \subseteq U$ such that $M$ contains only terms from $U'$. Then, $(U, M)$ is an open answer set of $P$ iff $(U', M)$ is an open answer set of $P$.*

*Proof.* For the "only if" direction, assume $(U, M)$ is an open answer set of $P$.

- $M$ is a model of $P^M_{U'}$. Take a rule $\alpha^+[] \leftarrow \beta^+[] \in P^M_{U'}$ with $M \models \beta^+[]$ originating from $\alpha \leftarrow \beta$ such that
  - $M \models \alpha^-[]$,
  - $M \models \mathit{not}\ \beta^-[]$,
  - $[]$ grounds in $U'$.

  Since $U' \subseteq U$, $[]$ grounds in $U$ and $\alpha^+[] \leftarrow \beta^+[] \in P^M_U$. $M$ is a model of $P^M_U$ such that $\exists l \in \alpha^+[] \cdot M \models l$.
- $M$ is a minimal model of $P^M_{U'}$. Assume not, then there is a $N \subset M$, $N$ model of $P^M_{U'}$. We prove that $N$ is a model of $P^M_U$, which is a contradiction with the minimality of $M$.

  Take a rule $\alpha^+[] \leftarrow \beta^+[] \in P^M_U$ with $N \models \beta^+[]$ originating from $r : \alpha \leftarrow \beta$ such that
  - $M \models \alpha^-[]$,
  - $M \models \mathit{not}\ \beta^-[]$,
  - $[]$ grounds in $U$.

  Since $N \subset M$ we have that $M \models \beta^+[]$. We distinguish between two cases:
  - $r$ is free. Then $r$ is of the form $q(\mathbf{t}) \vee \mathit{not}\ q(\mathbf{t}) \leftarrow$, such that $q(\mathbf{t})[] \leftarrow\ \in P^M_U$. Since $M$ is a model of $P^M_U$, we have that $q(\mathbf{t})[] \in M$. $M$ contains only terms from $U'$ such that $q(\mathbf{t})[] \leftarrow\ \in P^M_{U'}$ and $q(\mathbf{t})[] \in N$ since $N$ is a model of $P^M_{U'}$.
  - $r$ is non-free. Every variable in $r$ is grounded by $[]$ in $U'$. Indeed, take $X$ in $\mathit{vars}(\alpha \leftarrow \beta)$, then there is a $q(\mathbf{t}) \in \beta^+$ with $X \in \mathbf{t}$ since $P$ is a LGP such that $q(\mathbf{t})[] \in M$, and, since $M$ contains only terms from $U'$, $X[] \in U'$.

    Thus $[]$ grounds in $U'$ and $\alpha^+[] \leftarrow \beta^+[] \in P^M_{U'}$. $N$ is a model of $P^M_{U'}$ such that $\exists l \in \alpha^+[] \cdot N \models l$.

For the "if" direction, assume $(U', M)$ is an open answer set of $P$. Showing that $(U, M)$ is an open answer set of $P$ can be done using the same reasoning as above. □

**Theorem 5.34.** *Let $P$ be a program and $q$ an $n$-ary predicate in $P$. There is an answer set $M$ of $P$ with $q(\mathbf{a}) \in M$ iff $q$ is satisfiable w.r.t. $P^{\mathrm{g}}$. Moreover, this reduction is quadratic.*

*Proof.* For the "only if" direction, assume there is an answer set $M$ of $P$ with $q(\mathbf{a}) \in M$. Then, by Theorem 5.30, $(cts(P), M \cup \{g(a,b) \mid a,b \in cts(P)\})$ is an open answer set of $P^{\mathrm{g}}$, and $q(\mathbf{a}) \in M \cup \{g(a,b) \mid a,b \in cts(P)\}$, such that $q$ is satisfiable w.r.t. $P^{\mathrm{g}}$.

For the "if" direction, assume $q$ is satisfiable w.r.t. $P^{\mathrm{g}}$. Then there exists an open answer set $(U, M')$ of $P^{\mathrm{g}}$ with a $q(\mathbf{x}) \in M$. We have that $cts(P) \subseteq U$.

*Claim.* $M'$ contains only terms from $cts(P)$.

Assume the claim does not hold, thus there is a $r(\mathbf{y}) \in M'$ with some $y \in \mathbf{y}$ such that $y \notin cts(P)$. Since $(U, M')$ is an open answer set there is a $r(\mathbf{y}) \leftarrow \gamma[], \beta^+[] \in P^{\mathrm{g}}{}_U^{M'}$ originating from $r(\mathbf{t}) \vee \alpha \leftarrow \gamma, \beta \in P^{\mathrm{g}}$ such that $\gamma[] \subseteq M'$. Since $y$ is not constant the corresponding $t$ (i.e., such that $t[] = y$) is a variable. And thus, since all rules are loosely guarded, we have that there is some $g(t, Y) \in \gamma$ with $g(y, Y[]) \in \gamma[]$. Thus, since $M' \models \gamma[]$, we have that there must be an applied rule with head $g(y, Y[])$ in $P^{\mathrm{g}}{}_U^{M'}$. However, the only rules in $P^{\mathrm{g}}$ with a $g$-predicate in the head have constants as arguments, thus $y \in cts(P)$, a contradiction, and the claim holds.

$M'$ contains every $g(a,b)$ for $a,b \in cts(P)$ such that we can write $M' = M \cup \{g(a,b) \mid a,b \in cts(P)\}$. Furthermore, since $cts(P) \subseteq U$ by definition of universes and since $P^{\mathrm{g}}$ is a LGP, Lemma 5.33 is applicable such that $(cts(P), M')$ is an open answer set of $P^{\mathrm{g}}$. By Theorem 5.30, we have that $M$ is an answer set of $P$ and $q(\mathbf{x}) \in M$. □

**Theorem 5.35.** *Satisfiability checking w.r.t. LGPs is* NEXPTIME*-hard.*

*Proof.* By [DEGV01, Bar03] 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 5.34, 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 5.6, we improve[5] on Theorem 5.35 and show that both satisfiability checking w.r.t. GPs and w.r.t. LGPs is 2-EXPTIME-hard.

[5] Note that P ⊆ NP ⊆ EXPTIME ⊆ NEXPTIME ⊆ 2-EXPTIME ⊆ ... where P ⊂ EXPTIME, EXPTIME ⊂ 2-EXPTIME, ..., and NP ⊂ NEXPTIME, NEXPTIME ⊂ 2-NEXPTIME, ..., see, e.g., [Pap94, Tob01].

## 5.3 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.

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

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

where $\phi$ is a finite boolean formula of atoms (i.e., using $\neg$, $\vee$, and $\wedge$) and $\psi$ is an atom; we call $\phi$ the *antecedent* and $\psi$ the *consequent*. We refer to extended 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 $\alpha$, $\alpha^{\mathrm{X}} \equiv \{l \mid l \text{ generalized literal in } \alpha\}$, the set of generalized literals in $\alpha$. We extend $\alpha^+$ and $\alpha^-$ for g-literals as follows: $\alpha^+ = (\alpha \backslash \alpha^{\mathrm{X}})^+$ and $\alpha^- = (\alpha \backslash \alpha^{\mathrm{X}})^-$; thus $\alpha = \alpha^+ \cup not\ \alpha^- \cup \alpha^{\mathrm{X}}$.

A *generalized program (gP)* is a countable set of *rules* $\alpha \leftarrow \beta$, where $\alpha$ is a finite set of extended literals, $|\alpha^+| \leq 1$, $\beta$ is a countable[6] set of g-literals, and $\forall t, s \cdot t = s \notin \alpha^+$, i.e., $\alpha$ 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 $vars(l)$ as the (free) variables in $l$. For a rule $r$, we define $vars(r) \equiv \cup\{vars(l) \mid l \text{ g-literal in } r\}$. For a set of atoms $I$, we extend the $\models$ relation, as originally defined on pp. 46 for interpretations $I$, by induction, for any boolean formula of ground atoms. For such ground boolean formulas $\phi$ and $\psi$, 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 $I \not\models \phi$.

A *universe* $U$ for a gP $P$ is again defined as a non-empty countable superset of the constants in $P$. Let $\mathcal{B}_P^U$ be the set of regular ground atoms that can be formed from a gP $P$ and the terms in a universe $U$ for $P$. Call a pair $(U, I)$ where $U$ is a universe for $P$ and $I$ a subset of $\mathcal{B}_P^U$ a *pre-interpretation* of $P$. For a ground gP $P$ and a pre-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

$$\alpha \leftarrow \beta \backslash \beta^{\mathrm{X}}, (\beta^{\mathrm{X}})^{\mathrm{X}(U,I)} \ , \tag{5.13}$$

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

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

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

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[7], 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 $\phi$ 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.

**Definition 5.36.** *An open interpretation of a gP $P$ is a pre-interpretation $(U, M)$ where $M$ is an interpretation of $(P_U)^{\mathrm{X}(U,M)}$. 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, consistency, and query answering remain defined as before.

*Example 5.37.* 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 a pre-interpretation $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$. Then, the GeLi-reduct of $P_{\{x,y\}}$ is

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

Since $\{p(x), r(x), q(x), p(y)\}$ is indeed an interpretation of the latter program, we have that $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$ is an open interpretation. Moreover, $\{p(x), r(x), q(x), p(y)\}$ is an answer set such that the open interpretation $(\{x, y\}, \{p(x), r(x), q(x), p(y)\})$ is an open answer set.

Note that for a gP without generalized literals and a pre-interpretation $(U, I)$ of $P$, $(P_U)^{\mathrm{X}(U,I)} = P_U$, such that the open answer set semantics of Definition 5.36 for gPs without generalized literals coincides with the open answer set semantics of Definition 3.2.

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

*Example 5.38.* Take the following program $P$, i.e., the open answer set variant of the classical infinity axiom in guarded fixed point logic from [GW99] (see also Example 2.27, pp. 58):

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

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 $r_4$:

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

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:

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

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:

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

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 5.39.* 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 5.40.** *Let $P$ be a ground gP with an open interpretation $(U, M)$. Then,*

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

*Proof.* Let $\alpha \leftarrow \beta \in P$.
Then $\alpha^+ \leftarrow (\beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)})^+ \in (P^{\mathrm{x}(U,M)})^M$
iff $\alpha \leftarrow \beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)} \in P^{\mathrm{x}(U,M)}$ with $M \models \mathit{not}\ (\beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)})^-$ and $M \models \alpha^-$
iff $\alpha \leftarrow \beta \in P$, $M \models \mathit{not}\ (\beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)})^-$, $M \models \alpha^-$, and $\beta^{\mathrm{x}\mathrm{x}(U,M)} = \bigcup_{\forall \mathbf{Y} \cdot \phi \Rightarrow \psi \in \beta^{\mathrm{x}}} \{\psi[\mathbf{Y}|\mathbf{y}] \mid \mathbf{y} \subseteq U, M \models \phi[\mathbf{Y}|\mathbf{y}]\}$ (*) with $(\beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)})^- = (\beta \backslash \beta^{\mathrm{x}})^-$
iff $\alpha^+ \leftarrow \beta^+, \beta^{\mathrm{x}} \in P^M$ and (*) holds
iff $\alpha^+ \leftarrow \beta^+, \beta^{\mathrm{x}\mathrm{x}(U,M)} \in (P^M)^{\mathrm{x}(U,M)}$
iff $\alpha^+ \leftarrow (\beta \backslash \beta^{\mathrm{x}})^+, \beta^{\mathrm{x}\mathrm{x}(U,M)} \in (P^M)^{\mathrm{x}(U,M)}$
iff $\alpha^+ \leftarrow (\beta \backslash \beta^{\mathrm{x}}, \beta^{\mathrm{x}\mathrm{x}(U,M)})^+ \in (P^M)^{\mathrm{x}(U,M)}$.

□

We have similar results as in Theorems 3.13 and 3.30, 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* [vEK76] $T$ 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 $T^0(B) = B$[8], and $T^{n+1}(B) = T(T^n(B))$.

**Theorem 5.41.** *Let $P$ be a gP and $(U, M)$ an open answer set of $P$. Then, $\forall a \in M \cdot \exists n < \infty \cdot a \in T^n$.*

*Proof.* The proof is similar to the proof of Theorem 3.13, pp. 66. □

## 5.4 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 5.1, 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 $\alpha$, we construct $\alpha_p$ in two stages:

[8] 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 $T(\emptyset)$.

1. replace every regular $m$-ary atom $q(\mathbf{t})$ appearing in $\alpha$ (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 5.42.* Let $P$ be the gP:

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

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

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

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 5.43.** *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 $p(\mathbf{x}, \mathbf{0}, q) \in M'$. Furthermore, the size of $P_p$ is polynomial in the size of $P$.*

*Proof.* The proof is analogous to the proof of Theorem 5.3. □

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

$$a \neq b \ . \tag{5.14}$$

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:

$$\exists X \cdot \mathbf{true} \ . \tag{5.15}$$

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:

- $\mathtt{sat}(P)$ ensures that a model of $\mathtt{fix}(P)$ satisfies all rules in $P$,
- $\mathtt{gl}(P)$ is an auxiliary component defining atoms that indicate when a rule in $P$ belongs to the GL-reduct,
- $\mathtt{gli}(P)$ indicates when the antecedents of generalized literals are true, and
- $\mathtt{fpf}(P)$ ensures that every model of $\mathtt{fix}(P)$ is a minimal model of the GL-reduct of the GeLi-reduct of $P$; it uses the atoms defined in $\mathtt{gl}(P)$ 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 $\mathtt{gli}(P)$ 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 5.44.** *Let $P$ be a $p$-gP. The fixed point translation of $P$ is $\mathtt{fix}(P) \equiv \mathtt{sat}(P) \cup \mathtt{gli}(P) \cup \mathtt{gl}(P) \cup \mathtt{fpf}(P)$, where*

1. $\mathtt{sat}(P)$ *contains formulas*

$$\forall \mathbf{Y} \cdot \bigwedge \beta \Rightarrow \bigvee \alpha \tag{5.16}$$

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

2. $\mathtt{gl}(P)$ *contains the formulas*

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Leftrightarrow \bigwedge \alpha^- \wedge \bigwedge \neg \beta^- \tag{5.17}$$

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

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

$$\forall \mathbf{Z} \cdot g(\mathbf{Z}) \Leftrightarrow \phi \tag{5.18}$$

*for generalized literals $g : \forall \mathbf{Y} \cdot \phi \Rightarrow \psi \in P$[9] where $\phi$ contains the variables $\mathbf{Z}$,*

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

$$\forall \mathbf{X} \cdot p(\mathbf{X}) \Rightarrow [\mathrm{LFP}\ \ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{X}) \tag{5.19}$$

*with*

$$\phi(W, \mathbf{X}) \equiv W(\mathbf{X}) \vee \bigvee_{r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P} E(r) \tag{5.20}$$

*and*

$$E(r) \equiv \exists \mathbf{Y} \cdot X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge \bigwedge \beta^+[p \mid W] \wedge \bigwedge \gamma \wedge r(\mathbf{Y}) \tag{5.21}$$

*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 $\beta^+$ with $p$ replaced by $W$, and $\gamma$ is $\beta^x$ with*

[9] We assume that generalized literals are named.

- *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* $\phi$*, and, subsequently,*
- *every* $p$ *replaced by* $W$*.*

*The* completion *of* $P$ *is* $\texttt{compgl}(P) \equiv \texttt{fix}(P) \cup \{(5.14), (5.15)\}$.

The predicate $W$ appears only positively in $\phi(W, \mathbf{X})$ such that the fixed point formula in (5.19) 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.[10]

By the first disjunct in (5.20), we have that applying $\phi^{(U,M)}$ to a set $S \subseteq U^n$ does not lose information from $S$.

**Theorem 5.45.** *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 5.6. □

*Example 5.46.* We rewrite the program from Example 5.38 as the $p$-gP $P$:

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

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

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

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

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

[10] 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 5.40, the order of applying the reducts does not matter.

- $\forall Y \cdot r_4(Y) \Leftrightarrow \bigwedge in(Y)$, and
- $\forall X, Y \cdot r_5(X, Y) \Leftrightarrow p(X, Y, f)$.

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

- $E(r_1) \equiv \exists X, Y \cdot X_1 = X \wedge X_2 = 0 \wedge X_3 = q \wedge W(X, Y, f) \wedge r_1(X, Y)$,
- $E(r_4) \equiv \exists Y \cdot X_1 = Y \wedge X_2 = 0 \wedge X_3 = \mathit{well} \wedge W(Y, 0, q) \wedge$ $(\forall X \cdot g(X, Y) \Rightarrow W(X, 0, \mathit{well})) \wedge r_4(Y)$, 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 $U = \{q, f, \mathit{well}, 0, x_0, x_1, \ldots\}$ and[11]

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

$\texttt{sat}(P), \texttt{gl}(P)$, and $\texttt{gli}(P)$ are satisfied. We check that $\texttt{fpf}(P)$ is satisfied by $M$. We first construct the fixed point of $\phi^{(U,M)}$ where $\phi(W, X_1, X_2, X_3) \equiv W(X_1, X_2, X_3) \vee E(r_1) \vee E(r_4) \vee E(R_5)$ as in [Grä02a], 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 $E(r_1)$,
- $W^3 = \phi^{(U,M)}(W^2) = W^2 \cup \{(x_0, 0, \mathit{well})\}$, where $(x_0, 0, \mathit{well})$ is introduced by $E(r_4)$,
- $W^4 = \phi^{(U,M)}(W^3) = W^3 \cup \{(x_1, 0, \mathit{well})\}$,
- $\ldots$

The least fixed point $\mathrm{LFP}(\phi^{(U,M)})$ is then $\cup_{\alpha<\infty} W^\alpha$ [Grä02a]. The sentence $\texttt{fpf}(P)$ is then satisfied since every $p$-literal in $M$ is also in this least fixed point. $(U, M)$ is thus a model of $\texttt{compgl}(P)$, and it corresponds to an open answer set of $P$.

**Theorem 5.47.** *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 $\bigwedge \texttt{compgl}(P)$, where*

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

*i.e., the atoms corresponding to rules for which the GeLi-reduct version will be in the GL-reduct, and*

$$G \equiv \{g(\mathbf{z}) \mid g : \forall \mathbf{Y} \cdot \phi \Rightarrow \psi \in P, \mathit{vars}(\phi) = \mathbf{Z}, M \models \phi[\mathbf{Z} \mid \mathbf{z}]\} \, ,$$

*i.e., the atoms corresponding to true antecedents of generalized literals in $P$.*

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

*Proof.* Denote $M \cup R \cup G$ 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 $\bigwedge \mathtt{compgl}(P)$.

- $(U, M')$ is a model of (5.14), (5.15), $\mathtt{sat}(P)$, $\mathtt{gl}(P)$. This can be done as in the proof of Theorem 5.8.
- $(U, M')$ is a model of $\mathtt{gli}(P)$. By definition of $M'$, we have that $g(\mathbf{z}) \in M'$ iff $g(\mathbf{z}) \in G$ iff $M \models \phi[\mathbf{Z} \mid \mathbf{z}]$ iff $M' \models \phi[\mathbf{Z} \mid \mathbf{z}]$.
- $(U, M')$ is a model of $\mathtt{fpf}(P)$. 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.* $\mathbf{x} \in \phi^{(U,M')} \uparrow n$, $n < \infty$.

  We prove the claim by induction on $n$.
  $n = 1$ (Base step). If $p(\mathbf{x}) \in T^1$ there is some

$$r' : p(\mathbf{x}) \leftarrow \beta[]^+, (\beta^{\mathrm{X}}[])^{\mathrm{X}(U,M)} \in (P_U^{\mathrm{X}(U,M)})^M$$

  originating from $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ with variables $\mathbf{Y} = Y_1, \ldots, Y_d$ where $[] = [\mathbf{Y} \mid \mathbf{y}]$. We have
  - $\emptyset \models \mathrm{body}(r')$[12],
  - $M \models \alpha^-[]$, and
  - $M \models \mathit{not}\ \beta[]^-$.

  Thus $\alpha^-[]$ and $\neg\beta^-[]$ are true in $M$, such that, by definition of $M'$, $r(\mathbf{y}) \in M'$. We show that $(U, M'), \{W \to \emptyset\} \models E(r)[\mathbf{X} \mid \mathbf{x}]$. Because then $(U, M'), \{W \to \emptyset\} \models \phi(W, \mathbf{x})$ such that $\mathbf{x} \in \phi^{(U,M')}(\emptyset) = \phi^{(U,M')} \uparrow 1$.
  Take $\mathbf{y}$, we show that

$$(U, M'), \{W \to \emptyset\} \models x_1 = t_1[\mathbf{Y} \mid \mathbf{y}] \wedge \ldots \wedge x_n = t_n[\mathbf{Y} \mid \mathbf{y}] \wedge \bigwedge \beta^+[p \mid W][\mathbf{Y} \mid \mathbf{y}] \wedge \bigwedge \gamma[\mathbf{Y} \mid \mathbf{y}] \wedge r(\mathbf{y}) \,.$$

  We have that $x_i = t_i[\mathbf{Y} \mid \mathbf{y}]$ since $p(\mathbf{t})[\mathbf{Y} \mid \mathbf{y}] = p(\mathbf{x})$. We already have that $r(\mathbf{y}) \in M'$.
  It remains to show that $(U, M'), \{W \to \emptyset\} \models \bigwedge \beta^+[p \mid W][\mathbf{Y} \mid \mathbf{y}] \wedge \bigwedge \gamma[\mathbf{Y} \mid \mathbf{y}]$.
  - Take $l$ a conjunct in $\bigwedge \beta^+[p \mid W][\mathbf{Y} \mid \mathbf{y}]$. We have that $l$ is either
    - an equality $t = s[]$ for a $t = s \in \beta^+$, then, since $\emptyset \models \beta[]^+$, we have that $(U, M'), \{W \to \emptyset\} \models l$, or
    - an atom $W(\mathbf{U})[]$ for a $p(\mathbf{U}) \in \beta^+$. Then $p(\mathbf{U})[] \in \beta[]^+$, but $\emptyset \models \mathrm{body}(r')$ so this case is not possible.

[12] $\mathrm{body}(r')$ may contain equalities but no regular atoms.

- Take $l$ a conjunct in $\gamma[\mathbf{Y} \mid \mathbf{y}]$. Since $\gamma$ was constructed from $\beta^{\mathrm{X}}$, we have that $l$ is a $\forall \mathbf{Z} \cdot g(\mathbf{Z})[] \Rightarrow \psi'[]$ originating from $\forall \mathbf{Z} \cdot \phi \Rightarrow \psi \in \beta^+$ where $\psi'[]$ is again an equality $t = s[]$ for $\psi \equiv t = s$ or a $W(\mathbf{U})[]$ for $\psi \equiv p(\mathbf{U})$. Assume $(U, M'), \{W \rightarrow \emptyset\} \models g(\mathbf{Z})[][\mathbf{Z} \mid \mathbf{z}]$, and thus, by definition of $M'$, we have that $M \models \phi[][\mathbf{Z} \mid \mathbf{z}]$. We have that $\psi'[][\mathbf{Z} \mid \mathbf{z}]$ is either some $t = s[][\mathbf{Z} \mid \mathbf{z}]$ or a $W(\mathbf{U})[][\mathbf{Z} \mid \mathbf{z}]$. Since $\forall \mathbf{Z} \cdot \phi[] \Rightarrow \psi[] \in \beta[]^+$ and $M \models \phi[][\mathbf{Z} \mid \mathbf{z}]$, we have that $\psi[][\mathbf{Z} \mid \mathbf{z}] \in (\beta^{\mathrm{X}}[])^{\mathrm{X}(U,M)}$. Thus, since $\emptyset \models \mathrm{body}(r')$, we have that $\psi$ must be an equality and $M \models t = s[][\mathbf{Z} \mid \mathbf{z}]$. With $\psi'[][\mathbf{Z} \mid \mathbf{z}]$ is $t = s[][\mathbf{Z} \mid \mathbf{z}]$, we then have that $(U, M'), \{W \rightarrow \emptyset\} \models \psi'[][\mathbf{Z} \mid \mathbf{z}]$.

(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

$$r' : p(\mathbf{x}) \leftarrow \beta[]^+, (\beta^{\mathrm{X}}[])^{\mathrm{X}(U,M)} \in (P_U^{\mathrm{X}(U,M)})^M$$

originating from $r : p(\mathbf{t}) \vee \alpha \leftarrow \beta \in P$ with variables $\mathbf{Y} = Y_1, \ldots, Y_d$ where $[] = [\mathbf{Y} \mid \mathbf{y}]$. We have

- $T^{n-1} \models \mathrm{body}(r')$,
- $M \models \alpha^-[]$, and
- $M \models \mathit{not}\ \beta[]^-$.

We can then prove, again similar as in Theorem 5.8, that $\mathbf{x} \in \phi^{(U,M')} \uparrow n$, $n < \infty$.

We have that $\mathrm{LFP}(\phi^{(U,M')}) = \phi^{(U,M')} \uparrow \alpha$ for some ordinal $\alpha$. If $\alpha < n$, we have that $n = \alpha + k$ for $k < \infty$. One can see that $\phi^{(U,M')} \uparrow n = \phi^{(U,M')} \uparrow \alpha$, such 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 $(U, M')$. If $\alpha \geq n$, we have that $\phi^{(U,M')} \uparrow n \subseteq \phi^{(U,M')} \uparrow \alpha$, and again $\mathbf{x} \in \mathrm{LFP}(\phi^{(U,M')})$, such that $[\mathrm{LFP}\ W\mathbf{X}.\phi(W, \mathbf{X})](\mathbf{x})$ is true in $(U, M')$.

$\boxed{\Leftarrow}$ For the "if" direction, assume $(U, M')$ is a model of $\bigwedge \mathtt{compgl}(P)$. One can show that $(U, M)$ is an open answer set of $P$. □

Using Theorems 5.43 and 5.47, we can reduce satisfiability checking w.r.t. gPs to satisfiability checking in FPL. Moreover, since $\bigwedge \mathtt{compgl}(P)$ contains only one fixed point predicate, the translation falls in the alternation-free fragment of FPL.

**Theorem 5.48.** *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 $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{compgl}(P_p)$ is satisfiable. Moreover, this reduction is polynomial.*

*Proof.* Assume $q$ is satisfiable w.r.t. $P$. By Theorem 5.43, we have that $p(\mathbf{x}, \mathbf{0}, q)$ is in an open answer set of $P_p$, such that with Theorem 5.47, $p(\mathbf{x}, \mathbf{0}, q)$ is in a model of $\bigwedge \mathtt{compgl}(P_p)$.

For the opposite direction, assume $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{compgl}(P_p)$ is satisfiable. Then there is a model $(U, M')$ of $\bigwedge \mathtt{compgl}(P)$ with $p(\mathbf{x}, \mathbf{0}, q) \in M'$. We have that $M' = M \cup R \cup G$ as in Theorem 5.47, such that $(U, M)$ is an

open answer set of $P_p$ and $p(\mathbf{x}, \mathbf{0}, q) \in M$. From Theorem 5.43, we then have an open answer set of $P$ satisfying $q$.

The size of $\bigwedge \mathtt{compgl}(P_p)$ 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 $\bigwedge \mathtt{compgl}(P_p)$ is polynomial in the size of $P$. □

## 5.5 Open Answer Set Programming with Guarded gPs

As we did in Section 5.2 for programs, we introduce in this section a notion of guardedness such that the FPL translation of *guarded gPs* falls in $\mu$GF. We do not, however, consider their *loosely guarded* counterpart like we did in Section 5.2, but leave this as an exercise to the reader.

**Definition 5.49.** *A generalized literal* $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ *is* guarded *if* $\phi$ *is of the form* $\gamma \wedge \phi'$ *with* $\gamma$ *an atom, and* $vars(\mathbf{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$*; we call* $\gamma$ *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* $\gamma_b$ *a* body guard *of* $r$*. It is* fully guarded *if it is guarded and there is a* $\gamma_h \subseteq \alpha^-$ *such that* $vars(r) \subseteq vars(\gamma_h)$*;* $\gamma_h$ *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 5.50.* Reconsider the gP from Example 5.38. $r_1, r_2$, 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 $P$, $P^{\mathrm{f}}$ is defined as in Section 5.2 (pp. 175), i.e., as $P$ with the rules $\alpha \leftarrow \beta$ replaced by $\alpha \cup \mathit{not}\ \beta^+ \leftarrow \beta$ for the body guard $\gamma_b$ of $\alpha \leftarrow \beta$. For a GgP $P$, we have that $P^{\mathrm{f}}$ is a FGgP, where the head guard of each non-free rule is equal to the body guard. Moreover, the size of $P^{\mathrm{f}}$ is linear in the size of $P$.

**Theorem 5.51.** *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* $P^{\mathrm{f}}$*.*

*Proof.* The proof is analogous to the proof of Theorem 5.20 (pp. 155). □

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

**Theorem 5.52.** *Let* $P$ *be a gP. Then,* $P$ *is a (F)GgP iff* $P_p$ *is a (F)GgP.*

*Proof.* We only prove the GgP case, the case for FGgPs is similar.

For the "only if" direction, take a non-free rule $r_p : \alpha_p \leftarrow \beta_p, \mathit{in}(\mathbf{X}) \in P_p$. We have that $r : \alpha \leftarrow \beta$ is a non-free rule in $P$ where all generalized literals

are guarded and $vars(r) \subseteq vars(\gamma_b)$ with $\gamma_b \in \beta^+$. Take a generalized literal $\forall \mathbf{Y} \cdot \phi_p \wedge \bigwedge in(\mathbf{Y}) \Rightarrow \psi_p$ in $r_p$, then $\forall \mathbf{Y} \cdot \phi \Rightarrow \psi$ is in $r$, where it must be guarded, thus $\phi = \gamma \wedge \phi'$, with $\gamma$ an atom and $vars(\mathbf{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$. Then $\phi_p \wedge \bigwedge in(\mathbf{Y}) = \gamma_p \wedge \phi'_p \wedge \bigwedge in(\mathbf{Y})$ with $\gamma_p$ an atom and $vars(\mathbf{Y}) \cup vars(\phi'_p) \cup vars(in(\mathbf{Y})) \cup vars(\psi_p) \subseteq vars(\gamma_p)$. Thus all generalized literals in $r_p$ are guarded. Furthermore, we have that $vars(r_p) = vars(r) \subseteq vars(\gamma_b) = vars(\gamma_{b_p}) \in {\beta_p}^+$, such that $r_p$ is guarded.

For the "if" direction, take a non-free $r : \alpha \leftarrow \beta \in P$. Then $r_p : \alpha_p \leftarrow \beta_p, in(\mathbf{X})$ is non-free in $P_p$, thus it is guarded and there is a $\gamma_p \in {\beta_p}^+$ such that $vars(r_p) \subseteq \gamma_p$, furthermore, all generalized literals in $r_p$ are guarded.

Take a generalized literal $\forall Y \cdot \phi \Rightarrow \psi$ in $r$, then $\forall Y \cdot \phi_p \wedge \bigwedge in(\mathbf{Y}) \Rightarrow \psi_p$ is guarded in $r_p$ such that $\phi_p = \gamma_p \wedge \phi'_p$ (and thus $\phi = \gamma \wedge \phi'$) such that $vars(\mathbf{Y}) \cup vars(\phi'_p) \cup vars(in(\mathbf{Y})) \cup vars(\psi_p) \subseteq vars(\gamma_p)$ and thus $vars(\mathbf{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$ making the generalized literal in $r$ also guarded. Furthermore, we have that $vars(r) = vars(r_p) \subseteq vars(\gamma_{b_p}) = vars(\gamma_b) \in \beta^+$, such that $r$ is guarded. □

For a fully guarded $p$-gP $P$, we can rewrite $\texttt{compgl}(P)$ as the equivalent $\mu$GF formulas $\texttt{gcompgl}(P)$. 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 $vars(\mathbf{Y}) \cup vars(\phi') \cup vars(\psi) \subseteq vars(\gamma)$, making formula $\xi^g$ a guarded formula. The extension of this operator $\cdot^g$ for sets (or boolean formulas) of generalized literals is as usual.

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

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

$$\exists X \cdot X = X \ , \tag{5.22}$$

  such that it is guarded by $X = X$.
- Formula (5.16) is removed if $r : \alpha \leftarrow \beta$ is free or otherwise replaced by

$$\forall \mathbf{Y} \cdot \gamma_b \Rightarrow \bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \{\gamma_b\}) \vee \bigvee \beta^- \vee \bigvee \neg(\beta^{\mathrm{X}})^g \ , \tag{5.23}$$

  where $\gamma_b$ 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 $q(\mathbf{t}) \vee not\ q(\mathbf{t}) \leftarrow$ 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 $\texttt{compgl}(P)$.
- Formula (5.17) is replaced by the formulas

$$\forall \mathbf{Y} \cdot r(\mathbf{Y}) \Rightarrow \bigwedge \alpha^- \wedge \bigwedge \neg \beta^- \tag{5.24}$$

  and

$$\forall \mathbf{Y} \cdot \gamma_h \Rightarrow r(\mathbf{Y}) \vee \bigvee \beta^- \vee \bigvee \neg(\alpha^- \setminus \{\gamma_h\}) \ , \tag{5.25}$$

where $\gamma_h$ 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 (5.18) is replaced by the formulas

$$\forall \mathbf{Z} \cdot g(\mathbf{Z}) \Rightarrow \phi \tag{5.26}$$

and

$$\forall \mathbf{Z} \cdot \gamma \Rightarrow g(\mathbf{Z}) \vee \neg \phi' \tag{5.27}$$

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 $\gamma$ ($vars(g(\mathbf{Z}) \vee \neg\phi') = vars(\mathbf{Z}) = vars(\gamma)$).
- For every $E(r)$ in (5.19), replace $E(r)$ by

$$E'(r) \equiv \bigwedge_{t_i \notin \mathbf{Y}} X_i = t_i \wedge \exists \mathbf{Z} \cdot (\bigwedge \beta^+[p|W] \wedge \bigwedge \gamma \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y} | X_i] \ , \tag{5.28}$$

with $\mathbf{Z} = \mathbf{Y} \backslash \{t_i \mid t_i \in \mathbf{Y}\}$, 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 5.53.* 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 $\mu$GF formula.

**Theorem 5.54.** *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 5.24 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 $\xi$. □

**Theorem 5.55.** *Let $P$ be a fully guarded p-gP. Then, $\bigwedge \texttt{gcompgl}(P)$ is an alternation-free $\mu$GF formula.*

*Proof.* We first show that [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 $\mathbf{Z}$ bounds all other variables than $\mathbf{X}$. Furthermore, $W$ appears only positively in $\phi'$.

We next show that $\bigwedge \texttt{gcompgl}(P)$ is a $\mu$GF formula if $P$ is fully guarded.

- Formula (5.22) is guarded with guard $X = X$.
- Formula (5.23) for a non-free rule $\alpha \leftarrow \beta$ with a body guard $\gamma_b$; thus $vars(\alpha \leftarrow \beta) \subseteq vars(\gamma_b)$.
  - $vars(\bigvee \alpha \vee \bigvee \neg(\beta^+ \setminus \{\gamma_b\}) \vee \bigvee \beta^- \vee \bigvee \neg(\beta^{\mathbf{X}})^g) \subseteq \mathbf{Y} = vars(\alpha \leftarrow \beta) \subseteq vars(\gamma_b)$.
  - Furthermore, for all $\forall \mathbf{Y} \cdot \gamma \Rightarrow \phi' \vee \psi \in (\beta^{\mathbf{X}})^g$, we have $vars(\psi \vee \neg\phi') \cup vars(\mathbf{Y}) \subseteq vars(\psi) \cup vars(\phi') \cup vars(\mathbf{Y}) \subseteq vars(\gamma)$ (the latter since all generalized literals in $\alpha \leftarrow \beta$ are guarded).
- Formula (5.24) is guarded with guard $r(\mathbf{Y})$.
- Formula (5.25):
  - For a non-free rule $\alpha \leftarrow \beta$ with a head guard $\gamma_h$. Can be done similarly as formula (5.23).
  - 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 (5.25) is of the form $\forall \mathbf{Y} \cdot q(\mathbf{t}) \Rightarrow r(\mathbf{Y})$. $vars(r(\mathbf{Y})) = \mathbf{Y} = vars(\alpha \leftarrow \beta) = vars(q(\mathbf{t})) = vars(\gamma_h)$.
- Formula (5.26) is guarded by $g(\mathbf{Z})$: $vars(\phi) = \mathbf{Z} = vars(g(\mathbf{Z}))$.
- Formula (5.27) is guarded by $\gamma$: $vars(g(\mathbf{Z})) \cup vars(\phi') \cup \mathbf{Z} = vars(\phi) \cup vars(\phi') = vars(\gamma) \cup vars(\phi')$ (the latter since $\phi = \phi' \wedge \gamma$). Which equals $vars(\gamma)$ (since the corresponding guarded generalized literal is guarded: $vars(\phi') \subseteq vars(\gamma)$).
- For the last case, we show that $\phi'(W, \mathbf{X})$ is a guarded formula where $W$ does not appear in guards. Then formula (5.19), with $E'(r)$ instead of $E(r)$, is a valid $\mu$GF-formula.
  We show that for each $r : \alpha \leftarrow \beta$, $\exists \mathbf{Z} \cdot (\bigwedge \beta^+ \wedge \bigwedge \gamma \wedge r(\mathbf{Y}))[t_i \in \mathbf{Y} | X_i]$ is a guarded formula with guard $r(\mathbf{Y})[]$. Thus $W$ does not appear in a guard. Indeed, $vars((\bigwedge \beta^+ \wedge \bigwedge \gamma)[t_i \in \mathbf{Y} | X_i]) \cup \mathbf{Z} = (\mathbf{Y} \setminus \{t_i \in \mathbf{Y}\}) \cup \{X_i \mid t_i \in \mathbf{Y}\} \cup (\mathbf{Y} \setminus \{t_i \mid t_i \in \mathbf{Y}\}) = vars(r(\mathbf{Y})[])$.

Moreover, since $\texttt{gcompgl}(P)$ contains only one fixed point it is alternation-free. □

**Theorem 5.56.** *Let $P$ be a GgP and $q$ an $n$-ary predicate in $P$. $q$ 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. Moreover, this reduction is polynomial.*

*Proof.* We have that $P^{\mathrm{f}}$ is a FGgP. By Theorem 5.52, 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 5.51, we have that $q$ is satisfiable w.r.t. $P$ iff $q$ is satisfiable w.r.t. $P^{\mathrm{f}}$. By Theorem 5.48, we have that $q$ is satisfiable w.r.t. $P^{\mathrm{f}}$ iff $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{compgl}((P^{\mathrm{f}})_p)$ is satisfiable. Finally, Theorem 5.54 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 5.57.** *Satisfiability checking w.r.t. GgPs can be polynomially reduced to satisfiability checking of alternation-free $\mu$GF-formulas.*

*Proof.* For a GgP $P$, we have, by Theorem 5.55, that $\bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$ is an alternation-free $\mu$GF, which yields with Theorem 5.56, the required result. □

**Corollary 5.58.** *Satisfiability checking w.r.t. GgPs is in 2-*EXPTIME.

*Proof.* Since satisfiability checking of $\mu$GF formulas is 2-EXPTIME-complete (Theorem [1.1] in [GW99]), satisfiability checking w.r.t. GgPs is, by Corollary 5.57, 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 5.28.

In [Syr04], $\omega$-restricted programs allow for *cardinality constraints* and *conditional literals*. Conditional literals have the form $X.L : A$ where $X$ is a set of variables, $A$ is an atom (the condition) and $L$ is an atom or a naf-atom. Intuitively, conditional literals correspond to generalized literals $\forall X \cdot A \Rightarrow L$, i.e., the defined reducts add instantiations of $L$ to the body if the corresponding instantiation of $A$ is true. However, conditional literals appear only in cardinality constraints $Card(b, S)$[13] where $S$ is a set of literals (possibly conditional), such that a *for all* effect such as with generalized literals cannot be obtained with conditional literals.

Take, for example, the rule $q \leftarrow [\forall X \cdot b(X) \Rightarrow a(X)]$ and a universe $U = \{x_1, x_2\}$ with an interpretation containing $b(x_1)$ and $b(x_2)$. The reduct will contain a rule $q \leftarrow a(x_1), a(x_2)$ such that, effectively, $q$ holds only if $a$ holds everywhere where $b$ holds. The equivalent rule rewritten with a conditional literal would be something like $q \leftarrow Card(n, \{X.a(X) : b(X)\})$, resulting[14] in a rule $q \leftarrow Card(n, \{a(x_1), a(x_2)\})$. In order to have the *for all* effect, we have that $n$ must be 2. However, we cannot know this $n$ in advance, making it impossible to express a *for all* restriction.

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

[14] Assume we again have a universe $\{x_1, x_2\}$, formally, this is the Herbrand Universe.

Further note that consistent $\omega$-restricted programs (with cardinality constraints and conditional literals) always have finite answer sets, which would make a reduction from GgPs (in which infinity axioms can be expressed) to $\omega$-restricted programs non-trivial.

## 5.6 Relationship with Datalog LITE

We define *Datalog* LITE as in [GGV02]. A *Datalog rule* is a rule $\alpha \leftarrow \beta$ where $\alpha = \{a\}$ for some atom $a$ and $\beta$ does not contain generalized literals. A *basic Datalog program* is a finite set of Datalog rules such that no head predicate appears in negative bodies of rules. Predicates that appear only in the body of rules are *extensional* or *input* predicates. Note that equality is, by the definition of rules, never a head predicate and thus always extensional. The semantics of a basic Datalog program $P$, given a relational input structure $\mathcal{U}$ defined over extensional predicates of $P$[15], is given by the unique (subset) minimal model of $\Sigma_P$ whose restriction to the extensional predicates yields $\mathcal{U}$ ($\Sigma_P$ are the first-order clauses corresponding to $P$, see [AHV95]).

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 $\Sigma_P$ 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 [AHV95]:

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

where $S_i \equiv P_i(\mathcal{U}_{i-1}|edb(P_i))$ is the minimal model of $\Sigma_{P_i}$ among those models of $\Sigma_{P_i}$ 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

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

$\mathbf{Z} \equiv \mathbf{Y} \setminus \{Y \mid Y \notin \mathit{vars}(a)\}$, 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 5.17.

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 [GGV02]: 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.

### 5.6.1 Reduction from GgPs to Datalog LITE

In [GGV02], Theorem 8.5., a Datalog LITE query $(\pi_\varphi, q_\varphi)$ was defined for an alternation-free $\mu$GF sentence $\varphi$ such that

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

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

*Example 5.59.* Take the $\mu$GF sentence $\mathtt{gcomp}(P) \equiv \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ from Example 5.23, 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 $(\pi_{\mathtt{gcomp}(P)}, q_{\mathtt{gcomp}(P)})$ 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 $\varphi_1$ where both rules are guarded by the guard $p(X)$ of $\varphi_1$ (or, in general, the guard in the most closely encompassing scope). Disjunction is defined as usual:

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

where $p(X)$ serves again as guard.[16] The sentence $\varphi_1$ itself is translated into

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

Formulas $\varphi_2$ and $\varphi_3$ can be translated similarly. For $\varphi_4$, 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.,

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

The sentence $\varphi_4$ 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 $q_{\mathtt{gcomp}(P)} \leftarrow H_{\varphi_1}, H_{\varphi_2}, H_{\varphi_3}, H_{\varphi_4}$.

In Example 5.23, we had, for a universe $\{x\}$, the unique model $(\{x\}, \emptyset)$ of $\mathtt{gcomp}(P)$. Accordingly, we have that $\{x = x\}$ is the only relational input structure on the extensional predicates of $\pi_{\mathtt{gcomp}(P)}$, $r$ and $p$, that contains the term $x$ and results in a least fixed point model of $\pi_{\mathtt{gcomp}(P)}$ containing $q_{\mathtt{gcomp}(P)}$.

For the formal details of this reduction, we refer to [GGV02]. Satisfiability checking w.r.t. GgPs can be polynomially reduced, using the above reduction, to satisfiability checking in Datalog LITE.

**Theorem 5.60.** *Let $P$ be a GgP, $q$ an $n$-ary predicate in $P$, and $\varphi$ the $\mu GF$ sentence $\exists \mathbf{X} \cdot p(\mathbf{X}, \mathbf{0}, q) \wedge \bigwedge \mathtt{gcomp}((P^{\mathrm{f}})_p)$. $q$ is satisfiable w.r.t. $P$ iff $(\pi_\varphi, q_\varphi)$ is satisfiable. Moreover, this reduction is polynomial.*

*Proof.* By Theorem 5.56, we have that $q$ is satisfiable w.r.t. $P$ iff $\varphi$ is satisfiable. Since $\varphi$ is a $\mu$GF sentence, we have that $\varphi$ is satisfiable, i.e., there exists

[16] Actually, in this particular case, the rules would already be guarded without the guard of $\varphi_1$, but we include it, as this is not true in general.

a $(U, M)$ such that $(U, M) \models \varphi$, iff $(\pi_\varphi, q_\varphi)(M \cup id(U))$ evaluates to true, i.e., $(\pi_\varphi, q_\varphi)$ is satisfiable.

Since, by Theorem 5.56, the translation of $P$ to $\varphi$ is polynomial in the size of $P$ and the query $(\pi_\varphi, q_\varphi)$ is polynomial in $\varphi$ [GGV02], we have a polynomial reduction. □

### 5.6.2 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 5.61.** *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 $p(\mathbf{x}) \in S_j$, 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 5.62.** *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*

$$p(\mathbf{x}) \in S_j \iff p(\mathbf{x}) \in \mathcal{U}_n \,. \tag{5.29}$$

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

*Proof.* The "only if" direction of Equation (5.29) 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 5.61, $p(\mathbf{x}) \in \mathcal{U}_{k-1}|edb(P_k)$ 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 5.61, 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 5.63.** *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 $R \equiv (P_U^{\mathbf{X}(U,M)})^M$.
   Take a rule $r : a[\mathbf{X} \mid \mathbf{x}] \leftarrow \beta[\mathbf{X} \mid \mathbf{x}]^+, (\beta[\mathbf{X} \mid \mathbf{x}]^{\mathbf{X}})^{\mathbf{X}(U,M)} \in R$, 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 $b[] \in (\beta[\mathbf{X} \mid \mathbf{x}]^{\mathbf{X}})^{\mathbf{X}(U,M)}$, and thus, with $M \models b[]$, that $\mathcal{U}_n \models b[]$. With Theorem 15.2.11 in [AHV95], we have that $\mathcal{U}_n$ is a model of $\Sigma_P$, such that $a[] \in \mathcal{U}_n$, and thus $M \models a[]$.
2. $M$ is a minimal model of $R \equiv (P_U^{\mathbf{X}(U,M)})^M$.
   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 $N_j \equiv S_j \setminus (S_j \setminus N')$. We 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 $\Sigma_{P_j}$, which is a contradiction with the minimality of $S_j$.
   a) $N_j \subset S_j$. Immediate.
   b) $N_j | edb(P_j) = \mathcal{U}_{j-1} | edb(P_j)$.
      $l \in N_j | edb(P_j) \Rightarrow l \in S_j | edb(P_j)$ [Def $N_j$]
      $\Rightarrow l \in \mathcal{U}_{j-1} | edb(P_j)$ [Def $S_j$]
      and
      $l \in \mathcal{U}_{j-1} | edb(P_j) \Rightarrow l \in S_j | edb(P_j) \wedge l \in N'$ [Def $S_j$ and $\mathcal{U}_{j-1} \subseteq N'$]
      $\Rightarrow l \in N_j | edb(P_j)$ [Def $N_j$]
   c) $N_j$ is a model of $\Sigma_{P_j}$. Take $\forall \mathbf{X} \cdot \bigwedge \beta \Rightarrow a \in \Sigma_{P_j}$. Assume $N_j \models \bigwedge \beta[]$.
      - $a[] \leftarrow \beta[]^+, (\beta[]^{\mathbf{X}})^{\mathbf{X}(U,M)} \in R$. Indeed, $M \models$ *not* $\beta[]^-$: take a *not* $p(\mathbf{x}) \in$ *not* $\beta[]^-$. Then $N_j \models \neg p(\mathbf{x})$. $p$ is negative in a body of a rule in $P_j$ and thus $p \in edb(P_j)$. Moreover, $p(\mathbf{x}) \notin \mathcal{U}_{j-1}$. Indeed, if $p(\mathbf{x})$ were in $\mathcal{U}_{j-1}$, one would have that $p(\mathbf{x}) \in N_j$ by $b$), a contradiction. We have, by Lemma 5.62, that $p(\mathbf{x}) \notin \mathcal{U}_n$.
        $p(\mathbf{x}) \notin \mathcal{U}_n$, such that $M \not\models p(\mathbf{x})$ and $M \models$ *not* $p(\mathbf{x})$.
      - $N \models \beta[]^+, (\beta[]^{\mathbf{X}})^{\mathbf{X}(U,M)}$.

- Take $p(\mathbf{x}) \in \beta[]^+$. Then $N_j \models p(\mathbf{x})$, such that $S_j \models p(\mathbf{x})$ and $S_j \setminus N' \not\models p(\mathbf{x})$. Then, $S_j \models p(\mathbf{x})$ and $N' \models p(\mathbf{x})$, such that $p(\mathbf{x}) \in N$ or $p(\mathbf{x}) \in id(U)$, and thus $N \models p(\mathbf{x})$.
- Take $b[][\mathbf{Y} \mid \mathbf{y}] \in (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)}$ for $\forall \mathbf{Y} \cdot c[] \Rightarrow b[] \in \beta[]^{\mathrm{X}}$. Then $M \models c[][]$. We have, by definition of generalized literals, that $c$ is a head predicate in in a lower stratum $P_k$: $k < j$. By Lemma 5.62, we have that $c[][] \in S_k$, $k < j$, such that $c[][] \in \mathcal{U}_k|edb(P_j) \subseteq \mathcal{U}_{j-1}|edb(P_j)$. And thus, since $\mathcal{U}_{j-1}|edb(P_j) = S_j|edb(P_j)$, we have that $c[][] \in S_j$.
  Furthermore, since $\mathcal{U}_{j-1} \subseteq N'$, we have that $c[][] \in N'$, and thus $c[][] \in N_j$. Since $N_j \models \forall \mathbf{Y} \cdot c[] \Rightarrow b[]$, we have that $N_j \models b[][]$, and as before, we have then that $N \models b[][]$.

  Thus since $N$ is a model of $R$, we have that $a[] \in N$ and thus $a[] \in N'$. Furthermore, since $N \subset M$, we have that $a[] \in \mathcal{U}_n$. Since $a$ is a head predicate in $P_j$, we have that, with Lemma 5.62, that $a[] \in S_j$. And thus $a[] \in N_j$.

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.

We 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 $l \in \mathcal{U}_n$.

- BASE CASE. Assume $k = 1$. For the "only if" direction, assume $p(\mathbf{x}) \in M$. Since $M$ is an answer set, we have that $p(\mathbf{x}) \in T_M^{n_1}$, $n_1 < \infty$, We prove, by induction on $n_1$, that if $q(\mathbf{y}) \in T_M^{n_1}$ for a head predicate $q \in P_1$, then $q(\mathbf{y}) \in M'$ (and thus in particular for $p(\mathbf{x})$).
  - BASE CASE. Assume $n_1 = 1$. Thus there is a

    $$r : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)} \in R \equiv (P_U^{\mathrm{X}(U,M)})^M$$

    with $\emptyset \models \mathrm{body}(r)$, and $M \models \mathit{not}\ \beta[]$, originating from some $q(\mathbf{t}) \leftarrow \beta \in P_1$. Since $\mathit{not}\ \beta$ contains only extensional predicates, we have that $M' \models \mathit{not}\ \beta[]$. We then have that $r' : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')} \in R' \equiv (P_U^{\mathrm{X}(U,M')})^{M'}$. Moreover, $\emptyset \models (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')}$. Indeed, assume $b[\mathbf{Y} \mid \mathbf{y}] \in (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')}$ for $\forall \mathbf{Y} \cdot c \Rightarrow b \in \beta[]$, then $M' \models c[]$. Since $c$ is defined to be in a lower stratum, then the rule it appears in, we have that $c$ contains an extensional predicate, and thus $M \models c[]$ such that $b[] \in (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)}$, and then $\emptyset \models b[]$.
    Thus, since $M'$ is a model of $R'$, we have that $q(\mathbf{y}) \in M'$.

- INDUCTION HYPOTHESIS. If $q(\mathbf{y}) \in T_M^{n_1-1}$ for a head predicate $q \in P_1$, then $q(\mathbf{y}) \in M'$
- INDUCTION. Assume $q(\mathbf{y}) \in T_M^{n_1}$. Thus there is a $r : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)} \in R$ with $T_M^{n_1-1} \models \mathrm{body}(r)$, and $M \models \mathit{not}\ \beta[]$. Since $\mathit{not}\ \beta$ contains only extensional predicates, we have that $M' \models \mathit{not}\ \beta[]$. We then have that $r' : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')} \in R'$.
  By induction, we have that $M' \models \mathrm{body}(r')$, and , since $M'$ is a model of $R'$, we have that $q(\mathbf{y}) \in M'$.

The "if" direction is entirely analogous.

- INDUCTION HYPOTHESIS. Assume that for a head predicate $p$ in $P_l$, $l < k$, $M \models p(\mathbf{x})$ iff $M' \models p(\mathbf{x})$.
- INDUCTION. Assume $p$ is a head predicate in $P_k$. For the "only if" direction, assume $p(\mathbf{x}) \in M$. Since $M$ is an answer set, we have that $p(\mathbf{x}) \in T_M^{n_k}$. We prove, by induction on $n_k$, that if $q(\mathbf{y}) \in T_M^{n_k}$ for a head predicate $q \in P_k$, then $q(\mathbf{y}) \in M'$ (and thus in particular for $p(\mathbf{x})$).
  - BASE CASE. Assume $n_k = 1$. Thus there is a

$$r : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)} \in R$$

    with $\emptyset \models \mathrm{body}(r)$, and $M \models \mathit{not}\ \beta[]$. Since $\mathit{not}\ \beta$ contains only head predicates from lower strata or extensional predicates, we have, by induction, that $M' \models \mathit{not}\ \beta[]$. We then have that $r' : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')} \in R'$. Moreover, $\emptyset \models (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')}$. Indeed, assume $b[\mathbf{Y} \mid \mathbf{y}] \in (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')}$ for $\forall \mathbf{Y} \cdot c \Rightarrow b \in \beta[]$, then $M' \models c[]$. Since $c$ is defined to be in a lower stratum or is extensional, we have by induction that $M \models c[]$ such that $b[] \in (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)}$, and then $\emptyset \models b[]$.
    Thus, since $M'$ is a model of $R'$, we have that $q(\mathbf{y}) \in M'$.
  - INDUCTION HYPOTHESIS. If $q(\mathbf{y}) \in T_M^{n_k-1}$ for a head predicate $q \in P_k$, then $q(\mathbf{y}) \in M'$
  - INDUCTION. Assume $q(\mathbf{y}) \in T_M^{n_k}$. Thus there is a $r : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M)} \in R$ with $T_M^{n_k-1} \models \mathrm{body}(r)$, and $M \models \mathit{not}\ \beta[]$. Since $\mathit{not}\ \beta$ contains only extensional predicates or head predicates from lower strata, we have, by induction, that $M' \models \mathit{not}\ \beta[]$. We then have that $r' : q(\mathbf{y}) \leftarrow \beta[]^+, (\beta[]^{\mathrm{X}})^{\mathrm{X}(U,M')} \in R'$.
    By induction, we have that $M' \models \mathrm{body}(r')$, and , since $M'$ is a model of $R'$, we have that $q(\mathbf{y}) \in M'$.

  The "if" direction is entirely analogous.

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

From Theorem 5.63, we obtain a generalization of Corollary 2 in [GL88] (*If $\Pi$ 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 5.64.** *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 5.63, to take into account arbitrary input structures $\mathcal{U}$. For a stratified Datalog program $P$, possibly with generalized literals, define $F_P \equiv \{q(\mathbf{X}) \vee not\ q(\mathbf{X}) \leftarrow | \ q \text{ extensional (but not } =) \text{ in } P\}$.

**Theorem 5.65.** *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 5.63, 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 5.63. □

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 5.66.** *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. 181) 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 5.67.** *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 5.65 and 5.66. □

Theorems 5.60 and 5.67 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).[17] Furthermore, since Datalog LITEM, Datalog LITE, and alternation-free $\mu$GF are equivalent as well [GGV02], we have the following result.

**Theorem 5.68.** *Datalog* LITE*, alternation-free $\mu$GF, and open ASP with GgPs are equivalent.*

Satisfiability checking in both GF and LGF is 2-EXPTIME-complete [Grä99], as are their (alternation-free) extensions with fixed point predicates $\mu$GF and $\mu$LGF [GW99]. Theorem 5.68 gives us then immediately the following complexity result.

**Theorem 5.69.** *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 [GGV02], this yields an equivalent program $\neg\neg P$, where the recursion-freeness ensures that $\neg\neg P$ is stratified.

**Theorem 5.70.** *Let $P$ be a Datalog* LITER *program. Then $\neg\neg P \cup F_{\neg\neg P}$ is a GP.*

[17] Note that $(\pi_\varphi, q_\varphi)$ is a Datalog LITEM query [GGV02].

*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 5.71.** *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 5.65, $(\neg\neg P, q)$ 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 5.72.** *Satisfiability checking w.r.t. (L)GPs is 2-*EXPTIME*-complete.*

*Proof.* The reduction from alternation-free $\mu$GF sentences $\varphi$ to Datalog LITE queries $(\pi_\varphi, q_\varphi)$ specializes, as noted in [GGV02], to a reduction from GF sentences to recursion-free Datalog LITE queries. Moreover, the reduction contains only guarded rules such that GF sentences $\varphi$ are actually translated to Datalog LITER queries $(\pi_\varphi, q_\varphi)$.

Satisfiability checking in the guarded fragment GF is 2-EXPTIME-complete [Grä99], such that, using Theorem 5.71 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 5.28, 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 5.28. □

## 5.7 Application: CTL Reasoning using GgPs

In this section, we show how to reduce CTL satisfiability checking (see Section 2.3.3, pp. 54) to satisfiability checking w.r.t. GgPs, i.e., guarded programs with generalized literals.

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 [HR00]. One can show that $\mathsf{AX}p$ is equivalent to $\neg\mathsf{EX}\neg p$. Intuitively, $p$ holds at all next successors if there is no successor where $p$ does not hold. The formula $\mathsf{A}(p\ \mathsf{U}\ q)$ is equivalent with $\neg\mathsf{E}(\neg q\ \mathsf{U}\ (\neg p \wedge \neg q)) \wedge \mathsf{AF}q$. This is less easy to see immediately, but, to illustrate one direction, assume $\neg\mathsf{E}(\neg q\ \mathsf{U}\ (\neg p \wedge \neg q)) \wedge \mathsf{AF}q$ is satisfiable. By the second conjunct we have that, on all paths, we eventually

have a $q$. Choose the location of this $q$ to be minimal on each path, i.e., $q$ does not hold on earlier states of the path. It remains to show that $p$ holds up until every such minimal $q$. Assume not, then there is an earlier state where $\neg p \wedge \neg q$ holds, and thus, by $\neg \mathsf{E}(\neg q \ \mathsf{U} \ (\neg p \wedge \neg q))$, there must be earlier state where $q$ holds, which is a contradiction, since the chosen state were $q$ holds was minimal.

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:

$$[P](S) \vee not\ [P](S) \leftarrow \tag{$g_1$}$$

$$next(S,N) \vee not\ next(S,N) \leftarrow \tag{$g_2$}$$

$$succ(S) \leftarrow next(S,N) \quad \leftarrow S = S, not\ succ(S) \tag{$g_3$}$$

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

The 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

$$[\neg q](S) \leftarrow not\ [q](S) \tag{$d_1$}$$

  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

$$[q \wedge r](S) \leftarrow [q](S), [r](S) \tag{$d_2$}$$

  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

$$[\mathsf{AF}q](S) \leftarrow [q](S) \tag{$d_3^1$}$$

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

  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:

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

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 5.73.* Consider the absence of starvation formula (Example 2.25, pp. 55) $t \Rightarrow \mathsf{AF}c$. 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

$$[t](S) \vee not\ [t](S) \leftarrow$$
$$[c](S) \vee not\ [c](S) \leftarrow$$
$$next(S,N) \vee not\ next(S,N) \leftarrow$$
$$succ(S) \leftarrow next(S,N)$$
$$\leftarrow S = S, not\ succ(S)$$

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

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

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

**Theorem 5.74.** *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

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

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

1. $M$ is a model of $P \equiv ((G \cup D_p)_S^{\mathbf{X}(S,M)})^M$.
   - Free rules $(g_1)$ are always satisfied by $M$.
   - Free rules $(g_2)$ are always satisfied by $M$.
   - Take a rule $succ(s) \leftarrow next(s,t) \in P$ with $M \models next(s,t)$. Then, by definition of $M$, $(s,t) \in R$ such that, again by definition of $M$, $succ(s) \in M$.
   - There are no constraints $\leftarrow s = s \in P$. Indeed, otherwise $succ(s) \notin M$, such that there is no $t \in S$ for which $(s,t) \in R$, a contradiction since $R$ is total.
   - Take a rule $[\neg q](s) \leftarrow \ \in P$ originating from $(d_1)$. Then $[q](s) \notin M$, such that, by definition of $M$, $K, s \not\models q$. By definition of $\models$ for CTL, we then have that $K, s \models \neg q$, such that, by definition of $M$, $[\neg q](s) \in M$.
   - Take a rule $r : [q \wedge r](s) \leftarrow [q](s), [r](s) \in P$, with $M \models \mathrm{body}(r)$. Similarly to the previous case, we have that $M \models [q \wedge r](s)$.
   - Take a rule $[\mathsf{AF}q](s) \leftarrow [q](s) \in R$ with $[q](s) \in M$. Then, $K, s \models q$, by definition of $M$, such that $K, s \models \mathsf{AF}q$, and, by definition of $M$, $[\mathsf{AF}q](s) \in M$.
   - Take a rule $r : [\mathsf{AF}q](s) \leftarrow [\mathsf{AF}q](t_1), \ldots \in R$, originating from $(d_3^2)$, with $M \models \mathrm{body}(r)$. By definition of the GeLi-reduct, we have that for all $t_i$ in $r$, $next(s,t_i) \in M$, such that $(s,t_i) \in R$. Thus, we have that for all $t_i$ where $(s,t_i) \in R$, then $K, t_i \models \mathsf{AF}q$, such that $K, s \models \mathsf{AF}q$. Thus, by definition of $M$, we have that $[\mathsf{AF}q](s) \in M$.
   - Take a rule $[\mathsf{E}(q \ \mathsf{U} \ r)](s) \leftarrow [r](s) \in R$, with $[r](s) \in M$. Then $K, s \models r$, such that $K, s \models \mathsf{E}(q \ \mathsf{U} \ r)$, and thus, by definition of $M$, $[\mathsf{E}(q \ \mathsf{U} \ r)](s) \in M$.
   - Take a rule $r : [\mathsf{E}(q \ \mathsf{U} \ r)](s) \leftarrow [q](s), next(s,t), [\mathsf{E}(q \ \mathsf{U} \ r)](t) \in R$, with $M \models \mathrm{body}(r)$. Then $K, s \models q$, $(s,t) \in R$, $K, t \models \mathsf{E}(q \ \mathsf{U} \ r)$, such that $K, s \models \mathsf{E}(q \ \mathsf{U} \ r)$, and by definition of $M$, we have that $[\mathsf{E}(q \ \mathsf{U} \ r)](s) \in M$.
   - Take a rule $r : [\mathsf{EX}q](s) \leftarrow next(s,t), [q](t) \in R$ with $M \models \mathrm{body}(r)$. Similarly as before, it follows that $[\mathsf{EX}q](s) \in M$.
2. $M$ is a minimal model of $P$. Assume not, then there is a $N \subset M$, model of $P$. We show that this leads to a contradiction, by showing that $M \subseteq N$.
   - $next(s,t) \in M$, then $next(s,t) \leftarrow \ \in P$, and since $N$ is a model, we have that $next(s,t) \in N$.

- $succ(s) \in M$, then $(s,t) \in R$, such that $next(s,t) \in M$, and thus, by the previous $next(s,t) \in N$. Thus, $succ(s) \leftarrow next(s,t) \in P$ is applied w.r.t. $N$ such that $succ(s) \in N$.
- $[\neg q](s) \in M$. Then $K, s \models \neg q$, by definition of $M$, such that $K, s \not\models q$, and thus $[q](s) \notin M$. Thus, $[\neg q](s) \leftarrow\in P$, and since $N$ is a model, we have that $[\neg q](s) \in N$.

The rest of the cases are handled by induction on the structure of a literal.

- BASE CASE: $[P](s) \in M$ for a $P \in AP$. With the free rule, we again have that $[P](s) \in N$.
- INDUCTION HYPOTHESIS: Assume that for $q$ and $r$: if $[q](s) \in M$, then $[q](s) \in N$, and similarly for $r$.
- INDUCTION:
  - $[q \wedge r](s) \in M$, then $K, s \models q$ and $K, s \models r$, such that $[q](s) \in M$ and $[r](s) \in M$. By induction, we have that $[q](s) \in N$ and $[r](s) \in N$. Thus with $[q \wedge r](s) \leftarrow [q](s), [r](s) \in P$ applied w.r.t. $N$, we have that $[q \wedge r](s) \in N$.
  - Take $[\mathsf{AF}q](s) \in M$ (and thus, $K, s \models \mathsf{AF}q$). Assume, by contradiction, that $[\mathsf{AF}q](s) \notin N$. Then
    a) $K, s \models \neg q$. Indeed, otherwise, $K, s \models q$, such that $[q](s) \in M$, and, by induction, $[q](s) \in N$. With $[\mathsf{AF}q](s) \leftarrow [q](s) \in P$, we then have that $[\mathsf{AF}q](s) \in N$, a contradiction.
    b) $\exists (s, s') \in R \cdot [\mathsf{AF}q](s') \notin N$. Indeed, otherwise, the body of the GeLi-reduct of $(d_3^2)$ would be applied w.r.t. $N$ such that $[\mathsf{AF}q](s) \in N$, a contradiction.

    With the same reasoning, we have that
    a) $K, s' \models \neg q$, and
    b) $\exists (s', s'') \in R \cdot [\mathsf{AF}q](s') \notin N$.

    Continuing this way, one can construct a path $s_0 = s, s_1 = s', s_2 = s'', \ldots$ where $K, s_i \models \neg q$, for all $1 \leq i$. Thus, $K, s \models \mathsf{EG}\neg q$, a contradiction with $K, s \models \mathsf{AF}q$, such that $[\mathsf{AF}q](s) \in N$.
  - $[\mathsf{E}(q \,\mathsf{U}\, r)](s) \in M$. Then there is path $(s_0, s_1), \ldots, (s_{n-1}, s_n) \in R$ with $s = s_0$, such that $K, s_i \models q$, $1 \leq i < n$ and $K, s_n \models r$. Then $[q](s_i) \in M$, $1 \leq i < n$, and $[r](s_n) \in M$. By induction, we have that $[q](s_i) \in N$, $1 \leq i < n$, and $[r](s_n) \in N$. With rule $(d_4)$, we have that $[\mathsf{E}(q \,\mathsf{U}\, r)](s_n) \in N$. Since $next(s_i, s_{i+1}) \in N$, we have then, with rule $(d_5)$, that $[\mathsf{E}q \,\mathsf{U}\, r](s_{n-1}) \in N$. Continuing this way, we have that $[\mathsf{E}(q \,\mathsf{U}\, r)](s) \in N$.
  - $[\mathsf{EX}q](s) \in M$, then there is a $t$, such that $next(s,t) \in M$ (and thus in $N$), and with $K, t \models q$. Thus $[q](t) \in M$, and, by induction, $[q](t) \in N$, such that with $(d_6)$, we have that $[\mathsf{EX}q](s) \in N$.

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. We prove per induction on the structure of a CTL formula $q$, that

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

- BASE CASE: $q$ is a proposition $P$. It is immediate from the definition of $K$ that: $K, s \models P \iff [P](s) \in M$.
- INDUCTION HYPOTHESIS: It is proved for formulas $q$ and $r$.
- INDUCTION:
  - We have $K, s \models \neg q \iff K, s \not\models q \iff [q](s) \notin M$, where the latter is due to the induction hypothesis. Assume $K, s \models \neg q$, then $[\neg q](s) \leftarrow \in P$, and since $M$ is an answer set, we have that $[\neg q](s) \in M$. The other way around, assume $[\neg q](s) \in M$, then by minimality of $M$, there must be an applied rule with $[\neg q](s)$ in the head. Thus $(d_1)$ is applied and $[q](s) \notin M$, such that $K, s \models \neg q$.
  - We have $K, s \models q \wedge r \iff K, s \models q \wedge K, s \models r \iff [q](s) \in M \wedge [r](s) \in M$, where the latter is due to the induction hypothesis. Assume $K, s \models q \wedge r$, then $[q](s) \in M \wedge [r](s) \in M$, such that, since $M$ is an answer set, with rule $(d_2)$, $[q \wedge r](s) \in M$. Assume $[q \wedge r](s) \in M$, then by minimality of $M$, $[q](s) \in M \wedge [r](s) \in M$, such that $K, s \models q \wedge r$.
  - We have $K, s \models \mathsf{AF}q$ iff $\forall \pi, \pi_0 = s \cdot \exists 1 \leq i \cdot K, \pi_i \models q$.
    Assume $K, s \models \mathsf{AF}q$ then for all chains $next(s, \pi_1) \in M, \ldots$, there is a $\pi_i$ such that $[q](\pi_i) \in M$. Assume, by contradiction, that $[\mathsf{AF}q](s) \notin M$. Then, by rules $(d_3^1)$ and $(d_3^2)$, $[q](s) \notin M$ and there is a $next(s, s') \in M$ such that $[\mathsf{AF}q](s') \notin M$. We repeat the reasoning for $s' = \pi_1$, such that $[q](\pi_1) \notin M$ and there is a $next(\pi_1, s'') \in M$ such that $[\mathsf{AF}q](s'') \notin M$. We can continue this ad infinitum such that we defined a path $\pi$ with $\pi_0 = s$ and no $i$ such that $[q](\pi_i) \in M$. This is a contradiction.
    The other way around, assume $[\mathsf{AF}q](s) \in M$, then, by minimality of $M$, there is either a $[q](s) \in M$, in which case we are done, since, by induction, $K, s \models q$ and thus $K, s \models \mathsf{AF}q$, or we have that for all $s'$ where $next(s, s') \in M$, $[\mathsf{AF}q](s') \in M$. We can continue our argument for $s'$, and, by minimality of $M$, eventually we have to deduce that on every path $\pi$ from $s'$ (and thus from $s$), there is some $\pi_i$ such that $K, \pi_i \models q$.
  - We have $K, s \models \mathsf{E}(q \ \mathsf{U} \ r)$ iff $\exists \pi, \pi_j \cdot \pi_0 = s \wedge \forall 1 \leq i < j \cdot K, \pi_i \models q \wedge K, \pi_j \models r$ iff there exist $next(s, \pi_1) \in M, \ldots$, and $\forall 1 \leq i < j \cdot [q](\pi_i) \in M \wedge [r](\pi_j) \in M$, where the latter is by the induction hypothesis. Assume $K, s \models \mathsf{E}(q \ \mathsf{U} \ r)$, then $\forall 1 \leq i < j \cdot [q](\pi_i) \in M \wedge [r](\pi_j) \in M$. By $(d_4)$, we have that $[\mathsf{E}(q \ \mathsf{U} \ r)](\pi_j) \in M$. By $(d_5)$, we then have that $[\mathsf{E}(q \ \mathsf{U} \ r)](\pi_{j-1}) \in M$, and continuing the application of $(d_5)$, we end up with $[\mathsf{E}(q \ \mathsf{U} \ r)](s) \in M$.
    Assume $[\mathsf{E}(q \ \mathsf{U} \ r)](s) \in M$. By minimality of $M$, we then have either $[r](s) \in M$ or $[q](s) \in M$, $next(s, s') \in M$, and $[\mathsf{E}(q \ \mathsf{U} \ r)](s') \in M$.

In the former case, we have $K, s \models r$ and thus $K, s \models \mathsf{E}(q \mathsf{U} r)$. In the latter case, we have, again by minimality, either $[r](s') \in M$ or $[q](s') \in M$, $next(s', s'') \in M$, and $[\mathsf{E}(q \mathsf{U} r)](s'') \in M$. In the former case, we have $[q](s) \in M$, $next(s, s') \in M$, and $[r](s') \in M$ and thus $K, s \models \mathsf{E}(q \mathsf{U} r)$. In the latter case, we continue the construction. By minimality, this must eventually end with a constructed path $next(s, \pi_1) \in M, \ldots$, and $\forall 1 \leq i < j \cdot [q](\pi_i) \in M \wedge [r](\pi_j) \in M$ such that $K, s \models \mathsf{E}(q \mathsf{U} r)$.

– We have $K, s \models \mathsf{EX}q$ iff $\exists (s,t) \in R \cdot K, t \models q$ iff $\exists next(s,t) \in M \cdot [q](t) \in M$, where the latter is by the induction hypothesis. Assume $K, s \models \mathsf{EX}q$ then $\exists next(s,t) \in M \cdot [q](t) \in M$, with $(d_6)$, we then have that $[\mathsf{EX}q](s) \in M$.
Assume $[\mathsf{EX}q](s) \in M$, then by minimality of $M$ and rule $(d_6)$, there is a $next(s,t) \in M$ such that $[q](t) \in M$, and thus $K, s \models \mathsf{EX}q$.

□

Since CTL satisfiability checking is EXPTIME-complete (see Theorem 2.26, pp. 56) and satisfiability checking w.r.t. GgPs is 2-EXPTIME-complete (see Theorem 5.69, pp. 188), 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 $\psi$ is the maximal number of free variables in its subformulas [Grä02b]. We define *bound* programs by looking at their first-order form and the arity of its predicates.

**Definition 5.75.** *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 5.76.** *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 5.77.** *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 (5.19) on pp. 170 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 (5.19) increases, thus increasing the width. Hence, the restriction that the arity of predicates in $P$ should be bounded as well.

By Theorem 5.56 and 5.57, one can reduce satisfiability checking of a bound GgP to satisfiability of a $\mu$GF-formula with bounded width. The latter

can be done in EXPTIME by Theorem 1.2 in [GW99], such that satisfiability checking w.r.t. bound GgPs is in EXPTIME.

The EXPTIME-hardness follows from Theorem 5.74 and the EXPTIME-hardness of CTL satisfiability checking (Theorem 2.26). □

# 6

# Description Logics Reasoning via Open Answer Set Programming

In Section 6.1, we reduce satisfiability checking in the DL $\mathcal{SHIQ}$ to satisfiability checking under IWA w.r.t. CoLPs, and in Section 6.2, we show how a DL that adds constants and conjunction/disjunction of roles and removes transitive roles from $\mathcal{SHIQ}$, the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, can be simulated by acyclic FoLPs. The DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ extended with DL-safe rules can be simulated using free acyclic EFoLPs as shown in Section 6.3. Section 6.4 describes the DL $\mathcal{DLR}$ which supports $n$-ary relations; a fragment of $\mathcal{DLR}$, so-called $\mathcal{DLR}^{-\{\leq\}}$, can be simulated by bound guarded programs. We discuss in Section 6.5 some of the advantages and disadvantages of using open answer set programming instead of DLs for knowledge representation. Finally, we give an overview of related work in Section 6.6.

## 6.1 Simulating $\mathcal{SHIQ}$

Consider the following knowledge base $\Sigma$ (modified from Example 2.24):

$$\begin{array}{c} \mathit{Personnel} \equiv \mathit{Management} \sqcup \mathit{Workers} \sqcup \exists \mathit{boss}.\mathit{Management} \\ \mathit{Management} \sqsubseteq (\forall \mathit{take\_orders}.\mathit{Management}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Workers}) \end{array}$$

The first axiom expresses that personnel consists exactly of the managers, workers, and those people that are the boss of some managers. The second axiom says that every manager takes only orders from other managers and is the boss of at least 3 workers. Additionally, we assume $\Sigma$ contains the axiom $\mathsf{Trans}(\mathit{boss})$, indicating that if $x$ is a boss of $y$ and $y$ is a boss of $z$, then $x$ is a boss of $z$.

A model of this knowledge base is $\mathcal{I} = (\{j, w_1, w_2, w_3, m\}, \cdot^{\mathcal{I}})$, with $\cdot^{\mathcal{I}}$ defined by

$$\begin{aligned} Workers^{\mathcal{I}} &= \{w_1, w_2, w_3\} \\ Management^{\mathcal{I}} &= \{m\} \\ Personnel^{\mathcal{I}} &= \{j, w_1, w_2, w_3, m\} \\ boss^{\mathcal{I}} &= \{(j,m), (m,w_1), (m,w_2), (m,w_3), (j,w_1), (j,w_2), (j,w_3)\} \\ take\_orders^{\mathcal{I}} &= \emptyset \end{aligned}$$

We translate the two axioms as three CoLP constraints (the first axiom actually corresponds to two terminological axioms)[1]:

$$\begin{aligned} &\leftarrow Per(X), not\ (Man \sqcup Wor \sqcup \exists boss.Man)(X) \\ &\leftarrow not\ Per(X), (Man \sqcup Wor \sqcup \exists boss.Man)(X) \\ &\leftarrow Man(X), not\ ((\forall tak.Man) \sqcap (\geq\ 3\ boss.Wor))(X) \end{aligned}$$

Intuitively, we associate with the concept expressions on either side of $\sqsubseteq$ in a terminological axiom a new predicate name. We conveniently denote this new predicate like the corresponding concept expression. The constraints simulate the behavior of the terminological axioms. E.g., if there is a $Man(x)$ in an open answer set, and there is no $((\forall tak.Man) \sqcap (\geq\ 3\ boss.Wor))(x)$, we have a contradiction. This corresponds to the DL behavior of the corresponding axiom: if $x \in Management^{\mathcal{I}}$ and $x \notin ((\forall tak.Man) \sqcap (\geq\ 3\ boss.Wor))^{\mathcal{I}}$, we have a contradiction as the axiom requires that $Man^{\mathcal{I}} \subseteq ((\forall tak.Man) \sqcap (\geq\ 3\ boss.Wor))^{\mathcal{I}}$ for models $\mathcal{I}$.

Note that we do not encode the transitivity of *boss* directly as a constraint $\leftarrow boss(X, Y), boss(Y, Z), not\ boss(X, Z)$, as this is not a CoLP rule (and cannot be written as one). Instead, we take into account transitivity of roles when defining concept expressions that contain transitive roles (such as $\exists boss.Man$, see below).

After having translated the axioms as CoLP constraints, it remains to define the newly introduced predicates according to the DL semantics. Consider the first constraint

$$\leftarrow Per(X), not\ (Man \sqcup Wor \sqcup \exists boss.Man)(X)$$

We define $Per$ as a free predicate:

$$Per(X) \vee not\ Per(X) \leftarrow$$

Intuitively, the DL semantics gives an *open* (first-order) interpretation to its concept names: a domain element is either in the interpretation of a concept name or not.

Similarly, we have, for that particular constraint, the free rules

[1] We use short names for compactness: *Man* for *Management*, *Wor* for *Workers*, *Per* for *Personnel*, *tak* for *take_orders*. Furthermore, we assume that a logic program may contain predicate names starting with a capital letter; this should not lead to confusion with variables, which appear only as arguments of predicates.

$$\begin{aligned} Man(X) \vee not\ Man(X) &\leftarrow \\ Wor(X) \vee not\ Wor(X) &\leftarrow \\ boss(X, Y) \vee not\ boss(X, Y) &\leftarrow \end{aligned}$$

Note that *boss* is a role name, so we introduce it as a binary predicate. The predicate $(Man \sqcup Wor \sqcup \exists boss.Man)$ can be defined by the rules:

$$\begin{aligned} (Man \sqcup Wor \sqcup \exists boss.Man)(X) &\leftarrow Man(X) \\ (Man \sqcup Wor \sqcup \exists boss.Man)(X) &\leftarrow Wor(X) \\ (Man \sqcup Wor \sqcup \exists boss.Man)(X) &\leftarrow (\exists boss.Man)(X) \end{aligned}$$

Intuitively, if $(Man \sqcup Wor \sqcup \exists boss.Man)(x)$ is in an open answer set, then, by minimality of open answer sets, there has to be either a $Man(x)$, $Wor(x)$, or a $(\exists boss.Man)(x)$. Vice versa, if either $Man(x)$, $Wor(x)$, or a $(\exists boss.Man)(x)$ is in the open answer set, then $(Man \sqcup Wor \sqcup \exists boss.Man)(x)$ has to be as well since the rules must be satisfied. This corresponds exactly to the DL semantics for concept disjunction.

The predicate $(\exists boss.Man)$ is defined by the rules

$$\begin{aligned} (\exists boss.Man)(X) &\leftarrow boss(X, Y), Man(Y) \\ (\exists boss.Man)(X) &\leftarrow boss(X, Y), (\exists boss.Man)(Y) \end{aligned}$$

The rules explicitly say that $(\exists boss.Man)(x)$ holds in an open answer set iff there is some chain $boss(x, u_0), \ldots, boss(u_n, y)$, and $Man(y)$ that hold in that open answer set. By transitivity of *boss*, we should indeed have then that $(x, y) \in boss^{\mathcal{I}}$ such that $x \in (\exists boss.Man)^{\mathcal{I}}$.

The second constraint does not yield any new rules. The last constraint

$$\leftarrow Man(X), not\ ((\forall tak.Man) \sqcap (\geq 3\ boss.Wor))(X)$$

introduces a new free rule for the *tak* predicate:

$$tak(X, Y) \vee not\ tak(X, Y) \leftarrow$$

and a rule that defines concept conjunction as conjunction in the body of a rule:

$$((\forall tak.Man) \sqcap (\geq 3\ boss.Wor))(X) \leftarrow (\forall tak.Man)(X), (\geq 3\ boss.Wor)(X)$$

The predicate $(\forall tak.Man)$ is defined corresponding to the DL equivalence $\forall tak.Man \equiv \neg\exists tak.\neg Man$:

$$\begin{aligned} (\forall tak.Man)(X) &\leftarrow not\ (\exists tak.\neg Man)(X) \\ (\exists tak.\neg Man)(X) &\leftarrow tak(X, Y), (\neg Man)(Y) \\ (\neg Man)(X) &\leftarrow not\ Man(X) \end{aligned}$$

which also shows that negated concept expressions are defined using *not*. Further note that, since *tak* is not transitive, we have no recursion in the rule

for $\exists tak.\neg Man$ like for $\exists boss.Man$: $\exists tak.\neg Man$ should hold only when there is a direct *tak*-connection with a *Man* element.

Finally, the number restriction is defined as follows:

$$(\geq 3\ boss.Wor)(X) \leftarrow boss(X, Y_1), boss(X, Y_2), boss(X, Y_3), \\ Wor(Y_1), Wor(Y_2), Wor(Y_3), \\ Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3$$

It uses inequality to ensure that there are at least 3 different *boss* successors $y$ of some $x$ that are workers in an open answer set iff $(\geq 3\ boss.Wor)(x)$ is in the open answer set.

Before giving the formal translation, define the *closure* $clos(C, \Sigma)$ of a $\mathcal{SHIQ}$ concept expression $C$ and a $\mathcal{SHIQ}$ knowledge base $\Sigma$ as the smallest set satisfying the following conditions:

- $C \in clos(C, \Sigma)$,
- for each $C \sqsubseteq D$ an axiom in $\Sigma$ (role or terminological), $\{C, D\} \subseteq clos(C, \Sigma)$,
- for each $\mathsf{Trans}(R)$ in $\Sigma$, $\{R\} \subseteq clos(C, \Sigma)$,
- for every $D$ in $clos(C, \Sigma)$, we have
  - if $D = \neg D_1$, then $\{D_1\} \subseteq clos(C, \Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  - if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  - if $D = \exists R.D_1$, then $\{R, D_1\} \cup \{\exists S.D_1 \mid S \sqsubseteq^* R, S \neq R, \mathsf{Trans}(S) \in \Sigma\} \subseteq clos(C, \Sigma)$,
  - if $D = \forall R.D_1$, then $\{\exists R.\neg D_1\} \subseteq clos(C, \Sigma)$,
  - if $D = (\leq n\ Q.D_1)$, then $\{(\geq n+1\ Q.D_1)\} \subseteq clos(C, \Sigma)$,
  - if $D = (\geq n\ Q.D_1)$, then $\{Q, D_1\} \subseteq clos(C, \Sigma)$.

Note that for a $R^- \in clos(C, \Sigma)$, we do not necessarily add $R$ to the closure, instead, we replace in the CoLP translation occurrences of inverted roles $R^-$ by the inverted predicate $R^{\mathsf{i}}$. Concerning the addition of the extra $\exists S.D_1$ for $\exists R.D_1$ in the closure, note that $x \in (\exists R.D_1)^{\mathcal{I}}$ holds if there is some $(x, y) \in R^{\mathcal{I}}$ with $y \in D_1^{\mathcal{I}}$ or if there is some $S \sqsubseteq^* R$ with $S$ transitive such that $(x, u_0) \in S^{\mathcal{I}}, \ldots, (u_n, y) \in S^{\mathcal{I}}$ with $y \in D_1^{\mathcal{I}}$. The latter amounts to $x \in (\exists S.D_1)^{\mathcal{I}}$. Thus, in the open answer set setting, we have that $\exists R.D_1(x)$ is in the open answer set if $R(x, y)$ and $D_1(y)$ hold or $\exists S.D_1(x)$ holds for some transitive subrole $S$ of $R$. The predicate $\exists S.D_1$ will be defined by adding recursive rules, as in the above example, hence the inclusion of such predicates in the closure (which will be used to define the actual CoLP translation).

Furthermore, for a $(\leq n\ Q.D_1)$ in the closure, we add $\{(\geq n+1\ Q.D_1)\}$, since we will base our definition of the former predicate on the DL equivalence $(\leq n\ Q.D_1) \equiv \neg(\geq n+1\ Q.D_1)$.

Formally, we define $\Phi(C, \Sigma)$ to be the following CoLP, obtained from the $\mathcal{SHIQ}$ knowledge base $\Sigma$ and the concept expression $C$:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add the constraint

$$\leftarrow C(X), not\ D(X) \tag{6.1}$$

- For each role axiom $R \sqsubseteq S \in \Sigma$, add the constraint

$$\leftarrow r(X, Y), not\ s(X, Y) \tag{6.2}$$

where

$$r \equiv \begin{cases} Q^{\mathsf{i}} & \text{for } R = Q^-, Q \text{ a role name} \\ Q & \text{for } R = Q, Q \text{ a role name} \end{cases}$$

and similarly for $s$, i.e., replace $(\cdot)^-$ by $(\cdot)^{\mathsf{i}}$.

- Next, we distinguish between the types of concept expressions that appear in $clos(C, \Sigma)$. For each $D \in clos(C, \Sigma)$:
  - if $D$ is a concept name, add

$$D(X) \vee not\ D(X) \leftarrow \tag{6.3}$$

  - if $D$ is a role name, add

$$D(X, Y) \vee not\ D(X, Y) \leftarrow \tag{6.4}$$

  - if $D$ is an inverted role name $R^-$ for a role name $R$, add

$$R^{\mathsf{i}}(X, Y) \vee not\ R^{\mathsf{i}}(X, Y) \leftarrow \tag{6.5}$$

  - if $D = \neg E$, add

$$D(X) \leftarrow not\ E(X) \tag{6.6}$$

  - if $D = E \sqcap F$, add

$$D(X) \leftarrow E(X), F(X) \tag{6.7}$$

  - if $D = E \sqcup F$, add

$$\begin{array}{c} D(X) \leftarrow E(X) \\ D(X) \leftarrow F(X) \end{array} \tag{6.8}$$

  - if $D = \exists Q.E$, add

$$D(X) \leftarrow q(X, Y), E(Y) \tag{6.9}$$

where

$$q \equiv \begin{cases} R^{\mathsf{i}} & \text{for } Q = R^-, R \text{ a role name} \\ R & \text{for } Q = R, R \text{ a role name} \end{cases}$$

and for all $S \stackrel{*}{\sqsubseteq} Q$, $S \neq R$, with $\mathsf{Trans}(S) \in \Sigma$, add rules

$$D(X) \leftarrow (\exists S.E)(X) \tag{6.10}$$

If $\mathsf{Trans}(Q) \in \Sigma$, we further add the rule

$$D(X) \leftarrow q(x, y), D(Y) \tag{6.11}$$

– if $D = \forall R.E$, add

$$D(X) \leftarrow not\ (\exists R.\neg E)(X) \tag{6.12}$$

– if $D = (\leq n\ Q.E)$, add

$$D(X) \leftarrow not\ (\geq n+1\ Q.E)(X) \tag{6.13}$$

– if $D = (\geq n\ Q.E)$, add

$$D(X) \leftarrow q(X, Y_1), \ldots, q(X, Y_n), E(Y_1), \ldots, E(Y_n), \cup_{i \neq j}\{Y_i \neq Y_j\} \tag{6.14}$$

where

$$q \equiv \begin{cases} R^{\mathrm{i}} & \text{for } Q = R^-, R \text{ a role name} \\ R & \text{for } Q = R, R \text{ a role name} \end{cases}$$

Rule (6.9) is what one would intuitively expect for the exists restriction. However, in case $Q$ is transitive this rule is not enough. Indeed, if $q(x, y)$, $q(y, z)$, $E(z)$ are in an open answer set, one expects $(\exists Q.E)(x)$ to be in it as well if $Q$ is transitive. However, we have no rules enforcing $q(x, z)$ to be in the open answer set (as remarked above, this leads to non-CoLP rules). We can solve this by adding to (6.9) the rule (6.11), such that such a chain $q(x, y)$, $q(y, z)$, with $E(z)$ in the open answer set correctly deduces $D(x)$.

It may still be that there are transitive subroles of $Q$ that need the same recursive treatment as above. To this end, we introduce rule (6.10).

We do not need such a trick with the number restrictions since the roles $Q$ in a number restriction are required to be simple, i.e., without transitive subroles.

Finally, note how we treat inverted roles, we replace inverted roles $R^-$ by inverted predicates $R^{\mathrm{i}}$, which have, under the IWA (see pp. 72), a similar semantics.

**Theorem 6.1.** *Let $\Sigma$ be a $\mathcal{SHIQ}$ knowledge base and $C$ a $\mathcal{SHIQ}$ concept expression. Then, $\Phi(C, \Sigma)$ is a CoLP, with a size that is polynomial in the size of $C$ and $\Sigma$.*

*Proof.* Observing the rules in $\Phi(C, \Sigma)$, it is clear that this program is a CoLP.

The size of the elements in $clos(C, \Sigma)$ is linear and the size of $clos(C, \Sigma)$ is polynomial in $C$ and $\Sigma$. The size of the CoLP $\Phi(C, \Sigma)$ is polynomial in the size of $clos(C, \Sigma)$. The only non-trivial case in showing the latter arises by the addition of rule (6.14) which introduces $\frac{n(n-1)}{2}$ inequalities for a number restriction $(\geq n\ Q.E)$. We assume, as is not uncommon in DLs (see, e.g., [Tob01]), that the number $n$ is represented in unary notation

$$\underbrace{11\ldots1}_{n}$$

such that the number of introduced inequalities is quadratic in the size of the number restriction. □

**Theorem 6.2.** *A $\mathcal{SHIQ}$ concept expression $C$ is satisfiable w.r.t. a $\mathcal{SHIQ}$ knowledge base $\Sigma$ iff the predicate $C$ is satisfiable under IWA w.r.t. $\Phi(C, \Sigma)$.*

*Proof.* For the "only if" direction, assume the concept expression $C$ is satisfiable w.r.t. $\Sigma$, i.e., there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $C^{\mathcal{I}} \neq \emptyset$. Define $(U, M)$ such that $U \equiv \Delta^{\mathcal{I}}$ and

$$\begin{aligned} M \equiv & \{C(x) \mid x \in C^{\mathcal{I}}, C \in \mathit{clos}(C, \Sigma), C \text{ a concept expression}\} \\ & \cup \{R^{\mathsf{i}}(x, y) \mid (x, y) \in (R^{-})^{\mathcal{I}}, R^{-} \text{ or } R \text{ in } \mathit{clos}(C, \Sigma), R \text{ a role name}\} \\ & \cup \{R(x, y) \mid (x, y) \in R^{\mathcal{I}}, R^{-} \text{ or } R \text{ in } \mathit{clos}(C, \Sigma), R \text{ a role name}\} \ . \end{aligned}$$

We have that $(U, M)$ is an open answer set under IWA of $\Phi(C, \Sigma)$ that satisfies $C$:

1. $(U, M)$ is an open interpretation under IWA of $\Phi(C, \Sigma)$. By the DL semantics of inverted roles and the definition of $M$, we have that $R(x, y) \in M \iff R^{\mathsf{i}}(y, x) \in M$ such that the IWA (Equation (3.1), pp. 72) is satisfied.
2. Since $C^{\mathcal{I}} \neq \emptyset$ there clearly is an $x \in U$ such that $C(x) \in M$.
3. $M$ is a model under IWA of $\Phi(C, \Sigma)^{M}_{U}$. We check that every rule in $\Phi(C, \Sigma)^{M}_{U}$ is satisfiable:
   a) Take a rule $\leftarrow C(x)$ originating from (6.1). Then, $D(x) \notin M$, such that $x \notin D^{\mathcal{I}}$, by definition of $M$, and $x \notin C^{\mathcal{I}}$ by $C \sqsubseteq D$. Thus, $C(x) \notin M$ and the rule is satisfied.
   b) Rules originating from (6.2) can be done similarly.
   c) Rules originating from the free rules (6.3), (6.4), and (6.5) are satisfied.
   d) Take a rule $D(x) \leftarrow\in \Phi(C, \Sigma)^{M}_{U}$ originating from (6.6) such that $E(x) \notin M$ and thus $x \notin E^{\mathcal{I}}$, or, equivalently, $x \in (\neg E)^{\mathcal{I}} = D^{\mathcal{I}}$ such that $D(x) \in M$.
   e) Take a rule $E \sqcap F(x) \leftarrow E(x), F(x)$ originating from (6.7) with $E(x) \in M$ and $F(x) \in M$. Then, $x \in (E \sqcap F)^{\mathcal{I}} = D^{\mathcal{I}}$ such that $D(x) \in M$.
   f) Rules originating from (6.8) can be done like the previous case.
   g) Take a rule $(\exists Q.E)(x) \leftarrow q(x, y), E(y)$ originating from (6.9) with a body true in $M$. Then, $(x, y) \in Q^{\mathcal{I}}$ and $y \in E^{\mathcal{I}}$ such that $x \in (\exists Q.E)^{\mathcal{I}}$ and $(\exists Q.E)(x) \in M$.
   h) Take a rule $D(x) \leftarrow (\exists S.E)(x)$ originating from (6.10) with a body true in $M$. Then, $x \in (\exists S.E)^{\mathcal{I}}$ such that there is a $(x, y) \in S^{\mathcal{I}}$ and $y \in E^{\mathcal{I}}$. Thus, with $S \sqsubseteq Q$, $(x, y) \in Q^{\mathcal{I}}$ and $x \in (\exists Q.E)^{\mathcal{I}}$ such that $(\exists Q.E)(x) \in M$.
   i) Take a rule $D(x) \leftarrow q(x, y), D(y)$ originating from (6.11) with a body true in $M$. Then, $(x, y) \in Q^{\mathcal{I}}$ and $y \in (\exists Q.E)^{\mathcal{I}}$ such that there is a $(y, z) \in Q^{\mathcal{I}}$ and $z \in E^{\mathcal{I}}$. Since $Q$ is transitive, we have that $(x, z) \in Q^{\mathcal{I}}$ with $z \in E^{\mathcal{I}}$ such that $x \in (\exists Q.E)^{\mathcal{I}}$ and, by definition of $M$, $D(x) \in M$.

j) Take a rule $D(x) \leftarrow$ originating from (6.12) such that $(\exists R.\neg E)(x) \notin M$. Thus, $x \notin (\exists R.\neg E)^{\mathcal{I}}$ such that $x \in (\forall R.E)^{\mathcal{I}}$ and $D(x) \in M$.
k) Take a rule $D(x) \leftarrow$ originating from (6.13) such that $(\geq n+1\ Q.E)(x) \notin M$, and thus, $x \notin (\geq n+1\ Q.E)^{\mathcal{I}}$, such that $x \in (\leq n\ Q.E)^{\mathcal{I}}$ and $D(x) \in M$.
l) Take a rule $D(x) \leftarrow q(x,y_1),\ldots,q(x,y_n),E(y_1),\ldots,E(y_n)$ with a body true in $M$. Then $(x,y_1) \in Q^{\mathcal{I}},\ldots,(x,y_n) \in Q^{\mathcal{I}}$ and $y_1 \in E^{\mathcal{I}},\ldots,y_n \in E^{\mathcal{I}}$ with all $y_i$ pairwise different such that $x \in (\geq n\ Q.E)^{\mathcal{I}}$, and $D(x) \in M$.

4. $M$ is a minimal model under IWA of $\Phi(C,\Sigma)^M_U$. Assume not, then there is a model under IWA $N$ of $\Phi(C,\Sigma)^M_U$, such that $N \subset M$. We prove that $M \subseteq N$, which leads to a contradiction. Take $l \in M$. We distinguish between the following cases for $l$:
   a) $l = R(x,y)$ *for a role name* $R$. Then, by definition of $M$, $(x,y) \in R^{\mathcal{I}}$ for $R$ or $R^-$ in $clos(C,\Sigma)$.
      - If $R \in clos(C,\Sigma)$, then $R$ is free and we have that $R(x,y) \leftarrow\in \Phi(C,\Sigma)^M_U$ such that $R(x,y) \in N$.
      - If $R^- \in clos(C,\Sigma)$, then $R^{\mathsf{i}}$ is free. Since $(x,y) \in R^{\mathcal{I}}$, we have that $(y,x) \in (R^-)^{\mathcal{I}}$ and thus $R^{\mathsf{i}}(y,x) \in M$. Then, $R^{\mathsf{i}}(y,x) \leftarrow\in \Phi(C,\Sigma)^M_U$ such that $R^{\mathsf{i}}(y,x) \in N$. Since $N$ satisfies the IWA, we have that $R(x,y) \in N$.
   b) $l = R^i(x,y)$ *for a role name* $R$. This can be done like the previous.
   c) $l = E(x)$ *for a concept expression* $E \in clos(C,\Sigma)$. We look at the structure of $E$ and prove this by induction:
      i. BASE: $E = A$, $A$ a concept name. By rule (6.3), $E(x) \leftarrow\in \Phi(C,\Sigma)^M_U$ such that $E(x) \in N$.
      ii. INDUCTION HYPOTHESIS: Assume it is proved for concept expressions $E'$, $C_1$, and $C_2$.
      iii. $E = \neg E'$. Since $E(x) \in M$, we have that $x \in (\neg E')^{\mathcal{I}}$ such that $x \notin E'^{\mathcal{I}}$ and $E'(x) \notin M$. Then, $E(x) \leftarrow$ in $\Phi(C,\Sigma)^M_U$ and $E(x) \in N$.
      iv. $E = C_1 \sqcap C_2$, then $x \in (C_1 \sqcap C_2)^{\mathcal{I}}$ such that $x \in C_1^{\mathcal{I}}$ and $x \in C_2^{\mathcal{I}}$. Then $C_1(x) \in M$ and $C_2(x) \in M$. By induction, we have $C_1(x) \in N$ and $C_2(x) \in N$. With rule (6.7), we have that $E(x) \in N$.
      v. $E = C_1 \sqcup C_2$. Again per induction, and similar to the previous case.
      vi. $E = \exists Q.E'$. From $E(x) \in M$ we get $x \in (\exists Q.E')^{\mathcal{I}}$ and thus there is a $y$ such that $(x,y) \in Q^{\mathcal{I}}$ and $y \in E'^{\mathcal{I}}$. By definition of $M$, $q(x,y) \in M$ and $E'(y) \in M$. From $q(x,y) \in M$, we have, by the above, that $q(x,y) \in N$. By induction, we also have that $E'(y) \in N$. With rule (6.9), we then have that $E(x) \in N$.

vii. $E = \forall Q.E'$. From $E(x) \in M$, we have that $x \in E^{\mathcal{I}}$ such that $x \notin (\exists Q.\neg E')^{\mathcal{I}}$ and $(\exists Q.\neg E')(x) \notin M$. Thus, $E(x) \leftarrow\in \Phi(C,\Sigma)^M_U$ such that $E(x) \in N$.

viii. $E = (\leq\ n\ Q.E')$. From $(\leq n\ Q.E')(x) \in M$ we have that $x \in (\leq n\ Q.E')^{\mathcal{I}}$, such that $x \notin (\geq n+1\ Q.E')^{\mathcal{I}}$ and thus $E(x) \leftarrow\in \Phi(C,\Sigma)^M_U$, such that $E(x) \in N$.

ix. $E = (\geq\ n\ Q.E')$. Then $x \in E^{\mathcal{I}}$ such that there are at least $n$ different $y_i$ such that $(x,y_i) \in Q^{\mathcal{I}}$ and $y_i \in E'^{\mathcal{I}}$ and thus $q(x,y_i) \in M$ and $E'(y_i) \in M$ such that $q(x,y_i) \in N$ and, by induction, $E'(y_i) \in N$. With rule (6.14), we then have that $E(x) \in N$.

For the "if" direction. Assume $(U,M)$ is an open answer set under IWA of $\Phi(C,\Sigma)$ with $C(u) \in M$. Define an interpretation $\mathcal{I} \equiv (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}} \equiv U$, and $A^{\mathcal{I}} \equiv \{x \mid A(x) \in M\}$, for concept names $A$,

$$R^{\mathcal{I}} \equiv \{(x,y) \mid r(x,y) \in M\} \cup \bigcup_{\mathsf{Trans}(S)\in\Sigma, S \sqsubseteq^* R} (\{(x,y) \mid s(x,y) \in M\})^*$$

for role names or inverted role names $R$, where $()^*$ denotes *transitive closure* and $r$ is as before (equal to $R$ if $R$ is a role name, and $Q^{\mathsf{i}}$ if $R = Q^-$ for a role name $Q$), and similarly for $s$. Intuitively, we define $R$ like $M$ defines it, but since $M$ does not ensure transitivity of roles, we transitively close every subrole $S$ of $R$ that is declared to be transitive in $\Sigma$.

We have that $(R^-)^{\mathcal{I}} = \{(x,y) \mid (y,x) \in R^{\mathcal{I}}\}$ for a role name $R$. Indeed, assume $(x,y) \in (R^-)^{\mathcal{I}}$, then either $R^{\mathsf{i}}(x,y) \in M$ or there is a $S \sqsubseteq^* R^-$, $\mathsf{Trans}(S) \in \Sigma$ such that $(x,y) \in (\{(u,v) \mid s(u,v) \in M\})^*$. In the former case, $R(y,x) \in M$ since $M$ satisfies the IWA and thus $(y,x) \in R^{\mathcal{I}}$. In the latter case, there is a $s(x,u_0) \in M, \ldots, s(u_n,y) \in M$. If $S$ is a role name, we have $s = S$ such that $S^{\mathsf{i}}(y,u_n) \in M, \ldots, S^{\mathsf{i}}(u_0,x) \in M$ otherwise $S(y,u_n) \in M, \ldots, S(u_0,x) \in M$. Since $S \sqsubseteq^* R^-$, we have that $S^- \sqsubseteq^* R$ and $\mathsf{Trans}(S^-) \in \Sigma$ and thus $(y,x) \in R^{\mathcal{I}}$.

The other direction is similar.

*Claim.*

$$x \in D^{\mathcal{I}} \iff D(x) \in M, D \text{ a concept expression .}$$

We prove the claim by induction on the structure of $D$.

1. $D = A$ where $A$ is a concept name. Immediate by the definition of $A^{\mathcal{I}}$.
2. $D = \neg D'$.

$$\begin{aligned} x \in (\neg D')^{\mathcal{I}} &\Leftrightarrow x \notin D'^{\mathcal{I}} \\ &\Leftrightarrow D'(x) \notin M \text{ (by induction)} \\ &\Rightarrow D(x) \in M \text{ (by rule (6.6))} \end{aligned}$$

$$\begin{aligned}\neg D'(x) \in M &\Rightarrow D'(x) \notin M \ (M \text{ minimal and with rule (6.6) )}\\ &\Rightarrow x \notin D'^{\mathcal{I}} \quad \text{(by induction)}\\ &\Rightarrow x \in (\neg D')^{\mathcal{I}}\end{aligned}$$

3. $D = D_1 \sqcap D_2$.

$$\begin{aligned}x \in (D_1 \sqcap D_2)^{\mathcal{I}} &\Leftrightarrow x \in D_1^{\mathcal{I}} \text{ and } x \in D_2^{\mathcal{I}}\\ &\Leftrightarrow D_1(x) \in M \text{ and } D_2(x) \in M \text{ (by induction)}\\ &\Rightarrow D_1 \sqcap D_2(x) \in M \quad \text{(by rule (6.7))}\end{aligned}$$

$$\begin{aligned}D_1 \sqcap D_2(x) \in M &\Rightarrow D_1(x) \in M \text{ and } D_2(x) \in M\\ &\quad (M \text{ minimal, and rule (6.7)})\\ &\Rightarrow x \in {D_1}^{\mathcal{I}} \cap {D_2}^{\mathcal{I}} \text{ (by induction)}\\ &\Rightarrow x \in (D_1 \sqcap D_2)^{\mathcal{I}}\end{aligned}$$

4. $D = D_1 \sqcup D_2$. Like the previous case.
5. $D = \exists Q.D'$. For the "only if" direction, assume $x \in D^{\mathcal{I}}$, then there is a $(x, y) \in Q^{\mathcal{I}}$ and $y \in D'^{\mathcal{I}}$. By induction, we have that $D'(y) \in M$. By definition of $Q^{\mathcal{I}}$, we have that either $q(x, y) \in M$ or $(x, y) \in (\{(u, v) \mid s(x, y) \in M\})^*$ for some $S \sqsubseteq^* Q$ with $\mathsf{Trans}(S) \in \Sigma$.
   In the former case, we have with rule (6.9) that $D(x) \in M$. In the latter case, we have that $s(x, u_0) \in M, \ldots, s(u_n, y) \in M$. Since $\exists S.D' \in clos(C, \Sigma)$, we have that $\exists S.D'(u_n) \in M$ with rule (6.9). Repeatedly applying rule (6.11), yields $\exists S.D'(x) \in M$. By rule (6.10), we then have that $D(x) \in M$.
   For the "if" direction, assume $D(x) \in M$. Then, $D(x) \in T^{\mathrm{i}^n}$ for a finite $n$. We prove it by induction on $n$. Looking at rules (6.9), (6.10), and (6.11), we have that $n > 1$, such that the base case of our induction is $n = 2$.
   - BASE CASE. Assume $D(x) \in T^{\mathrm{i}^2}$. Then neither (6.10) nor (6.11) could have been used to deduce this (since their bodies contain a literal that cannot be in $T^{\mathrm{i}^1}$). Thus, by rule (6.9), we have that $q(x, y) \in M$ and $D'(y) \in M$. Thus, by induction on the structure of concept expressions, we have that $y \in D'^{\mathcal{I}}$ and, with $(x, y) \in Q^{\mathcal{I}}$, we have that $x \in D^{\mathcal{I}}$.
   - INDUCTION HYPOTHESIS. Assume that for concept expressions of the form $\exists S.D'$ and some $y \in U$ with $(\exists S.D')(y) \in T^{\mathrm{i}^{n-1}}$, we have $y \in (\exists S.D')^{\mathcal{I}}$.
   - INDUCTION. Take $(\exists S.D')(y) \in T^{\mathrm{i}^n}$. Then one of the rules originating from (6.9), (6.10), and (6.11) must have an applied body true in $T^{\mathrm{i}^{n-1}}$.
     - Take (6.9) such that $s(y, z) \in M$ and $D'(z) \in M$. By induction on the structure of concept expressions, we have that $(y, z) \in S^{\mathcal{I}}$ and $z \in D'^{\mathcal{I}}$ such that $y \in (\exists S.D')^{\mathcal{I}}$.

- Take (6.10) such that $(\exists S'.D')(y) \in T^{\mathrm{i}^{n-1}}$ for $S' \sqsubseteq^{*} S$ with $S' \neq S$ and $\mathsf{Trans}(S') \in \Sigma$. Then, by induction, $y \in (\exists S'.D')^{\mathcal{I}}$ such that there is a $(y,z) \in S'^{\mathcal{I}}$ and $z \in D'^{\mathcal{I}}$. We have that $s'(y,z) \in M$ or there is a $R \sqsubseteq^{*} S'$, $\mathsf{Trans}(R) \in \Sigma$, such that $(y,z) \in (\{(u,v) \mid r(u,v) \in M\})^{*}$. In the former case, we have with constraints (6.2) that $s(y,z) \in M$ and thus $(y,z) \in S^{\mathcal{I}}$. In the latter case, we have, since $R \sqsubseteq^{*} S$, $\mathsf{Trans}(R) \in \Sigma$, that $(y,z) \in S^{\mathcal{I}}$. Thus $y \in (\exists S.D')^{\mathcal{I}}$.
- Take (6.11), then $s(y,z) \in M$ and $(\exists S.D')(z) \in M \cap T^{\mathrm{i}^{n-1}}$. Thus $(y,z) \in S^{\mathcal{I}}$ and, by induction, $z \in (\exists S.D')^{\mathcal{I}}$ such that $y \in (\exists S.D')^{\mathcal{I}}$.

6. $D = \forall R.D'$.

$$\begin{aligned} x \in (\forall R.D')^{\mathcal{I}} &\Leftrightarrow x \notin (\exists R.\neg D')^{\mathcal{I}} \\ &\Leftrightarrow \exists R.\neg D'(x) \notin M \text{ (by the previous)} \\ &\Rightarrow \forall R.D'(x) \in M \quad \text{(by rule (6.12))} \end{aligned}$$

$$\begin{aligned} \forall R.D'(x) \in M &\Rightarrow \exists R.\neg D'(x) \notin M \ (M \text{ minimal, and rule (6.12)}) \\ &\Rightarrow x \notin (\exists R.\neg D')^{\mathcal{I}} \quad \text{(by the previous)} \\ &\Rightarrow x \in (\forall R.D')^{\mathcal{I}} \end{aligned}$$

7. $D = (\leq n\ Q.D')$.

$$\begin{aligned} x \in (\leq n\ Q.D')^{\mathcal{I}} &\Leftrightarrow \#\{y \mid (x,y) \in Q^{\mathcal{I}} \wedge y \in D'^{\mathcal{I}}\} \leq n \\ &\Rightarrow \#\{y \mid q(x,y) \in M \wedge D'(y) \in M\} \leq n \text{ (by induction)} \\ &\Rightarrow \#\{y \mid q(x,y) \in M \wedge D'(y) \in M\} \not\geq n+1 \\ &\Rightarrow (\geq n+1\ Q.D')(x) \notin M \ (M \text{ minimal}) \\ &\Rightarrow (\leq n\ Q.D')(x) \in M \text{ (by (6.13))} \end{aligned}$$

$$\begin{aligned} (\leq n\ Q.D')(x) \in M &\Rightarrow (\geq n+1\ Q.D')(x) \notin M \ (M \text{ minimal}) \\ &\Rightarrow \#\{y \mid q(x,y) \in M \wedge D'(y) \in M\} \leq n \text{ (by (6.14))} \\ &\Rightarrow \#\{y \mid (x,y) \in Q^{\mathcal{I}} \wedge y \in D'^{\mathcal{I}}\} \leq n \ (Q \text{ simple}) \\ &\Rightarrow x \in (\leq n\ Q.D')^{\mathcal{I}} \end{aligned}$$

8. $D = (\geq n\ Q.D')$.

$$\begin{aligned} x \in (\geq n\ Q.D')^{\mathcal{I}} &\Leftrightarrow \#\{y \mid (x,y) \in Q^{\mathcal{I}} \wedge y \in D'^{\mathcal{I}}\} \geq n \\ &\Rightarrow \#\{y \mid q(x,y) \in M \wedge D'(y) \in M\} \geq n \ (Q \text{ simple}) \\ &\Rightarrow (\geq n\ Q.D')(x) \in M \text{ (by rule (6.14))} \end{aligned}$$

$$\begin{aligned} (\geq n\ Q.D')(x) \in M &\Leftrightarrow \#\{y \mid q(x,y) \in M \wedge D'(y) \in M\} \geq n \ (M \text{ is min.}) \\ &\Rightarrow \#\{y \mid (x,y) \in Q^{\mathcal{I}} \wedge y \in D'^{\mathcal{I}}\} \geq n \text{ (by induction)} \\ &\Rightarrow x \in (\geq n\ Q.D')^{\mathcal{I}} \end{aligned}$$

We can now check that $\mathcal{I}$ satisfies every terminological axiom $D_1 \sqsubseteq D_2$. Take $x \in D_1^{\mathcal{I}}$ and $x \notin D_2^{\mathcal{I}}$, then we have just shown that $D_1(x) \in M$ and $D_2(x) \notin M$, and by rule (6.1), this gives a contradiction.

Take a role axiom $R_1 \sqsubseteq R_2$. Take $(x,y) \in R_1^{\mathcal{I}}$. Then $r_1(x,y) \in M$ or there is some $S \sqsubseteq^{*} R_1$, $\mathsf{Trans}(S) \in \Sigma$, such that $(x,y) \in (\{(u,v) \mid s(u,v) \in M\})^{*}$. In

the former case, we have by the constraint (6.2), that $r_2(x, y) \in M$ such that $(x, y) \in R_2^{\mathcal{I}}$. In the latter case, we have that $S \sqsubseteq^* R_2$ and $\mathsf{Trans}(S) \in \Sigma$ such that $(x, y) \in R_2^{\mathcal{I}}$.

If $\mathsf{Trans}(R) \in \Sigma$, then $R^{\mathcal{I}}$ should be transitive. Take $(x, y) \in R^{\mathcal{I}}$ and $(y, z) \in R^{\mathcal{I}}$. We distinguish between four cases (we only prove the first one, the others are similar).

- $r(x, y) \in M$ and $(y, z) \in (\{(u, v) \mid s(u, v) \in M\})^*$ for some $S \sqsubseteq^* R$ and $\mathsf{Trans}(S) \in \Sigma$. Thus, there is some $s(y, u_0) \in M, \ldots, s(u_n, z) \in M$. We have that $S \sqsubseteq S_1 \sqsubseteq S_2 \sqsubseteq \ldots \sqsubseteq R$ such that, by constraints (6.2), $s_1(y, u_0) \in M, \ldots, s_1(u_n, z) \in M$, and, finally, $r(y, u_0) \in M, \ldots, r(u_n, z) \in M$. Since $R \sqsubseteq^* R$, $\mathsf{Trans}(R) \in \Sigma$, and $r(x, y) \in M$,$r(y, u_0) \in M$, ..., $r(u_n, z) \in M$ we have that $(x, z) \in R^{\mathcal{I}}$.
- $(x, y) \in (\{(u, v) \mid s_1(u, v) \in M\})^*$ for some $S_1 \sqsubseteq^* R$ and $\mathsf{Trans}(S_1) \in \Sigma$, and $(y, z) \in (\{(u, v) \mid s_2(u, v) \in M\})^*$ for some $S_2 \sqsubseteq^* R$ and $\mathsf{Trans}(S_2) \in \Sigma$.
- $(x, y) \in (\{(u, v) \mid s(u, v) \in M\})^*$ for some $S \sqsubseteq^* R$ and $\mathsf{Trans}(S) \in \Sigma$, and $r(y, z) \in M$.
- $r(x, y) \in M$ and $r(y, z) \in M$.

Remains to check that $C^{\mathcal{I}}$ is not empty. We have that $C(u) \in M$, and we know that this is only possible if $u \in C^{\mathcal{I}}$. □

By the EXPTIME-hardness of $\mathcal{SHIQ}$ satisfiability checking, we have a similar lower bound for satisfiability checking under IWA w.r.t. CoLPs.

**Theorem 6.3.** *Satisfiability checking under IWA w.r.t. CoLPs is* EXPTIME*-hard.*

*Proof.* Satisfiability checking of $\mathcal{SHIQ}$ concept expressions w.r.t. a $\mathcal{SHIQ}$ knowledge base is EXPTIME-complete (Corollary 6.29 in [Tob01]). By Theorem 6.2 and Theorem 6.1, we can polynomially reduce such satisfiability checking to satisfiability checking under IWA w.r.t. CoLPs. □

We have an EXPTIME upper bound for satisfiability checking under IWA w.r.t. CoLPs such that the completeness follows.

**Theorem 6.4.** *Satisfiability checking under IWA w.r.t. CoLPs is* EXPTIME*-complete.*

*Proof.* Membership follows from Theorem 3.39 (pp. 94) and hardness from Theorem 6.3. □

## 6.2 Simulating $\mathcal{ALCHOQ}(\sqcup, \sqcap)$

In this section, we consider the DL that can be obtained from $\mathcal{SHIQ}$ by allowing for nominals ($\mathcal{O}$), role disjunction ($\sqcup$), and role conjunction ($\sqcap$), and

by removing the support for transitive and inverse roles. The resulting DL is $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ (leaving out transitivity yields $\mathcal{ALC}$ instead of $\mathcal{S}$).

Consider the knowledge base from Example 2.24:

$$\begin{aligned} \mathit{Personnel} &\equiv \mathit{Management} \sqcup \mathit{Workers} \sqcup \{\mathit{john}\} \\ \{\mathit{john}\} &\sqsubseteq \exists \mathit{boss}.\mathit{Management} \\ \mathit{Management} &\sqsubseteq (\forall \mathit{take\_orders}.\mathit{Management}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Workers}) \end{aligned}$$

where *boss* is *not* transitive – it was in the previous section. Personnel consists exactly of the managers, workers, and a particular individual *john* where *john* is the boss of some manager.

We show how to translate $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ satisfiability checking w.r.t. $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ knowledge bases to the type of programs where constants are allowed: FoLPs (see Chapter 4), in particular acyclic FoLPs. The above knowledge base translates, similar to the previous section, to the constraints

$$\begin{aligned} &\leftarrow \mathit{Per}(X), \mathit{not}\ (\mathit{Man} \sqcup \mathit{Wor} \sqcup \{\mathit{john}\})(X) \\ &\leftarrow \mathit{not}\ \mathit{Per}(X), (\mathit{Man} \sqcup \mathit{Wor} \sqcup \{\mathit{john}\})(X) \\ &\leftarrow \{\mathit{john}\}(X), \mathit{not}\ (\exists \mathit{boss}.\mathit{Man})(X) \\ &\leftarrow \mathit{Man}(X), \mathit{not}\ ((\forall \mathit{tak}.\mathit{Man}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Wor}))(X) \end{aligned}$$

with the definition of the predicates as follows:

$$\begin{aligned} \mathit{Per}(X) \vee \mathit{not}\ \mathit{Per}(X) &\leftarrow \\ \mathit{Man}(X) \vee \mathit{not}\ \mathit{Man}(X) &\leftarrow \\ \mathit{Wor}(X) \vee \mathit{not}\ \mathit{Wor}(X) &\leftarrow \\ \mathit{boss}(X, Y) \vee \mathit{not}\ \mathit{boss}(X, Y) &\leftarrow \\ \mathit{tak}(X, Y) \vee \mathit{not}\ \mathit{tak}(X, Y) &\leftarrow \\ (\mathit{Man} \sqcup \mathit{Wor} \sqcup \{\mathit{john}\})(X) &\leftarrow \mathit{Man}(X) \\ (\mathit{Man} \sqcup \mathit{Wor} \sqcup \{\mathit{john}\})(X) &\leftarrow \mathit{Wor}(X) \\ (\mathit{Man} \sqcup \mathit{Wor} \sqcup \{\mathit{john}\})(X) &\leftarrow \{\mathit{john}\}(X) \\ (\exists \mathit{boss}.\mathit{Man})(X) &\leftarrow \mathit{boss}(X, Y), \mathit{Man}(Y) \\ ((\forall \mathit{tak}.\mathit{Man}) \sqcap (\geq 3\ \mathit{boss}.\mathit{Wor}))(X) &\leftarrow (\forall \mathit{tak}.\mathit{Man})(X), (\geq 3\ \mathit{boss}.\mathit{Wor})(X) \\ (\forall \mathit{tak}.\mathit{Man})(X) &\leftarrow \mathit{not}\ (\exists \mathit{tak}.\neg \mathit{Man})(X) \\ (\exists \mathit{tak}.\neg \mathit{Man})(X) &\leftarrow \mathit{tak}(X, Y), (\neg \mathit{Man})(Y) \\ (\neg \mathit{Man})(X) &\leftarrow \mathit{not}\ \mathit{Man}(X) \\ (\geq 3\ \mathit{boss}.\mathit{Wor})(X) &\leftarrow \mathit{boss}(X, Y_1), \mathit{boss}(X, Y_2), \\ &\quad\ \mathit{boss}(X, Y_3), \\ &\quad\ \mathit{Wor}(Y_1), \mathit{Wor}(Y_2), \mathit{Wor}(Y_3), \\ &\quad\ Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3 \end{aligned}$$

The only predicate that is not yet defined is $\{\mathit{john}\}$. We define such nominals by facts:

$$\{\mathit{john}\}(\mathit{john}) \leftarrow$$

such that, intuitively, the only $x$ that makes $\{\mathit{john}\}(x)$ true in an open answer set is *john*. The other new constructs in $\mathcal{ALCHOQ}(\sqcup,\sqcap)$, compared to

$\mathcal{SHIQ}$, are role conjunction and disjunction. A role expression $boss \sqcup tak$ can be translated by the rules

$$(boss \sqcup tak)(X, Y) \leftarrow boss(X, Y)$$
$$(boss \sqcup tak)(X, Y) \leftarrow tak(X, Y)$$

and $boss \sqcap tak$ by

$$(boss \sqcap tak)(X, Y) \leftarrow boss(X, Y), tak(X, Y)$$

We define the closure, taking into account nominals and role expressions: the *closure* $clos(C, \Sigma)$ of an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept expression $C$ and an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base $\Sigma$ is the smallest set satisfying the following conditions:

- $C \in clos(C, \Sigma)$,
- for each $C \sqsubseteq D$ an axiom in $\Sigma$ (role or terminological), $\{C, D\} \subseteq clos(C, \Sigma)$,
- for every $D$ in $clos(C, \Sigma)$, we have
  - if $D = \neg D_1$, then $\{D_1\} \subseteq clos(C, \Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  - if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq clos(C, \Sigma)$,
  - if $D = \exists R.D_1$, then $\{R, D_1\} \subseteq clos(C, \Sigma)$,
  - if $D = \forall R.D_1$, then $\{\exists R.\neg D_1\} \subseteq clos(C, \Sigma)$,
  - if $D = (\leq n \ \ Q.D_1)$, then $\{(\geq n + 1 \ \ Q.D_1)\} \subseteq clos(C, \Sigma)$,
  - if $D = (\geq n \ \ Q.D_1)$, then $\{Q, D_1\} \subseteq clos(C, \Sigma)$.

Note that nominals $\{o\}$ are not in the above case analysis; they are considered base objects, such as concept names and role names. Further note that we assumed that $D$ can be both a role and a concept expression (for $\sqcup$ and $\sqcap$).

Formally, we define $\Phi(C, \Sigma)$ to be the following acyclic FoLP, obtained from the $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base $\Sigma$ and the concept expression $C$:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add the constraint

$$\leftarrow C(X), not \ D(X) \tag{6.15}$$

- For each role axiom $R \sqsubseteq S \in \Sigma$, add the constraint

$$\leftarrow R(X, Y), not \ S(X, Y) \tag{6.16}$$

- Next, we distinguish between the types of concept and role expressions that appear in $clos(C, \Sigma)$. For each $D \in clos(C, \Sigma)$:
  - if $D$ is a concept name, add

$$D(X) \vee not \ D(X) \leftarrow \tag{6.17}$$

  - if $D$ is a role name, add

$$D(X, Y) \vee not \ D(X, Y) \leftarrow \tag{6.18}$$

– if $D = \{o\}$, add
$$D(o) \leftarrow \tag{6.19}$$
– if $D = \neg E$, add
$$D(X) \leftarrow not\ E(X) \tag{6.20}$$
– if $D = E \sqcap F$, $D$ a concept expression, add
$$D(X) \leftarrow E(X), F(X) \tag{6.21}$$
– if $D = E \sqcup F$, $D$ a concept expression, add
$$\begin{array}{c} D(X) \leftarrow E(X) \\ D(X) \leftarrow F(X) \end{array} \tag{6.22}$$
– if $D = E \sqcap F$, $D$ a role expression, add
$$D(X, Y) \leftarrow E(X, Y), F(X, Y) \tag{6.23}$$
– if $D = E \sqcup F$, $D$ a role expression, add
$$\begin{array}{c} D(X, Y) \leftarrow E(X, Y) \\ D(X, Y) \leftarrow F(X, Y) \end{array} \tag{6.24}$$
– if $D = \exists Q.E$, add
$$D(X) \leftarrow Q(X, Y), E(Y) \tag{6.25}$$
– if $D = \forall R.E$, add
$$D(X) \leftarrow not\ (\exists R.\neg E)(X) \tag{6.26}$$
– if $D = (\leq n\ Q.E)$, add
$$D(X) \leftarrow not\ (\geq n+1\ Q.E)(X) \tag{6.27}$$
– if $D = (\geq n\ Q.E)$, add
$$D(X) \leftarrow Q(X, Y_1), \ldots, Q(X, Y_n), E(Y_1), \ldots, E(Y_n), \cup_{i \neq j}\{Y_i \neq Y_j\} \tag{6.28}$$

**Theorem 6.5.** *Let $\Sigma$ be an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base and $C$ an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept expression. Then, $\Phi(C, \Sigma)$ is an acyclic FoLP, with a size that is polynomial in the size of $C$ and $\Sigma$.*

*Proof.* Observing the rules in $\Phi(C, \Sigma)$, it is clear that this program is a FoLP. Furthermore, it is acyclic: rules with non-empty head correspond to concept or role expression definitions with positive bodies that contain only concept or role expressions that are structurally smaller. E.g., a concept disjunction $D \sqcup E$ is defined by rules $(D \sqcup E)(X) \leftarrow D(X), E(X)$.

The polynomiality of the size of $\Phi(C, \Sigma)$ can be seen like in the proof of Theorem 6.1. □

**Theorem 6.6.** *An $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ concept expression $C$ is satisfiable w.r.t. an $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ knowledge base $\Sigma$ iff the predicate $C$ is satisfiable w.r.t. $\Phi(C,\Sigma)$.*

*Proof.* For the "only if" direction, assume the concept expression $C$ is satisfiable w.r.t. $\Sigma$, i.e., there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $C^{\mathcal{I}} \neq \emptyset$. We rename the singleton element from $\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ as $o$, which is possible by the unique name assumption. We construct the open answer set $(U, M)$ with $U \equiv \Delta^{\mathcal{I}}$ and $M \equiv \{C(x) \mid x \in C^{\mathcal{I}}, C \in \mathit{clos}(C,\Sigma), C \text{ concept expression}\} \cup \{R(x,y) \mid (x,y) \in R^{\mathcal{I}}, R \in \mathit{clos}(C,\Sigma), R \text{ role expression}\}$.

We have that $(U, M)$ is an open answer set of $\Phi(C,\Sigma)$ that satisfies $C$:

1. Since $C^{\mathcal{I}} \neq \emptyset$ there clearly is an $x \in U$ such that $C(x) \in M$.
2. $M$ is a model of $\Phi(C,\Sigma)^M_U$. We check that every rule in $\Phi(C,\Sigma)^M_U$ is satisfied:
   a) Rules (6.15), (6.16), (6.17), (6.18), (6.20), (6.21), (6.22), (6.25), (6.26) (6.27), and (6.28) can be done as in the proof of Theorem 6.2.
   b) Take a rule $D(o) \leftarrow\in \Phi(C,\Sigma)^M_U$ originating from (6.19). We have that $o \in \{o\}^{\mathcal{I}}$ such that, by definition of $M$, $D(o) \in M$.
   c) Take a rule $E \sqcap F(x,y) \leftarrow E(x,y), F(x,y)$ originating from (6.23) with $E(x,y) \in M$ and $F(x,y) \in M$. Then, $(x,y) \in (E \sqcap F)^{\mathcal{I}} = D^{\mathcal{I}}$ such that $D(x,y) \in M$.
   d) Rules originating from (6.24) are similar as the previous.
3. $M$ is a minimal model of $\Phi(C,\Sigma)^M_U$. Assume not, then there is a model $N$ of $\Phi(C,\Sigma)^M_U$, such that $N \subset M$. We prove that $M \subseteq N$, which leads to a contradiction. Take $l \in M$. We distinguish between the following cases for $l$:
   a) $l = E(x,y)$ *for a role expression* $E \in \mathit{clos}(C,\Sigma)$. We look at the structure of $E$ and prove this by induction:
      i. BASE: $E$ a role name. By rule (6.18), $E(x,y) \leftarrow\in \Phi(C,\Sigma)^M_U$ such that $E(x,y) \in N$.
      ii. INDUCTION HYPOTHESIS: Assume it is proved for role expressions $C_1$ and $C_2$.
      iii. $E = C_1 \sqcap C_2$, then $(x,y) \in (C_1 \sqcap C_2)^{\mathcal{I}}$ such that $(x,y) \in C_1^{\mathcal{I}}$ and $(x,y) \in C_2^{\mathcal{I}}$. Then $C_1(x,y) \in M$ and $C_2(x,y) \in M$. By induction, we have $C_1(x,y) \in N$ and $C_2(x,y) \in N$. With rule (6.23), we then have that $E(x,y) \in N$.
      iv. $E = C_1 \sqcup C_2$. Again per induction, and similar to the previous case.
   b) $l = E(x)$ *for a concept expression* $E \in \mathit{clos}(C,\Sigma)$. This can be done as in the proof of Theorem 6.2.

For the "if" direction. Assume $(U, M)$ is an open answer set of $\Phi(C,\Sigma)$ with $C(u) \in M$. Define an interpretation $\mathcal{I} \equiv (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}} \equiv U$, $A^{\mathcal{I}} \equiv \{x \mid A(x) \in M\}$ for concept names $A$, $R^{\mathcal{I}} \equiv \{(x,y) \mid R(x,y) \in M\}$ for role names $R$ and $\{o\}^{\mathcal{I}} = \{o\}$, for $o \in \mathit{cts}(\Phi(C,\Sigma))$.

We have that $|\{o\}^{\mathcal{I}}| = 1$.

Furthermore, one can show, along the lines of the proof of Theorem 6.2, that

$$x \in D^{\mathcal{I}} \iff D(x) \in M, D \text{ a concept expression}$$

and

$$(x, y) \in D^{\mathcal{I}} \iff D(x, y) \in M, D \text{ a role expression}$$

It is then easy to check that $\mathcal{I}$ satisfies every terminological axiom $D_1 \sqsubseteq D_2$ as well as every role axiom.

It remains to check that $C^{\mathcal{I}}$ is not empty. We have that $C(u) \in M$ and we know that this is only possible, by the above, if $u \in C^{\mathcal{I}}$. □

We investigated decidability for FoLPs (which contain constants) in Chapter 4 by a reduction to finite answer set programming and as such obtained restricted types of FoLPs: local, semi-local, and acyclic FoLPs. This need for restricted FoLPs explains why we did not consider transitivity of roles for simulation (and thus we simulated $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ instead of $\mathcal{SHOQ}(\sqcup, \sqcap)$). To simulate transitivity one needs rules such as (6.11): $D(X) \leftarrow Q(X, Y), D(Y)$ which are not acyclic such that they cannot be rewritten as local FoLPs that have the bounded finite model property.

We did not allow for inverse roles since FoLPs do not allow for inverted predicates (they lead to infinity programs and a reduction to finite answer set programming is no longer possible).

On a final note on the choice of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$: we did not allow for role negations $\neg R$[2] in addition to role conjunction and disjunction. We could simulate such a negation by rules

$$(\neg R)(X, Y) \leftarrow not\ R(X, Y)$$

However, such rules are neither CoLP nor FoLP rules (there is no positive atom that connects $X$ and $Y$) such that a simulation into a (known) decidable fragment of programs under the open answer set semantics is not possible.

We have an EXPTIME lower bound for acyclic FoLPs.

**Theorem 6.7.** *Satisfiability checking w.r.t. acyclic FoLPs is* EXPTIME*-hard.*

*Proof.* Since satisfiability checking of the sublanguage $\mathcal{AL}$ w.r.t. a set of axioms is EXPTIME-complete [BCM$^+$03], we have, with Theorems 6.5 and 6.6, the hardness result. □

**Theorem 6.8.** *Satisfiability checking w.r.t. EFoLPs $(Q, R)$ where $Q \cup R$ is acyclic is* EXPTIME*-hard.*

*Proof.* Satisfiability checking w.r.t. an acyclic FoLP $P$ can be reduced to satisfiability checking w.r.t. to the EFoLP $(P, \emptyset)$ where $P \cup \emptyset$ is acyclic such that with Theorem 6.7 the result follows. □

[2] $(\neg R)^{\mathcal{I}} \equiv \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$.

**Theorem 6.9.** *Satisfiability checking w.r.t. free acyclic EFoLPs is* EXPTIME*-hard.*

*Proof.* Satisfiability checking w.r.t. an acyclic FoLP $P$ can be reduced to satisfiability checking w.r.t. to the free acyclic EFoLP $(P, \emptyset)$ such that with Theorem 6.7 the result follows. □

**Theorem 6.10.** *Satisfiability checking w.r.t. local FoLPs is* EXPTIME*-hard.*

*Proof.* By Theorem 4.24 (pp. 134) and Theorem 4.25, we can reduce satisfiability checking w.r.t. acyclic FoLPs to local FoLPs such that, by the EXPTIME-hardness of the former, also satisfiability checking w.r.t. local FoLPs is EXPTIME-hard. □

**Theorem 6.11.** *Satisfiability checking w.r.t. local EFoLPs is* EXPTIME*-hard.*

*Proof.* Satisfiability checking w.r.t. a local FoLP $P$ can be reduced to satisfiability checking w.r.t. to the local EFoLP $(P, \emptyset)$ such that with Theorem 6.10 the result follows. □

**Theorem 6.12.** *Satisfiability checking w.r.t. semi-local FoLPs is* EXPTIME*-hard.*

*Proof.* Every local FoLP is semi-local, by definition of semi-local FoLPs (see Definition 4.12, pp. 124), such that by Theorem 6.10, the result follows. □

**Theorem 6.13.** *Satisfiability checking w.r.t. semi-local EFoLPs is* EXPTIME*-hard.*

*Proof.* Satisfiability checking w.r.t. a semi-local FoLP $P$ can be reduced to satisfiability checking w.r.t. to the semi-local EFoLP $(P, \emptyset)$ such that with Theorem 6.12 the result follows. □

## 6.3 Simulating $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ with DL-safe Rules

In [MSS04], integrated reasoning of DLs with *DL-safe* rules was introduced. DL-safe rules are unrestricted Horn clauses where only the communication between the DL knowledge base and the rules is restricted; they enable one to express knowledge inexpressible with DLs alone, e.g., triangular knowledge such as [MSS04]

$$BadChild(X) \leftarrow GrChild(X), parent(X, Y), parent(Z, Y), hates(X, Z)$$

saying that a grandchild that hates its sibling is a bad child.

We introduce DL-safe rules like in [MSS04]. For a DL knowledge base $\Sigma$ let $N_C$ and $N_R$ be the concept and role names in $\Sigma$ and $N_P$ is a set of unary

or binary[3] predicate symbols such that $N_C \cup N_R \subseteq N_P$. A *DL-atom* is an atom of the form $A(s)$ or $R(s,t)$ for $A \in N_C$ and $R \in N_R$. A *DL-safe rule* is a rule of the form $a \leftarrow b_1, \ldots, b_n$ where $a$ and $b_i$, $1 \leq i \leq n$, are regular atoms[4] and every variable in the rule appears in a non-DL-atom in the rule body. Note that symbols $o$ from $\{o\}$ may be used as constants in DL-safe rules. A *DL-safe program* is a finite set of DL-safe rules. Let $cts(\Sigma, P)$ be the set of individuals and constants in $\Sigma$ or $P$, i.e.,

$$cts(\Sigma, P) \equiv \{o \mid \{o\} \in \Sigma\} \cup cts(P) \ .$$

We provide an alternative semantics based on DL interpretations like in [HPS04b] rather than on the first-order interpretations used in [MSS04]. However, both semantics are compatible as indicated in [MSS04]. For $(\Sigma, P)$ and an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\Sigma$[5] we extend $\mathcal{I}$ for $N_P$ and $cts(P)$ such that for unary predicates $p \in N_P$, $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, for binary predicates $f \in N_P$, $f^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and $o \in \Delta^{\mathcal{I}}$ for $o \in cts(P)$. Such an extended interpretation is, by definition, an interpretation of $(\Sigma, P)$.

A *binding* for an interpretation $\mathcal{I}$ of $(\Sigma, P)$ is a function $\sigma : vars(P) \cup cts(\Sigma, P) \rightarrow \Delta^{\mathcal{I}}$ with $\sigma(o) \equiv o$ for $o \in cts(\Sigma, P)$; it maps constants/nominals and variables to domain elements. A unary atom $a(s)$ is then true w.r.t. $\sigma$ and $\mathcal{I}$ if $\sigma(s) \in a^{\mathcal{I}}$, and a binary atom $f(s,t)$ is true w.r.t. $\sigma$ and $\mathcal{I}$ if $(\sigma(s), \sigma(t)) \in f^{\mathcal{I}}$. A rule $r$ is satisfied by $\mathcal{I}$ iff for every binding $\sigma$ w.r.t. $\mathcal{I}$ that makes the atoms in the body true, the head is also true. An interpretation of $(\Sigma, P)$ is a model if it is a model of $\Sigma$ and it satisfies every rule in $P$.

In Section 6.2, we reduced $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ satisfiability checking to acyclic FoLP satisfiability checking. We can reduce satisfiability checking of predicates in $N_P$ w.r.t. $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ knowledge bases extended with DL-safe rules to satisfiability checking w.r.t. free acyclic EFoLPs.[6] We provide some intuition with an example.

Take a knowledge base $\Sigma$

$$\exists manf\,.\,Co \sqcap \exists has\_price \sqsubseteq Product \ ,$$

expressing that if something is manufactured in some country and it has a price then it is a product.[7] We have some facts in a DL-safe program $P$ about the world we are considering:

$$\begin{array}{rr} is\_product\_of(p, c_1) \leftarrow & manf(p, japan) \leftarrow \\ is\_product\_of(p, c_2) \leftarrow & Co(japan) \leftarrow \end{array}$$

[3] In [MSS04], $n$-ary predicates are allowed.

[4] No equality is allowed.

[5] We assume, with loss of generality (by the unique name assumption), that for nominals $\{o\}$ in $\Sigma$, there is a $o \in \Delta^{\mathcal{I}}$ such that $\{o\}^{\mathcal{I}} = \{o\}$.

[6] In [MSS04], the $\mathcal{SHOIN}(\mathbf{D})$ DL is considered instead of $\mathcal{ALCHOQ}(\sqcup,\sqcap)$.

[7] $\exists has\_price$ is shorthand for $\exists has\_price.\top$, where $\top^{\mathcal{I}} \equiv \Delta^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. For the formal EFoLP translation, we can assume that $\top$ is equivalent to $A \sqcup \neg A$ for some concept $A$.

saying that $p$ is a product of company $c_1$ and company $c_2$, that $p$ is manufactured in Japan and that Japan is a country. Those facts are vacuously DL-safe since they do not contain variables. Additionally, we have a DL-safe rule in $P$ saying that if a product is a product of 2 companies then those companies are competitors[8],

$$r_1 : competitors(C_1, C_2) \leftarrow Product(P), is\_product\_of(P, C_1), \\ is\_product\_of(P, C_2)$$

Note that this is indeed a DL-safe rule since every variable occurs in a *is_product_of* atom, which is a non-DL-atom in the body of the rule. The only DL-atom in the rule is $Product(P)$. A model $\mathcal{I}$ of $(\Sigma, P)$ is $\mathcal{I} = (\{japan, c_1, c_2, p, x\}, \cdot^{\mathcal{I}})$ with $\cdot^{\mathcal{I}}$ defined as follows:

$$\begin{aligned} Co^{\mathcal{I}} &= \{japan\} \\ Product^{\mathcal{I}} &= \{p\} \\ manf^{\mathcal{I}} &= \{(p, japan)\} \\ has\_price^{\mathcal{I}} &= \{(p, x)\} \\ is\_product\_of^{\mathcal{I}} &= \{(p, c_1), (p, c_2)\} \\ competitors^{\mathcal{I}} &= \{(c_1, c_2)\} \end{aligned}$$

We translate $(\Sigma, P)$ to a free acyclic EFoLP: the DL axiom is translated to the constraint

$$\leftarrow (\exists manf.Co \sqcap \exists has\_price)(X), not\ Product(X)$$

where we introduce predicates corresponding to the concept expressions. Furthermore, we define these predicates by the rules

$$\begin{aligned} (\exists manf.Co \sqcap \exists has\_price)(X) &\leftarrow (\exists manf.Co)(X), (\exists has\_price)(X) \\ (\exists manf.Co)(X) &\leftarrow manf(X, Y), Co(Y) \\ (\exists has\_price)(X) &\leftarrow has\_price(X, Y) \\ Product(X) \vee not\ Product(X) &\leftarrow \\ Co(X) \vee not\ Co(X) &\leftarrow \\ manf(X, Y) \vee not\ manf(X, Y) &\leftarrow \\ has\_price(X, Y) \vee not\ has\_price(X, Y) &\leftarrow \end{aligned}$$

Since DL-safe rules have essentially a first-order interpretation it may be that $(c_1, c_2) \in competitors^{\mathcal{I}}$ for a model $\mathcal{I}$ of $(\Sigma, P)$ without any justification in $\mathcal{I}$: the body of $r_1$ in $P$ does not need to be satisfied in order to have

[8] Actually, to correspond entirely with the desired semantics, we need to indicate that $C_1$ and $C_2$ are different companies. This seems to be not possible with the DL-safe rules in [MSS04], however, it is with EFoLPs using $\neq$.

$(c_1, c_2) \in \mathit{competitors}^{\mathcal{I}}$. The open answer set semantics, however, only deduces $\mathit{competitors}(c_1, c_2)$ in an open answer set if the body of $r_1$ is satisfied in that open answer set, since otherwise the open answer set would not be minimal (one could omit $\mathit{competitors}(c_1, c_2)$ and still have an open answer set).

To solve this, we introduce for each predicate $q$ of a DL-safe program, a free rule: $\mathit{competitor}(C_1, C_2) \vee \mathit{not}\ \mathit{competitor}(C_1, C_2) \leftarrow$ . One has then always a motivation for $\mathit{competitor}(C_1, C_2)$, mimicking the first-order semantics.

Formally, we define the acyclic FoLP $\chi(\Sigma, P)$ like the $\Phi(C, \Sigma)$ from Section 6.2 where $C$ is some arbitrary concept from $\Sigma$ with additionally the following free rules added:

$$p(X) \vee \mathit{not}\ p(X) \leftarrow \ ,$$

for each unary $p \in N_P \setminus N_C$ and

$$p(X, Y) \vee \mathit{not}\ p(X, Y) \leftarrow \ ,$$

for each binary $p \in N_P \setminus N_R$.

**Theorem 6.14.** *Let $(\Sigma, P)$ consist of an $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ knowledge base $\Sigma$ and a DL-safe program $P$. Then, $(\chi(\Sigma, P), P)$ is a free acyclic EFoLP, with a size that is polynomial in the size of $\Sigma$ and $P$.*

*Proof.* As in Theorem 6.5, we have that $\chi(\Sigma, P)$ is an acyclic FoLP. Furthermore, $P$ contains only unary and binary predicates such that $(\chi(\Sigma, P), P)$ is an EFoLP and since $\chi(\Sigma, P)$ contains free rules for every predicate in $P$, $(\chi(\Sigma, P), P)$ is a free acyclic EFoLP.

The polynomiality of the size of $\chi(\Sigma, P)$ can be seen like in the proof of Theorem 6.5. □

**Theorem 6.15.** *Let $\Sigma$ be an $\mathcal{ALCHOQ}(\sqcup,\sqcap)$ knowledge base, $P$ a DL-safe program, and $p$ a unary predicate in $N_P$. Then, $p$ is satisfiable w.r.t. $(\Sigma, P)$ iff $p$ is satisfiable w.r.t. $(\chi(\Sigma, P), P)$.*

*Proof.* From [MSS04] (Theorem 1), we have that $\mathcal{I}$ is a model of $(\Sigma, P)$ iff $\mathcal{I}$ is a model of $(\Sigma, P')$ with $P' \equiv P_{cts(\Sigma,P)}$, which follows from the DL-safeness, i.e., every variable in $P$ must appear in a non-DL atom.

For the "only if" direction, assume $p$ is satisfiable w.r.t. $(\Sigma, P')$, i.e., there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $p^{\mathcal{I}} \neq \emptyset$. We construct the open answer set $(U, M)$ with $U \equiv \Delta^{\mathcal{I}}$ and $M \equiv \{q(x) \mid x \in q^{\mathcal{I}}, q \in (\chi(\Sigma, P'), P'), q \text{ unary}\} \cup \{r(x, y) \mid (x, y) \in r^{\mathcal{I}}, r \in (\chi(\Sigma, P'), P'), r \text{ binary }\}$.

We have that $(U, M)$ is an open answer set of $(\chi(\Sigma, P'), P')$ that satisfies $p$:

1. Since $p^{\mathcal{I}} \neq \emptyset$ there is an $x \in U$ such that $p(x) \in M$.
2. $M$ is a model of $R \equiv R_1 \cup R_2 \equiv \chi(\Sigma, P')^M_U \cup P'^M$. We check that every rule in $R$ is satisfied:
   a) That every rule in $R_1$ is satisfied can be done as in the proof of Theorem 6.6.

b) Take a rule $l \leftarrow \beta \in R_2 = P_{cts(\Sigma,P)}$ originating from $l \leftarrow \beta \in P$.[9] Assume that $\beta \subseteq M$ ($\beta$ does not contain equality atoms). Take $q(s) \in \beta$, then $s \in cts(\Sigma, P)$. Since $q(s) \in M$, we have that $s \in q^{\mathcal{I}}$, such that $q(s)$ is true in $\mathcal{I}$. We can repeat this argument for binary atoms and conclude that the body $\beta$ is true in $\mathcal{I}$. Since $\mathcal{I}$ is a model of $P_{cts(\Sigma,P)}$, we have that $l$ is true in $\mathcal{I}$, from which we can deduce that $l \in M$.

3. $M$ is a minimal model of $R$. Assume not, then there is a model $N$ of $R$, such that $N \subset M$. We prove that $M \subseteq N$, which leads to a contradiction. Take $l \in M$. We distinguish between the following cases for $l$:
   a) $l = E(x, y)$ *for a binary predicate* $E \in preds(\Phi(C, \Sigma))$*, i.e., a role expression in* $\Sigma$. This can be done like in the proof of Theorem 6.6.
   b) $l = E(x)$ *for a unary predicate* $E \in preds(\Phi(C, \Sigma))$*, i.e., a concept expression in* $\Sigma$. This can be done like in the proof of Theorem 6.6.
   c) $l = q(x)$ *for a unary predicate* $q \in N_P \setminus N_C$. By definition, we have that $q(X) \vee not\ q(X) \leftarrow\in \chi(\Sigma, P')$, and thus $q(x) \leftarrow\in \chi(\Sigma, P')^M_U$ and $q(x) \in N$.
   d) $l = q(x, y)$ *for a binary predicate* $q \in N_P \setminus N_R$. Similar as the above.

For the "if" direction, assume $(U, M)$ is an open answer set of $R$ with $p(u) \in M$. Define an interpretation $\mathcal{I} \equiv (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}} \equiv U$, $p^{\mathcal{I}} \equiv \{x \mid p(x) \in M\}$ for unary predicates $p \in N_P$, $q^{\mathcal{I}} \equiv \{(x, y) \mid q(x, y) \in M\}$ for binary predicates $q \in N_P$, [10] and $\{o\}^{\mathcal{I}} = \{o\}$, for nominals $\{o\}$ in $\Sigma$.

We have that $|\{o\}^{\mathcal{I}}| = 1$ and the unique name assumption holds.

Furthermore, one can show, along the lines of the proof of Theorem 6.6, that

$$\begin{aligned} x \in D^{\mathcal{I}} &\iff D(x) \in M, D \text{ a concept expression} \\ (x, y) \in D^{\mathcal{I}} &\iff D(x, y) \in M, D \text{ a role expression} \\ x \in q^{\mathcal{I}} &\iff q(x) \in M, q \text{ unary in } N_P \setminus N_C \\ (x, y) \in q^{\mathcal{I}} &\iff q(x, y) \in M, q \text{ binary in } N_P \setminus N_R \end{aligned}$$

One can check that $\mathcal{I}$ satisfies every terminological axiom $D_1 \sqsubseteq D_2$ as well as every role axiom.

Take a rule $l \leftarrow \beta \in P_{cts(\Sigma,P)}$ with $\beta$ true in $\mathcal{I}$. By the above, we have that $\beta$ is true in $M$ such that, since $l \leftarrow \beta \in P^M_{cts(\Sigma,P)}$, $l$ is true in $M$ and thus $l$ true in $\mathcal{I}$.

It remains to check that $p^{\mathcal{I}}$ is not empty. We have that $p(u) \in M$ and we know that this is only possible, by the above, if $u \in p^{\mathcal{I}}$. □

## 6.4 Simulating $\mathcal{DLR}^{-\{\leq\}}$

The DL $\mathcal{DLR}$ [CDGL97, BCM+03] is a DL that supports $n$-ary relations, instead of just unary and binary ones. Since guarded programs allow for $n$-

[9] Note that $\beta$ does not contain negation as failure by definition of DL-safe programs.
[10] Since $N_C \cup N_R \subseteq N_P$ this also defines the concept and role names of $\Sigma$.

ary predicates, it is interesting to investigate to which extent $\mathcal{DLR}$ can be simulated by guarded programs under an open answer set semantics.

We introduce $\mathcal{DLR}$ as in [BCM+03]. The basic building blocks in $\mathcal{DLR}$ are *concept names* $A$ and *relation names* $\mathbf{P}$ where $\mathbf{P}$ denotes arbitrary $n$-ary relations for $2 \leq n \leq n_{max}$ and $n_{max}$ is a given finite nonnegative integer. Role expressions $\mathbf{R}$ and concept expressions $C$ can be formed according to the following syntax rules:

$$\mathbf{R} \rightarrow \top_n \mid \mathbf{P} \mid (\$i/n : C) \mid \neg\mathbf{R} \mid \mathbf{R_1} \sqcap \mathbf{R_2}$$
$$C \rightarrow \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists[\$i]\mathbf{R} \mid \leq k[\$i]\mathbf{R}$$

where we assume $i$ is between 1 and $n$ in $(\$i/n : C)$, and similarly in $\exists[\$i]\mathbf{R}$ and $\leq k[\$i]\mathbf{R}$ if $\mathbf{R}$ is an $n$-ary relation. Moreover, we assume that the above constructs are well-typed, e.g., $\mathbf{R}_1 \sqcap \mathbf{R}_2$ is defined only for relations of the same arity. The semantics of $\mathcal{DLR}$ is given by interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, $\mathbf{R}^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$ for an $n$-ary relation $\mathbf{R}$, and the following conditions are satisfied ($\mathbf{P}, \mathbf{R}, \mathbf{R}_1$, and $\mathbf{R}_2$ have arity $n$):

$$\begin{aligned}
\top_n^{\mathcal{I}} &\subseteq (\Delta^{\mathcal{I}})^n \\
\mathbf{P}^{\mathcal{I}} &\subseteq \top_n^{\mathcal{I}} \\
(\neg\mathbf{R})^{\mathcal{I}} &= \top_n^{\mathcal{I}} \backslash \mathbf{R}^{\mathcal{I}} \\
(\mathbf{R}_1 \sqcap \mathbf{R}_2)^{\mathcal{I}} &= \mathbf{R}_1^{\mathcal{I}} \cap \mathbf{R}_2^{\mathcal{I}} \\
(\$i/n : C)^{\mathcal{I}} &= \{(d_1, \ldots, d_n) \in \top_n^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\} \\
\top_1^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} &= {C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}} \\
(\exists[\$i]\mathbf{R})^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists(d_1, \ldots, d_n) \in \mathbf{R}^{\mathcal{I}} \cdot d_i = d\} \\
(\leq k[\$i]\mathbf{R})^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid |\{(d_1, \ldots, d_n) \in \mathbf{R}^{\mathcal{I}} \mid d_i = d\}| \leq k\}
\end{aligned}$$

Note that in $\mathcal{DLR}$ the negation of role expressions is defined w.r.t. $\top_n^{\mathcal{I}}$ instead of w.r.t. $\Delta^{\mathcal{I}}$. A $\mathcal{DLR}$ knowledge base consists of terminological axioms and role axioms defining subset relations between concept expressions and role expressions (of the same arity) respectively.

We consider a fragment of $\mathcal{DLR}$, which we call $\mathcal{DLR}^{-\{\leq\}}$, i.e., $\mathcal{DLR}$ without the expressions $\leq k[\$i]\mathbf{R}$ since such expressions cannot be simulated with guarded programs.

*Example 6.16.* Consider the concept expression $\leq 1[\$1]R$ where $R$ is a binary role (this concept expression corresponds to the number restriction $\leq 1R$). One would simulate the $\leq$ again by negation as failure:

$$\leq 1[\$1]R(X) \leftarrow not\ q(X)$$

for some new $q$ with $q$ defined as follows:

$$q(X) \leftarrow R(X, Y_1), R(X, Y_2), Y_1 \neq Y_2$$

However, the latter rule is not a guarded rule – there is no atom that contains $X$, $Y_1$, and $Y_2$ – and it is not loosely guarded since $Y_1$ and $Y_2$ do not appear together in an atom in the body (they appear together in a naf-atom). So, in general, expressing number restrictions such as $\leq k[\$i]\mathbf{R}$ is out of reach for guarded programs.

Define the *closure* $clos(C, \Sigma)$ of a $\mathcal{DLR}^{-\{\leq\}}$ concept expression $C$ and a $\mathcal{DLR}^{-\{\leq\}}$ knowledge base $\Sigma$ as the smallest set satisfying the following conditions:

- $C \in clos(C, \Sigma)$,
- $\top_1 \in clos(C, \Sigma)$,
- for each $C \sqsubseteq D$ an axiom in $\Sigma$ (role or terminological), $\{C, D\} \subseteq clos(C, \Sigma)$,
- for every $D$ in $clos(C, \Sigma)$, $clos(C, \Sigma)$ should contain every subformula that is a concept expression or a role expression,
- if $clos(C, \Sigma)$ contains $n$-ary relation names, it must contain $\top_n$.

Formally, we define $\Phi(C, \Sigma)$ to be the following bound GP, obtained from the $\mathcal{DLR}$ knowledge base $\Sigma$ and the concept expression $C$:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add the constraint

$$\leftarrow C(X), not\ D(X) \tag{6.29}$$

- For each role axiom $\mathbf{R} \sqsubseteq \mathbf{S} \in \Sigma$ where $\mathbf{R}$ and $\mathbf{S}$ are $n$-ary, add the constraint

$$\leftarrow \mathbf{R}(X_1, \ldots, X_n), not\ \mathbf{S}(X_1, \ldots, X_n) \tag{6.30}$$

- For each $\top_n \in clos(C, \Sigma)$, add the free rule

$$\top_n(X_1, \ldots, X_n) \vee not\ \top_n(X_1, \ldots, X_n) \leftarrow \tag{6.31}$$

  Furthermore, for each $n$-ary relation name $\mathbf{P} \in clos(C, \Sigma)$, we add the constraint

$$\leftarrow \mathbf{P}(X_1, \ldots, X_n), not\ \top_n(X_1, \ldots, X_n) \tag{6.32}$$

  Intuitively, the latter rule ensures that $\mathbf{P}^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}}$. We add a constraint

$$\leftarrow not\ \top_1(X) \tag{6.33}$$

  which enforces that for every element $x$ in the universe, $\top_1(x)$ is true in the open answer set. The latter rule ensures that $\top_1^{\mathcal{I}} = \Delta^{\mathcal{I}}$ for the corresponding interpretation. It can be guarded with $X = X$.
- Next, we distinguish between the types of concept and role expressions that appear in $clos(C, \Sigma)$. For each $D \in clos(C, \Sigma)$:

– if $D$ is a concept name, add

$$D(X) \vee \mathit{not}\ D(X) \leftarrow \tag{6.34}$$

– if $\mathbf{D}$ is an $n$-ary relation name, add

$$\mathbf{D}(X_1, \ldots, X_n) \vee \mathit{not}\ \mathbf{D}(X_1, \ldots, X_n) \leftarrow \tag{6.35}$$

– if $D = \neg E$ for a concept expression $E$, add

$$D(X) \leftarrow \mathit{not}\ E(X) \tag{6.36}$$

Note that we assume that such a rule is guarded by $X = X$.

– if $D = \neg\mathbf{R}$ for an $n$-ary role expression $\mathbf{R}$, add

$$D(X_1, \ldots, X_n) \leftarrow \top_n(X_1, \ldots, X_n), \mathit{not}\ \mathbf{R}(X_1, \ldots, X_n) \tag{6.37}$$

Note that if $\mathcal{DLR}$ negation was defined w.r.t. to $(\Delta^{\mathcal{I}})^n$ instead of $\top_n^{\mathcal{I}}$, we would not be able to write the above as a guarded rule.

– if $D = E \sqcap F$ for concept expressions $E$ and $F$, add

$$D(X) \leftarrow E(X), F(X) \tag{6.38}$$

– if $D = \mathbf{E} \sqcap \mathbf{F}$ for $n$-ary role expressions $\mathbf{E}$ and $\mathbf{F}$, add

$$D(X_1, \ldots, X_n) \leftarrow \mathbf{E}(X_1, \ldots, X_n), \mathbf{F}(X_1, \ldots, X_n) \tag{6.39}$$

– if $D = (\$i/n : C)$, add

$$D(X_1, \ldots, X_i, \ldots, X_n) \leftarrow \top_n(X_1, \ldots, X_i, \ldots, X_n), C(X_i) \tag{6.40}$$

– if $D = \exists[\$i]\mathbf{R}$, add

$$D(X) \leftarrow \mathbf{R}(X_1, \ldots, X_{i-1}, X, X_{i+1}, \ldots, X_n) \tag{6.41}$$

**Theorem 6.17.** *Let $\Sigma$ be a $\mathcal{DLR}^{-\{\leq\}}$ knowledge base and $C$ a $\mathcal{DLR}^{-\{\leq\}}$ concept expression. Then, $\Phi(C, \Sigma)$ is a bound GP, with a size that is polynomial in the size of $C$ and $\Sigma$.*

*Proof.* Observing the rules in $\Phi(C, \Sigma)$, it is clear that this program is a GP. Furthermore, every rule contains at most $n_{max}$ variables which is also the bound for the arity of the predicates such that $\Phi(C, \Sigma)$ is a bound GP by Definition 5.75 (pp. 195).

The size of $clos(C, \Sigma)$ is linear in $C$ and $\Sigma$. The size of the GP $\Phi(C, \Sigma)$ is polynomial in the size of $clos(C, \Sigma)$[11] such that the result follows. □

[11] The size of $\Phi(C, \Sigma)$ is polynomial in the size of $clos(C, \Sigma)$ provided the size of $C$ and $\Sigma$ increases such that the $n$ in an added $n$-ary role expression is polynomial in the size of the maximal arity of role expressions in $C$ and $\Sigma$. Although the size of $C$ and $\Sigma$ increases linearly upon adding a relation name $\mathbf{R}$ with arity $2^n$, where $n$ is the maximal arity of relation names in $C$ and $\Sigma$, the size of $\Phi(C, \Sigma)$ increases exponentially: one needs to add, e.g., rules

$$\top_{2^n}(X_1, \ldots, X_{2^n}) \vee \mathit{not}\ \top_{2^n}(X_1, \ldots, X_{2^n}) \leftarrow$$

**Theorem 6.18.** *A $\mathcal{DLR}^{-\{\leq\}}$ concept expression $C$ is satisfiable w.r.t. a $\mathcal{DLR}^{-\{\leq\}}$ knowledge base $\Sigma$ iff the predicate $C$ is satisfiable w.r.t. $\Phi(C, \Sigma)$.*

*Proof.* The proof is along the lines of the proofs of Theorem 6.2 and 6.6. □

Satisfiability checking of a $\mathcal{DLR}$ concept expression w.r.t. a $\mathcal{DLR}$ knowledge base is EXPTIME-complete [CDGL98]. At least for the fragment $\mathcal{DLR}^{-\{\leq\}}$ of $\mathcal{DLR}$, the reduction, via Theorem 6.18, to open answer set programming w.r.t. bound GPs is optimal as satisfiability checking w.r.t. bound GPs is in EXPTIME (with Theorem 5.77 and the fact that every GP is a GgP).

## 6.5 Discussion: OASP vs. DLs

In this section, we discuss some of the advantages and disadvantages of open answer set programming (in particular of the decidable fragments described in the previous chapters) versus description logics in the context of knowledge representation and reasoning.

Using EFoLPs instead of an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base with DL-safe rules on top has the advantage of nonmonotonicity by means of negation as failure in both the FoLP part and the arbitrary program part, whereas both DLs and DL-safe rules are monotonic (DL-safe rules are Horn clauses and thus do not allow for negation as failure).

*Example 6.19.* Add a rule to the company example knowledge base, expressing that if John is not married, he works late at the office:

$$works\_late(john) \leftarrow not\ married(john)$$

Adding such a rule to our knowledge will have the effect that every open answer set includes the literal $works\_late(john)$, i.e., John always works late. However, consecutively adding the newly acquired knowledge that John is actually married with a rule

$$married(john) \leftarrow$$

will make sure that John never works late in answers to our current knowledge. This type of *nonmonotonicity* is one of the main strengths of logic programming paradigms for knowledge representation; it was identified in [BS03] as one of the requirements on a logic for reasoning on the Web. DLs lack this feature and are *monotonic*, e.g., one could try to translate the above rule as the following DL axiom.

$$\neg Married \sqcap \{john\} \sqsubseteq Works\_late \sqcap \{john\}$$

However, interpretations satisfying this axiom have a choice in making John work later or not (there are interpretations where John is married and others where he is not), such that adding that John is married would not invalidate any previously concluded facts.

As was shown in the previous sections, expressive DLs can be simulated using open answer set programming. However, DLs have only a limited set of constructs while CoLPs or EFoLPs have a flexible rule presentation which often allows for a more compact representation of knowledge than would be possible in DLs.

*Example 6.20.* One can represent the knowledge that a team must at least consist of a technical expert, a secretary, and a team leader, where the leader and the technical expert are not the same, by the following rule.

$$team(X) \leftarrow has\_member(X, Y_1), tech(Y_1), has\_member(X, Y_2), \\ secret(Y_2), leader(X, Y_3), Y_1 \neq Y_3$$

Note that this definition of a team does not exclude non-listed members to be part of the team. Moreover, in the presence of other rules with *team* in the head, a team may be qualified by one of those rules. E.g., including a fact $team(007)$, would qualify *007* as a team, regardless of its members. Compared with DL qualified number restrictions $(\geq\ n\ R.C)$ where one indicates that there are more than $n$ $R$-successors that are of type $C$, CoLPs and EFoLPs can constrain different successor relationships (*has_member* and *leader*) instead of just one ($R$). Moreover, they can be very specific about which successors should be different and which ones may be equal ($Y_1$ may be equal to $Y_2$, but should be different from $Y_3$), or to which different types the successors belong (*tech* and *secret*) instead of one type ($C$).

Using inverted predicates, one can rewrite the above rule as the CoLP rule

$$team(X) \leftarrow is\_member\_of^{\mathrm{i}}(X, Y_1), tech(Y_1), is\_member\_of^{\mathrm{i}}(X, Y_2), \\ secret(Y_2), leader(X, Y_3), Y_1 \neq Y_3$$

Intuitively, one can mix inverted predicates $is\_member\_of^{\mathrm{i}}$ with normal predicates *leader*. However, in DL number restrictions $\leq R.C$ you either qualify over a role or an inverted role name.

Representing such *generalized number restrictions* using DLs would be significantly harder while arguably less succinct.

We can explicitly close the domain when using open answer set programming, i.e., only allow reasoning with constants and thus forbidding the use of anonymous elements to make deductions. Indeed, one can, as in [GP93], simply add the rules $H(a) \leftarrow$ for every constant $a$, and a constraint $\leftarrow not\ H(X)$ such that all domain elements must be constants. A similar intervention, restricting the reasoning to individuals, is impossible within standard DLs and was one of the arguments to extend DLs with nonmonotonic tools [DNR02]. One could enforce closed domain reasoning in DLs by working internally with the translation to open answer set programming.

A clear (current) disadvantage of using OASP instead of DLs is the lack of practical algorithms and associated reasoners in the former. Note that practical does not necessarily mean optimal: although the theoretical complexity of, e.g., $\mathcal{SHIQ}$, is EXPTIME-complete, practical tableau algorithms run in 2-NEXPTIME in the worst case [Tob01]. The reason is that the EXPTIME-completeness of $\mathcal{SHIQ}$ satisfiability checking results from a translation to checking non-emptiness of 2ATA (see, e.g., [CGL02]) where the latter is in EXPTIME w.r.t. to the number of states. However, although the number of states of the translated automaton is polynomial in the size of the $\mathcal{SHIQ}$ concept that one is checking (such that one has an EXPTIME upper bound for $\mathcal{SHIQ}$ satisfiability checking as well), the size of the whole automaton is much larger: one defines transition functions for an exponential number of labels. Thus, the automata approach is not practically implementable.

As decidability of CoLPs is also shown by a reduction to 2ATA, we expect a similar effect: good theoretical complexity, bad worst-case reasoners.

Decidability of CoLPs (Chapter 3) and guarded programs (Chapter 5) was shown by a reduction to automata and fixed point logic respectively such that no practical algorithms for these fragments are available. We have, however, an actual algorithm for satisfiability checking w.r.t. local FoLPs, i.e., by a reduction to finite answer set programming, but the 2-EXPTIME$^{\Sigma_2^{\mathrm{P}}}$ complexity again illustrates the very high cost of such algorithms.

## 6.6 Related Work

We distinguish between two lines of research involving the reconciliation of DLs and logic programming paradigms: the approach that tries to simulate DLs reasoning with logic programming by taking a DL knowledge base and reducing it to a program such that both conclude the same regarding satisfiability checking (see Section 6.6.1) and the approach that unites the strengths of DLs and LP by letting them coexist and interact, but without reducing one formalism to the other per se (see Section 6.6.2). We will refer to the former approach as *simulating* and the latter as *integrating*.

Open answer set programming can be considered to be a simulating approach: in Sections 6.1, 6.2, and 6.4, we simulate satisfiability checking in $\mathcal{SHIQ}$, $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, and $\mathcal{DLR}^{-\{\leq\}}$ by CoLPs, acyclic FoLPs, and bound guarded programs respectively. On the other hand, it can also be classified in the integrating approach: [MSS04] described an extension of DLs with DL-safe rules with one associated semantics (which thus falls in the integrating approach). We showed a simulation of this approach using the language of free acyclic EFoLPs in Section 6.3.

In the following sections, we discuss typical examples of each category and highlight the differences with open answer set programming.

### 6.6.1 Simulation of DLs in Rule-based Paradigms

[GHVD03] imposes restrictions on the occurrence of DL constructs in terminological axioms to enable a simulation using Horn clauses, i.e., clauses of the form $a \leftarrow b_1, \ldots, b_n$ where $a$, $b_i$, $1 \leq i \leq n$, are non-equality atoms. Note that the Horn clauses are interpreted under a FOL semantics (no minimality), and as such the mapping is not actually to a LP paradigm but to a rule-based paradigm in the broader sense.

The translation from terminological axioms to Horn clauses maps, e.g., an axiom $C_1 \sqcap C_2 \sqsubseteq D$, for concept names $C_1, C_2$, and $D$, to a Horn clause

$$D(X) \leftarrow C_1(X), C_2(X)$$

and a $D \sqsubseteq C_1 \sqcap C_2$ to

$$C_1(X) \leftarrow D(X)$$
$$C_2(X) \leftarrow D(X)$$

Not all DL constructs can be encoded as Horn clauses. E.g., axioms containing disjunction on the right hand side, as in $D \sqsubseteq C \sqcup D$, universal restriction on the left hand side, or existential restriction on the right hand side are prohibited since Horn clauses cannot represent them. Moreover, neither negation of concept expressions nor number restrictions can be represented. This results in a type of DLs that is less expressive than, e.g., $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ which we simulated.

In [AB02], the DL $\mathcal{ALCQI}$ is successfully translated into a logic program under the answer set semantics. However, to take into account infinite interpretations [AB02] presumes, for technical reasons, the existence of function symbols, which leads, in general, to undecidability of reasoning.

In a first phase [AB02] defines a type of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ equals a fixed finite Herbrand Universe of constants. Thus, instead of modifying the answer set semantics with open domains, [AB02] closes the DL domain. One can then introduce rules

$$top(a) \leftarrow$$

for each $a$ in the Herbrand Universe such that DL interpretations and answer sets speak about the same domain. Concepts $b$ can be introduced by rules

$$b(X) \leftarrow top(X), not\ not\_b(X)$$
$$not\_b(X) \leftarrow top(X), not\ b(X)$$

and similarly for roles. The rest of the constructs can then be defined similarly like we did in the previous sections. Inverse roles are taken care of by rules

$$r(X, Y) \leftarrow p(Y, X)$$

if $R = P^-$ for a role name $P$.

In the second phase, [AB02] takes care of the general case (i.e., with normal, possibly infinite, DL interpretations) by introducing the rules

$$top(a') \leftarrow$$
$$top(f(X)) \leftarrow top(X)$$

which generates an infinite Herbrand Universe, intended to simulate the openness. However, adding function symbols yields undecidable answer set programming in general, and it is not discussed why the obtained translation would be decidable (note that the rule introducing the function symbol is not $\omega$-restricted).

[HMS03] and [Swi04] simulate reasoning in DLs with a LP formalism by using an intermediate translation to first-order clauses. In [HMS03], $\mathcal{SHIQ}$ knowledge bases are reduced to first-order formulas, on which basic superposition calculus is then applied. The result is transformed into a function-free version which is translated to a disjunctive Datalog program. It would be interesting to see whether a similar technique can work to reduce, e.g., guarded programs, to disjunctive Datalog programs under a finite answer set semantics. However, [HMS03]'s technique uses basic superposition calculus which is only applicable to first-order logic. As described in Chapter 5, we can reduce satisfiability checking under the open answer set semantics to satisfiability checking of fixed point logic formulas. As the latter are first-order extensions, one would need an extension of the basic superposition calculus that can cope with this; we are not aware of any such extensions.

[Swi04] translates $\mathcal{ALCQI}$ concept expressions to first-order formulas, grounds them with a finite number of constants, and transforms the result to a logic program. One can use a finite number of constants by the finite model property of $\mathcal{ALCQI}$; in the presence of terminological axioms this is no longer possible since the finite model property is lost.

The approach is interesting since it provides efficient reasoning for a particular DL – its efficiency is comparable to that of DLP [PS99]. We focused in Chapter 4 on finding a particular fragment of programs under the open answer set semantics that could be reduced to finite answer set programming, resulting in acyclic FoLPs. Acyclic FoLPs can simulate satisfiability checking of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept expressions w.r.t. a $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base. From that viewpoint, we also reduce a particular DL to finite answer set programming, the basic difference being that [Swi04] allows for inverse roles and prohibits axioms while we allow for axioms and prohibit inverse roles (and include support for nominals).

In [VBDDS97], the simulation of a DL with acyclic axioms in *open logic programming* (see Section 3.6.4, pp. 106) is shown. More specifically, open logic programming simulates reasoning in the DL $\mathcal{ALCN}$, $\mathcal{N}$ indicating the use of unqualified number restrictions, where terminological axioms consist of non-recursive concept definitions; $\mathcal{ALCN}$ is a subclass of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$.

Essentially, this shows that there are other LP approaches that are just as viable as open answer programming to simulate DLs; the main contribution of

this dissertation is however, the identification of decidable subclasses for open answer set programming. And as such, translations of DLs can be shown to fall in such decidable fragments; the approach of [VBDDS97] requires careful investigation of the SLDNFA proof procedure (which is incomplete in general).

### 6.6.2 Integration of DLs and Rule-based Paradigms

In [LR96], the DL $\mathcal{ALCNR}$ ($\mathcal{R}$ stands for role intersection) is extended with Horn clauses

$$q(\mathbf{Y}) \leftarrow p_1(\mathbf{X}_1), \ldots, p_n(\mathbf{X}_n)$$

where the variables in $\mathbf{Y}$ must appear in $\mathbf{X}_1 \cup \ldots \cup \mathbf{X}_n$; $p_1, \ldots, p_n$ are either concept names, role names, or ordinary predicates not appearing in the DL part, and $q$ is an ordinary predicate. Note that $\mathcal{ALCNR}$ is less general than the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ that we considered. There is no safeness in the sense that every variable must appear in a non-DL atom (i.e., with an ordinary predicate), as it was in, e.g., [MSS04]. The semantics is like in [MSS04]: extended interpretations that satisfy both the DL and clauses part (as FOL formulas).

Query answering is undecidable if recursive Horn clauses are allowed, but decidability can be regained by restricting the DL part or by enforcing that the clauses are role safe (each variable in a role atom $R(X, Y)$ for a role $R$ must appear in a non-DL atom). Note that the latter restriction is less strict than the DL-safeness of [MSS04], where also variables in concept atoms $A(X)$ need to appear in non-DL atoms. On the other hand, [MSS04] allows for the more expressive DL $\mathcal{SHOIN}(\mathbf{D})$, and the head predicates may be DL-atoms as well. In relation with our work: we simulated [MSS04]'s approach for $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ (which is more expressive than $\mathcal{ALCNR}$) in Section 6.3 where we needed the DL-safeness and not just role safeness as in [LR96].

An $\mathcal{AL}$-log [DLNS98] system consists of two subsystems: an $\mathcal{ALC}$ knowledge base and a set of Horn clauses of the above form, where variables in the head must appear in the body, only concept names besides ordinary predicates are allowed in the body (thus no role names), and there is a safeness condition as in [MSS04] saying that every variable appears in a non-DL atom. As argued in [MSS04], the approach in [MSS04] is more general since more expressive DLs than $\mathcal{ALC}$ are allowed, role atoms are allowed and the head predicate does not need to be ordinary. Since we simulated [MSS04]'s approach – if atoms are unary or binary and for the $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ DL – we can also simulate $\mathcal{AL}$-log reasoning.

In [Ros05], an extension of the results in [Ros99], *r-hybrid* knowledge bases are defined. The alphabet of predicates $\mathcal{A}$ is the disjoint union of structural (read *DL*) predicates $\mathcal{A}_P$ and predicates $\mathcal{A}_R$. An r-hybrid knowledge base is a pair $(\mathcal{T}, \mathcal{P})$ where the first component is a DL[12] where $\mathcal{T}$ contains no

[12] Actually, [Ros05] considers the more general case of first-order theories.

predicates from $\mathcal{A}_R$ (intuitively, $\mathcal{A}_R$ are the ordinary predicates from the rule part), and $\mathcal{P}$ is a disjunctive program where each rule $R$ has the form [Ros05]:

$$p_1(\mathbf{X}_1) \vee \ldots \vee p_n(\mathbf{X}_n) \leftarrow r_1(\mathbf{Y}_1), \ldots, r_m(\mathbf{Y}_m), s_1(\mathbf{Z}_1), \ldots, s_k(\mathbf{Z}_k), \\ not\ u_1(\mathbf{W}_1), \ldots, not\ u_h(\mathbf{W}_h)$$

where the $r_i$, $u_i$ are predicates from $\mathcal{A}_R$, i.e., ordinary predicates, and $s_i$ are predicates from $\mathcal{A}_P$, i.e., the $s_i(\mathbf{Z}_i)$ are DL-atoms. Furthermore, each variable in $R$ must occur in one of the $r_i(\mathbf{Y}_i)$'s. The latter condition is exactly the safeness condition from [MSS04]. However, [Ros05] allows for disjunction in the head and negation as failure in the body for non-DL atoms. Moreover, the semantics for r-hybrid knowledge bases differs from [MSS04]'s semantics. Intuitively, due to safeness, one can restrict oneself again to the version of $\mathcal{P}$ that is grounded with the constants in the knowledge base. Next, one can, given an extended interpretation of the DL knowledge base and the program, remove the DL-atoms from $\mathcal{P}$ by applying a reduct-like construction. An extended interpretation is then an *NM-model* if its projection onto the DL concepts and roles satisfies the DL knowledge base, and the projection onto the ordinary predicates is an answer set of the reduced ground program.

This approach is very expressive and decidable for DLs like $\mathcal{SHOIN}(\mathbf{D})$. Moreover, it extends [MSS04] in the sense that an actual answer set semantics is used instead of a first-order one. The approach in [Ros05], using the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ and rules consisting of unary and binary predicates, cannot be reduced to free acyclic EFoLPs since the arbitrary rule component in the latter implements a first-order semantics by imposing that all head predicates should be free. This worked for simulating [MSS04] since the interpretation of the rules is first-order, but will not work for simulating [Ros05].

In [ELST04a, EIST05, ELST04b] *description logic programs* are introduced; atoms in the program component may be *dl-atoms*

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{t})$$

where $S_i$ are concepts or roles, $p_i$ are (ordinary) predicates, $Q(\mathbf{t})$ is a concept inclusion axiom, its negation, a concept $C(t)$ or its negation $\neg C(t)$ for a term $t$, a role $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, $op_i$ is one of three operators that can, intuitively, indicate the augmentation of $S_i$ or $\neg S_i$ in the DL part with the extension of $p_i$ (which is defined by the rules), or a constraining of $S_i$ to $p_i$. The semantics is given by an interpretation that is a subset of the Herbrand Base of the program part grounded with constants or individuals from the combined DL and program. A ground dl-atom $DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{t})$ is true in such an interpretation if, intuitively, adding to the DL the assertions deduced from the $S_i op_i p_i$, the query $Q(\mathbf{t})$ to that modified DL holds. E.g., if $op_i$ means augmenting $S_i$, then assertions $S_i(\mathbf{e})$ are added to the DL knowledge base for each $p_i(\mathbf{e})$ in the interpretation; one can thus query the knowledge in the DL part and each query can also provide the DL with information that the rule part deduced, yielding a bi-directional flow of information.

In [ELST04a], the semantics for the programs is an answer set semantics, while in [ELST04b] a well-founded semantics is investigated. Both discuss the expressiveness and complexity for the expressive DLs $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. In [EIST05], the results of an implementation with experiments were reported.

Finally, SWRL [HSB+04] is a *Semantic Web Rule Language* and extends the syntax and semantics of OWL DL (i.e., $\mathcal{SHOIN}(\mathbf{D})$) with unary/binary Datalog RuleML [Rul], i.e., Horn-like rules. This extension is undecidable [HPS04b].

# 7

# Conclusions and Directions for Future Research

In order to solve the lack of modularity in answer set programming with a closed world assumption, we defined open answer set programming. Although open answer set programming solves the problem with closed-domain reasoning, it is undecidable in general. We showed this by reducing the domino problem – is there a tiling of the infinite plane using a finite set of domino types – to open answer set programming. We subsequently identified 3 families of logic programs for which reasoning under the open answer set semantics is decidable. Those 3 families include different types of syntactically restricted logic programs and were categorized according to 3 different decidability vehicles (two-way alternating tree automata, finite answer set programming, and guarded fixed point logic):

- Reasoning with *Conceptual Logic Programs (CoLPs)* was reduced to checking non-emptiness of two-way alternating tree automata, which yielded an EXPTIME upper bound for reasoning – satisfiability checking – with CoLPs. Predicates in CoLPs are unary or binary, rules have a tree structure, and inverted predicates are allowed. Although restricted, they are still expressive enough for conceptual modeling as we illustrated by translating a particular ORM model. Furthermore, CoLPs can simulate reasoning in the expressive description logic $\mathcal{SHIQ}$. The latter reduction implies EXPTIME-completeness.
- *Forest Logic Programs (FoLPs)* were identified and reasoning w.r.t. several types of FoLPs was reduced to finite answer set programming. FoLPs add support for constants to CoLPs. E.g., for *local* FoLPs this yielded a 2-EXPTIME$^{\Sigma_2^p}$ upper bound. A simulation of the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, which includes nominals, yielded an EXPTIME lower bound.
  Note the significant complexity gap between this EXPTIME-hardness and 2-EXPTIME$^{\Sigma_2^p}$ membership for local FoLPs. Intuitively, this can be understood by looking at an analogous phenomenon in DLs: the DL $\mathcal{SHIQ}$ is EXPTIME-complete, but practical reasoners for $\mathcal{SHIQ}$ are in 2-NEXPTIME [Tob01]. Similarly, the reduction to finite open answer set programming of

local FoLPs can be seen as an effective reasoning algorithm and thus less optimal than theoretically attainable. Future work includes a tightening of the upper complexity bound; note that a reduction to tree automata is not immediately applicable as FoLPs do not have the tree model property. We further extended FoLPs with arbitrary ground rules and showed that such EFoLPs can simulate expressive DLs that are extended with DL-safe rules. This illustrated how an integration of DLs and rules can be embedded in open answer set programming. The arbitrary ground rules may only contain unary and binary predicates. Future work includes checking how to cope with arbitrary $n$-ary predicates in this ground rule part.

- *Guarded programs* allow for $n$-ary predicates, which makes them, in this respect, more expressive than CoLPs or FoLPs. They are, however, less liberal in their use of inequality. Decidability of reasoning with guarded programs is based on a reduction to guarded fixed point logic, an extension of first-order logic with fixed point formulas. This allows to characterize a logic program by a fixed point logic formula where the latter formula can be seen as an extension of Clark's completion. Moreover, the resulting fixed point logic formula can be translated to a Datalog LITE program, i.e., a stratified program with *generalized literals* with a fixed point semantics. This reduces an open answer set semantics to a fixed point (bottom-up) semantics for stratified programs which is remarkable as it shows that negation as failure (under an answer set semantics) can be seen as semantic sugar (one can, e.g., express, the circular knowledge $a(X) \leftarrow \mathit{not}\ b(X)$ and $b(X) \leftarrow \mathit{not}\ a(X)$, which is not stratified).
  We further showed that normal (finite) answer set programming can be reduced to decidable (loosely) guarded open answer set programming such that the latter is an extension of the former. We extended guarded programs with generalized literals which led to the simulation of computation tree logic. Moreover, Datalog LITE can be simulated by such extended guarded programs, showing equivalence of Datalog LITE, (alternation-free) guarded fixed point logic, and guarded programs with generalized literals. Reasoning is 2-EXPTIME-complete in general, and EXPTIME-complete for *bound* guarded programs (with generalized literals). Finally, we showed how the DL $\mathcal{DLR}^{-\{\leq\}}$, which allows for $n$-ary roles, can be simulated by bound guarded programs.

We defined several classes of logic programs, decidable for the open answer set semantics, and illustrated their expressiveness by simulations of several expressive DLs (possibly with DL-safe rules). Moreover, we have native support for nonmonotonicity by means of negation as failure, a feature that is missing in standard DLs. Additionally, the rule-based syntax allows for a more succinct expression of knowledge than the more rigid DL syntax.

The DL $\mathcal{SHOIQ}$ has support for both nominals ($\mathcal{O}$) and inverse roles ($\mathcal{I}$). On the other hand, CoLPs contain inverted predicates but no constants and vice versa for FoLPs. It is interesting to check whether one can allow for both

inverted predicates and constants and still have decidable reasoning. Note that a program with inverted predicated cannot be reduced to finite answer set programming (like we did with local FoLPs) as inverted predicates may lead to programs that have only infinite open answer sets. A program with constants cannot be reduced to a tree automaton (like we did with CoLPs) as constants, induce, at best, forest models instead of tree models. So, the combination of inverted predicates and constants seems to be not trivial.

We plan to look into the correspondence with Datalog and use decidability results for Datalog satisfiability checking, as, e.g., in [HMSS01], 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 $\mu$GF, and Datalog LITE. Intuitively, Datalog LITE is not expressive enough to simulate normal $\mu$GF since such $\mu$GF formulas could contain negated fixed point variables, which would result in a non-stratified program when translating to Datalog LITE [GGV02]. 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 $\mu$GF and normal $\mu$GF are equivalent, i.e., whether the alternation hierarchy can always be collapsed.

We simulated $\mathcal{DLR}^{-\{\leq\}}$ with bound guarded programs. However, we did not need to use the full power of bound guarded programs. E.g., $\mathcal{DLR}^{-\{\leq\}}$ does not support nominals while bound guarded programs allow for constants. We could extend $\mathcal{DLR}^{-\{\leq\}}$ with nominals, resulting in the, to the best of our knowledge, yet unexplored DL, $\mathcal{DLRO}^{-\{\leq\}}$. Furthermore, while normally only nominals $\{o\}$ are allowed, we could allow for *general* nominals $\{(o_1, o_2, \ldots, o_n)\}$, i.e., an $n$-ary tuple of individuals. The translation to bound guarded programs would contain then rules

$$\{(o_1, o_2, \ldots, o_n)\}(o_1, o_2, \ldots, o_n) \leftarrow$$

defining the particular *role nominal* $\{(o_1, o_2, \ldots, o_n)\}$ as an $n$-ary predicate that is only true for the tuple $(o_1, o_2, \ldots, o_n)$.

Equilibrium logic is a nonmonotonic system for propositional logic, defined in [Pea96]. The semantics of propositional formulas is given by *equilibrium models*. Interestingly, for logic programs, equilibrium models coincide with answer sets. In [PV04], equilibrium logic is extended for first-order logic. Since open answer set programming extends answer set programming by open domains, or, equivalently, considers those first-order interpretations $(U, M)$ for which $M$ is an answer set of $P$ grounded with $U$, it would be interesting to see whether first-order equilibrium logic is a generalization of open answer set programming. In particular, whether a first-order equilibrium model of a logic program (with variables) corresponds to an open answer set of that program.

Moreover, since first-order equilibrium logic is undecidable in general, one could attempt to identify decidable fragments of first-order equilibrium logic by using a translation to guarded fixed point logic. This would require the characterization of an equilibrium model as a fixed point of an operator defined w.r.t. some reduct of a general first-order formula (instead of a logic program with variables like for open answer set programming).

In [HV02], we extend the DL $\mathcal{SHOQ}(\mathbf{D})$ with a preference order. This order indicates whether a certain axiom is more preferred than another and may defeat the meaning of that axiom. For example, we could be tempted to assume that, in general, movie stars are bright people. If we came to the discovery that movie stars residing in Hollywood are actually not that clever, we would not be able to retain this information consistently. However by *defeating* the rule saying that movie stars are clever with the rule saying they are not if they are Hollywood stars, we can still retain a consistent knowledge base.

In addition to adding a preference order on axioms, implementing the notion of defeat, we introduce in [HV02] an order on the models of such a description logic knowledge base, taking into account the order on the axioms. Nonmonotonicity is then introduced by preferring models that defeat as few axioms as possible, and if defeat cannot be avoided, we select those models that defeat less preferred axioms.

The ideas applied in [HV02] for defeasible description logics, were first defined in the context of answer set programming in [VNV02]. In *preferred answer set programming* rules may be defeated and a preference order on rules induces a preference on the extended answer sets.

Given the correspondence between DLs and open answer set programming (i.e., the simulation of the former with the latter) and the fact that open answer set programming extends normal finite answer set programming, a unifying *preferred open answer set programming* would have several desired features of different kinds of knowledge representation formalisms: open domain reasoning, flexible rule-based representation, nonmonotonicity, and resolution of conflicts using preference.

# References


[AB02] G. Alsaç and C. Baral. Reasoning in Description Logics using Declarative Logic Progamming. Technical report, Arizona State University, 2002. [4, 225, 226]

[ABC00] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs using Logic Programs with Exceptions. In *Proc. of the 4th International Conference on Flexible Query Answering Systems*, pages 27–41. Springer, 2000. [44]

[AE01] P. C. Attie and E. A. Emerson. Synthesis of Concurrent Programs for an Atomic Read/Write Model of Computation. *ACM Trans. Program. Lang. Syst.*, 23(2):187–242, 2001. [55]

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. [3, 180, 184]

[ANB98] H. Andréka, I. Németi, and J. Van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. of Philosophical Logic*, 27(3):217–274, 1998. [14, 82, 154]

[Bar03] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003. [44, 48, 133, 136, 140, 163]

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003. [4, 5, 49, 122, 213, 218, 219]

[Ben97] J. Van Benthem. Dynamic Bits and Pieces. In *ILLC research report*. University of Amsterdam, 1997. [14, 153]

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives of Mathematical Logic. Springer, 1997. Second printing (Universitext) 2001. [27, 30, 67]

[BGH01] S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not Enough. In *Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, pages 151–159. CEUR, 2001. [49]

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001. [3, 49]

[Bon03] P.A. Bonatti. Finitary Open Logic Programs. In *Answer Set Programming: Advances in Theory and Implementation (ASP03)*, pages 84–97. Volume 78 of CEUR Proc., 2003. [2, 108, 109, 111]

[Bon04] P. A. Bonatti. Reasoning with Infinite Stable Models. *Artificial Intelligence*, 156:75–111, 2004. [105, 106]

[Bra77] R.J. Brachman. What's in a Concept: Structural Foundations for Semantic Networks. *Int. Journal of Man-Machine Studies*, 9(2):127–152, 1977. [49]

[BS85] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985. [49]

[BS00] F. Baader and U. Sattler. Tableau Algorithms for Description Logics. In *Proc. of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, volume 1847 of *LNAI*, pages 1–18. Springer, 2000. [49]

[BS03] F. Bry and S. Schaffert. An Entailment Relation for Reasoning on the Web. In *Proc. of Rules and Rule Markup Languages for the Semantic Web*, LNCS, pages 17–34. Springer, 2003. [222]

[BvHH$^+$] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. [4, 49]

[Cac02] T. Cachat. Two-Way Tree Automata Solving Pushdown Games. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 303–317. Springer, 2002. [41]

[CDGL97] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive Query Containment in Description Logics with $n$-ary Relations. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 5–9, 1997. [218]

[CDGL98] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998. [222]

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. [17, 54, 55]

[CGL02] D. Calvanese, G. De Giacomo, and M. Lenzerini. 2ATAs make DLs easy. In *Proc. of the 2002 Description Logic Workshop (DL'02)*, 2002. [6, 224]

[CH82] A. K. Chandra and D. Harel. Horn Clauses and the Fixpoint Query Hierarchy. In *Proc. of PODS '82*, pages 158–163. ACM Press, 1982. [14, 147]

[CI05] F. Calimeri and G. Ianni. External Sources of Computation for Answer Set Solvers. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005)*, number 3662 in LNAI, pages 105–118. Springer, 2005. [111]

[Cla87] K. L. Clark. Negation as Failure. In *Readings in Nonmonotonic Reasoning*, pages 311–325. Kaufmann, 1987. [107, 152]

[Col] A Disjunctive Encoding of 3-Colorability. `http://www.dbai.tuwien.ac.at/proj/dlv/examples/3col`. [44]

[dBPLF05] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. In

*Proc. of the International World Wide Web Conference (WWW 2005)*. ACM, 2005. [54]

[DDS98] M. Denecker and D. De Schreye. SLDNFA: An Abductive Procedure for Abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998. [108]

[DEGV01] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001. [48, 94, 136, 140, 163]

[DLNS98] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998. [4, 227]

[DNR02] F. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Trans. Comput. Logic*, 3(2):177–225, 2002. [223]

[DvHB+00] S. Decker, F. van Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, and S. Melnik. The Semantic Web - on the respective roles of XML and RDF. *IEEE Internet Computing*, 2000. [49]

[DVV99] M. De Vos and D. Vermeir. Choice Logic Programs and Nash Equilibria in Strategic Games. In *Proc. of Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 266–276. Springer, 1999. [44]

[EC82] E. A. Emerson and E. M. Clarke. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Sciene of Computer Programming*, 2(3):241–266, 1982. [15, 55, 56]

[EFF+04] T. Eiter, W. Faber, M. Fink, G. Pfeifer, and S. Woltran. Complexity of Model Checking and Bounded Predicate Arities for Non-ground Answer Set Programming. In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*, pages 377–387. Morgan Kaufmann, 2004. [136]

[EFL+00] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. In *Proc. of the First International Conference on Computational Logic (CL 2000)*, volume 1861 of *LNCS*, pages 807–821. Springer, 2000. [44]

[EFL+02] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. The DLV$^k$ planning system. In JELIA 2002 [JEL02], pages 541–544. [44]

[EFLP99] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The Diagnosis Frontend of the DLV System. *AI Communications*, 12(1-2):99–111, 1999. [44]

[EFLP00] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. *Logic-Based Artificial Intelligence*, pages 79–103, 2000. [44]

[EFST00] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on Updates of Logic Programs. In *Proc. of European Conference on Logic in Artificial Intelligence (JELIA 2002)*, volume 1919 of *LNAI*, pages 2–20. Springer, 2000. [44]

[EG93] T. Eiter and G. Gottlob. Complexity Results for Disjunctive Logic programming and Application to Nonmonotonic Logics. In *Proceedings of the 1983 International Logic Programming Symposium*, pages 266–279, Vancouver, 1993. MIT Press. [48]

[EG97] T. Eiter and G. Gottlob. Expressiveness of Stable Model Semantics for Disjunctive Logic Programs with Functions. *Journal of Logic Programming*, 33(2):167–178, 1997. [104]

[EG05] T. Eiter and G. Gottlob. Reasoning Under Minimal Upper Bounds in Propositional Logic. Technical Report INFSYS RR-1843-05-06, TU Wien, 2005. [66]

[EH82] E. A. Emerson and Joseph Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In *Proc. of the fourteenth annual ACM symposium on Theory of Computing*, pages 169–180. ACM Press, 1982. [17, 54]

[EIST05] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Nonmonotonic Description Logic Programs: Implemenation and Experiments. In *11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2004)*, number 3452 in LNAI, pages 511–527. Springer, 2005. [4, 228, 229]

[EJ00] E. A. Emerson and C. S. Jutla. The Complexity of Tree Automata and Logics of Programs. *SIAM J. Comput.*, 29(1):132–158, 2000. [44]

[ELST04a] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with DLs for the Semantic Web. In *Proc. of 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 141–151. Morgan Kaufmann, 2004. [4, 228, 229]

[ELST04b] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-Founded Semantics for Description Logic Programs in the Semantic Web. In *Proc. of 3th International Workshop on Rules and Rule Markup Languages for the Semantic Web (RULEML 2004)*, number 3323 in LNCS, pages 81–97. Springer, 2004. [228, 229]

[Eme90] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers B.V., 1990. [17, 54, 55, 56, 122]

[FH91] F. and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Technical Report RR-91-10, TU Dresden, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1991. [50]

[FHvH+00] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. In R. Dieng et al., editor, *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, LNAI. Springer-Verlag, 2000. [49]

[FL05] P. Ferraris and V. Lifschitz. Mathematical Foundations of Answer Set Programming. In *We Will Show Them! Essays in Honour of Dov Gabbay*, `http://www.cs.utexas.edu/users/vl/papers/mfasp.ps`, 2005. [48]

[Flu99] J. Flum. On the (Infinite) Model Theory of Fixed-point Logics. *Models, Algebras, and Proofs*, pages 67–75, 1999. [56]

[FvHH+01] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001. [49]

[GGV02] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3(1):1–35, 2002. [14, 15, 16, 17, 180, 181, 182, 183, 188, 189, 233]

[GHO02] E. Grädel, C. Hirsch, and M. Otto. Back and Forth Between Guarded and Modal Logics. *ACM Transactions on Computational Logic*, 3:418–463, 2002. [155]

[GHVD03] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003. [4, 225]

[GL88] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of International Conference on Logic Programming (ICLP 1988)*, pages 1070–1080. MIT Press, 1988. [1, 3, 13, 17, 44, 186]

[GP93] M. Gelfond and H. Przymusinska. Reasoning in Open Domains. In *Logic Programming and Non-Monotonic Reasoning*, pages 397–413. MIT Press, 1993. [2, 103, 104, 223]

[Grä99] E. Grädel. On the Restraining Power of Guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999. [14, 16, 188, 189]

[Grä02a] E. Grädel. Guarded Fixed Point Logic and the Monadic Theory of Trees. *Theoretical Computer Science*, 288:129–152, 2002. [57, 172]

[Grä02b] E. Grädel. Model Checking Games. In *Proceedings of WOLLIC 02*, volume 67 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002. [195]

[GW99] E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 45–54. IEEE Computer Society, 1999. [13, 14, 56, 57, 58, 153, 154, 160, 166, 179, 188, 196]

[Hal01] T. Halpin. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers, 2001. [4, 8, 96, 101]

[HFB+00] I. Horrocks, D. Fensel, J. Boekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. The Ontology Inference Layer OIL, 2000. [49]

[HM01] V. Haarslev and R. Moller. Description of the RACER System and its Applications. In *Proc. of Description Logics 2001*, 2001. [49]

[HMS03] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. FZI-Report 1-8-11/03, Forschungszentrum Informatik (FZI), 2003. [4, 226]

[HMSS01] A. Halevy, I. Mumick, Y. Sagiv, and O. Shmueli. Static Analysis in Datalog Extensions. *Journal of the ACM*, 48(5):971–1012, 2001. [233]

[Hor98] I. Horrocks. The FaCT system. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in LNAI, pages 307–312. Springer, 1998. [49]

[HPS04a] I. Horrocks and P. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *J. of Web Semantics*, 2004. To Appear. [54]

[HPS04b] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM, 2004. [4, 215, 229]

[HR00] M. R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000. [55, 189]

[HS98] I. Horrocks and U. Sattler. A Description Logic with Transitive and Converse Roles and Role Hierarchies. LTCS-Report 98-05, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1998. [9]

[HS01] I. Horrocks and U. Sattler. Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic. In *Proc. of IJCAI'01*, pages 199–204. Morgan Kaufmann, 2001. [115]

[HS03] J. Hladik and U. Sattler. A Translation of Looping Alternating Automata to Description Logics. In *Proc. of CADE-19*, volume 2741 of *LNAI*. Springer, 2003. [121]

[HSB+04] I. Horrocks, P. F. Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule language Combining OWL and RuleML, May 2004. [4, 229]

[HST99] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proc. of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in LNCS, pages 161–180. Springer, 1999. [7, 49, 53]

[HV02] S. Heymans and D. Vermeir. A Defeasible Ontology Language. In Robert Meersman and Zahir Tari et al., editors, *Confederated International Conferences: CoopIS, DOA, and ODBASE 2002*, number 2519 in Lecture Notes in Computer Science, pages 1033–1046. Springer, 2002. [234]

[HV03a] S. Heymans and D. Vermeir. Integrating Description Logics and Answer Set Programming. In *International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, number 2901 in LNCS, pages 146–159. Springer, December 2003. [20]

[HV03b] S. Heymans and D. Vermeir. Integrating Ontology Languages and Answer Set Programming. In *Fourteenth International Workshop on Database and Expert Systems Applications*, pages 584–588. IEEE Computer Society, 2003. [20]

[HV03c] S. Heymans and D. Vermeir. Integrating Semantic Web Reasoning and Answer Set Programming. In *Answer Set Programming: Advances in Theory and Implementation (ASP03)*, pages 194–208. Volume 78 of CEUR Proceedings, 2003. [20]

[HVNV04] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Semantic Web Reasoning with Conceptual Logic Programs. In Grigoris Antoniou and Harold Boley, editors, *3th International Workshop on Rules and Rule Markup Languages for the Semantic Web*, number 3323 in LNCS, pages 113–127. Springer, 2004. [20]

[HVNV05a] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Guarded Open Answer Set Programming. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005)*, number 3662 in LNAI, pages 92–104, Diamante, Italy, September 2005. Springer. [20]

[HVNV05b] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic

Programs. In A. Goméz-Pérez and J. Euzenat, editors, *2nd European Semantic Web Conference (ESWC 2005)*, number 3532 in LNCS, pages 392–407, Heraklion, Greece, 2005. Springer. [20]

[HVNV06] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Guarded Open Answer Set Programming with Generalized Literals. In J. Dix and S.J. Hegner, editors, *Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006)*, number 3861 in LNCS, pages 179–200. Springer, 2006. [20]

[IS98] K. Inoue and Ch. Sakama. Negation as Failure in the Head. *Journal of Logic Programming*, 35(1):39–78, 1998. [48, 132]

[JEL02] *Proc. of European Conference on Logic in Artificial Intelligence (JELIA 2002)*, volume 2424 of *LNAI*. Springer, 2002. [237, 244]

[JM02] M. Jarrar and R. Meersman. Formal Ontology Engineering in the DOGMA Approach. In *Proc. of CoopIS/DOA/ODBASE*, volume 2519 of *LNCS*, pages 1238–1254. Springer, 2002. [4]

[Koz83] D. Kozen. Results on the Propositional $\mu$-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. [15, 56]

[Kun87] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4(4):289–308, 1987. [103]

[Lif02] V. Lifschitz. Answer Set Programming and Plan Generation. *Artificial Intelligence*, 138(1-2):39–54, 2002. [11, 44, 47]

[LL03] J. Lee and V. Lifschitz. Loop Formulas for Disjunctive Logic Programs. In *Proc. of ICLP 2003*, volume 2916 of *LNCS*, pages 451–465. Springer, 2003. [14, 152]

[LPF] N. Leone, G. Pfeifer, and W. Faber. The DLV Project – A Disjunctive Datalog System. `http://www.dbai.tuwien.ac.at/proj/dlv/`. [44]

[LPV01] V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001. [64]

[LR96] A. Y. Levy and M. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *Proc. of ECAI'96*, pages 323–327, 1996. [4, 227]

[LRS97] N. Leone, P. Rullo, and F. Scarcello. Disjunctive Stable Models: Unfounded sets, Fixpoint Semantics, and Computation. *Information and Computation*, 135(2):69–112, 1997. [44]

[LS99] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, February 1999. [49]

[LS00] C. Lutz and U. Sattler. Mary Likes all Cats. In F. Baader and U. Sattler, editors, *Proc. of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS, pages 213–226. RWTH Aachen, 2000. [6]

[LT84] J. Lloyd and R. Topor. Making Prolog More Expressive. *J. Log. Program.*, 1(3):225–240, 1984. [16]

[LZ02] F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. of 18th National Conference on Artificial Intelligence*, pages 112–117. AAAI, 2002. [14]

[Min85] M. Minsky. A Framework for Representing Knowledge. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 245–262. Kaufmann, Los Altos, CA, 1985. [49]

[Mos74] Y.N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974. [56]

[MS87] D.E. Muller and P.E. Schupp. Alternating Automata on Infinite Trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987. [39]

[MSS04] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *Proc. of International Semantic Web Conference (ISWC 2004)*, number 3298 in LNCS, pages 549–563. Springer, 2004. [4, 12, 214, 215, 216, 217, 224, 227, 228]

[MT91] V. W. Marek and M. Truszczyński. Autoepistemic Logic. *Journal of the ACM*, 38(3):588–619, 1991. [48]

[MW84] Z. Manna and P. Wolper. Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984. [55]

[NS96] I. Niemelä and P. Simons. Efficient Implementation of the Well-founded and Stable Model Semantics. In *Proc. of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 289–303, 1996. [44]

[NS97] I. Niemelä and P. Simons. SMODELS - An Implementation of the Stable Model and Well-founded Semantics for Normal Logic Programs. In *Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997)*, volume 1265 of *LNAI*, pages 420–429, 1997. [44]

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. [21, 23, 24, 25, 26, 31, 163]

[Pea96] D. Pearce. A New Logical Characterisation of Stable Models and Answer Sets. In *Proc. of Nonmonotonic Extensions of Logic Programming*, number 1216 in LNCS, pages 57–70. Springer, 1996. [234]

[PP90] H. Przymusinska and T. Przymusinski. Semantic Issues in Deductive Databases and Logic Programs. In R. Banerji, editor, *Formal Approaches to Artificial Intelligence: A Sourcebook*. North-Holland, 1990. [2]

[PS99] P. Patel-Schneider. DLP. In *Proc. of the 1999 International Workshop on Description Logics (DL'99)*, volume 22 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999. [226]

[PV04] D. Pearce and A. Valverde. Towards a First Order Equilibrium Logic for Nonmonotonic Reasoning. In J.J. Alferes and J. Leite, editors, *9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, number 3229 in LNAI, pages 147–160. Springer, 2004. [234]

[Ros99] R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. of DL'99*, pages 160–164, 1999. [227]

[Ros05] R. Rosati. On the Decidability and Complexity of Integrating Ontologies and Rules. *Journal of Web Semantics*, 3(1), 2005. [4, 227, 228]

[Rul] The Rule Markup Initiative. `http://www.ruleml.org`. [229]

[RWRR01] A. L. Rector, C. Wroe, J. Rogers, and A. Roberts. Untangling Taxonomies and Relationships: Personal and Practical Problems in Loosely Coupled Development of Large Ontologies. In *K-CAP 2001: Proc. of the International Conference on Knowledge Capture*, pages 139–146, New York, NY, USA, 2001. ACM Press. [49]

[SC85] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985. [17, 54]

[Sch93] J. Schlipf. Some Remarks on Computability and Open Domain Semantics. In *Proc. of the Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*, 1993. [3, 104]

[Sch95] J. Schlipf. Complexity and Undecidability Results for Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):257–288, 1995. [104]

[Sim] P. Simons. SMODELS Homepage. `http://www.tcs.hut.fi/Software/smodels/`. [44, 113]

[SN99] T. Soininen and I. Niemelä. Developing a Declarative Rule Language for Applications in Product Configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL 1999)*, number 1551 in LNCS, pages 305–319. Springer, 1999. [44]

[SNTS01] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing Configuration Knowledge with Weight Constraint Rules. In *Proc. of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge*, 2001. [44]

[Swi04] T. Swift. Deduction in Ontologies via Answer Set Programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proc. of LPNMR 2004*, volume 2923 of *LNCS*, pages 275–288. Springer, 2004. [4, 226]

[SWM04] M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2004. [50]

[Syr01] T. Syrjänen. Omega-Restricted Logic Programs. In *Proc. of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *LNCS*, pages 267–279. Springer, 2001. [111, 113]

[Syr04] T. Syrjänen. Cardinality Constraint Programs. In *Proc. of JELIA'04*, pages 187–200. Springer, 2004. [179]

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955. [57]

[Tho90] W. Thomas. Automata on Infinite Objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990. [34, 35, 36]

[Tob01] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001. [53, 163, 202, 208, 224, 231]

[UG96] M. Uschold and M. Grüninger. Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996. [4, 49]

[Var97] M. Y. Vardi. Why is Modal Logic so Robustly Decidable? Technical Report TR97-274, Rice University, April 12, 1997. [77]

[Var98] M. Y. Vardi. Reasoning about the Past with Two-Way Automata. In *Proceedings of the 25th Int. Coll. on Automata, Languages and Programming (ICALP '98)*, pages 628–641. Springer, 1998. [5, 35, 40, 41, 42, 44, 83, 120]

[VB97] K. Van Belleghem. *Open Logic Programming as a Knowledge Representation Language for Dynamic Problem Domains*. PhD thesis, Depart-

ment of Computer Science, K.U. Leuven, Leuven, Belgium, December 1997. [2, 106, 107, 108]

[VBDDS97] K. Van Belleghem, M. Denecker, and D. De Schreye. A Strong Correspondence between DLs and Open Logic Programming. In *Proc. of International Conference on Logic Programming (ICLP 1997)*, pages 346–360. MIT Press, 1997. [4, 226, 227]

[vEK76] M. H. van Emden and R. A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the Association for Computing Machinery*, 23(4):733–742, 1976. [66, 168]

[VNV02] D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In JELIA 2002 [JEL02], pages 432–443. [44, 234]

[VNV03] D. Van Nieuwenborgh and D. Vermeir. Ordered Diagnosis. In *Proc. of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2003)*, volume 2850 of *LNAI*, pages 244–258. Springer, 2003. [44]

[VRS91] A. Van Gelder, K. A. Ross, and J. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):619–649, 1991. [102, 108]

[VS93] A. Van Gelder and J. Schlipf. Commonsense Axiomatizations for Logic Programs. *Journal of Logic Programming*, 17:161–195, 1993. [2, 102, 103]

# Index