Ann Math Artif Intell (2006) 47: 103–137
DOI 10.1007/s10472-006-9030-5

Conceptual logic programs

Stijn Heymans · Davy Van Nieuwenborgh · Dirk Vermeir

Received: 1 February 2006 / Accepted: 23 June 2006 /
Published online: 13 September 2006
© Springer Science + Business Media B.V. 2006

Abstract *Open answer set programming* (OASP) solves the lack of modularity in closed world answer set programming 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 *conceptual logic programs* (CoLPs). CoLPs are logic programs with unary and binary predicates (possibly inverted) where rules have a tree shape. Decidability of satisfiability checking of predicates w.r.t. CoLPs is shown by a reduction to non-emptiness checking of two-way alternating tree automata. We illustrate the expressiveness of CoLPs by simulating the description logic $\mathcal{SHIQ}$. CoLPs thus 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 description logics paradigm (decidable open domain reasoning).

Keywords answer set programming · open domains · description logics

Mathematics Subject Classifications (2000) 68T27 · 68T30 · 68N17

S. Heymans (✉)
Digital Enterprise Research Institute (DERI), University of Innsbruck,
Innsbruck, Austria
e-mail: stijn.heymans@deri.org

D. Van Nieuwenborgh · D. Vermeir
Department of Computer Science, Vrije Universiteit Brussel, VUB Pleinlaan 2,
B1050 Brussels, Belgium

D. Van Nieuwenborgh
e-mail: dvnieuwe@vub.ac.be

D. Vermeir
e-mail: dvermeir@vub.ac.be

1 Introduction

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 considers the constants that are specified in the logic program. Take, for example, a logic program consisting of the following rules

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

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 [18], will then *ground* the program with all constants in the program, resulting in the *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 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 [43]. In [43], 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.

Gelfond and Przymusinska [19] 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*. We extend the principle of k-belief sets in [19] by allowing for arbitrary, thus possibly infinite, non-empty countable supersets of the program's constants, so-called *universes*. *Open answer sets* are then pairs (U, M) with M an answer set of the program P grounded with the universe U. Recapitulating our example, we have that ({john, x}, {pass(john), fail(x)}) is an open answer set of the program. Indeed, the grounding is now w.r.t. {john, x} instead of {john} where x is a

[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*.

new *anonymous* element. The grounding 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.

However, as reasoning with k-belief sets is already undecidable [37] 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 a specific form of programs for which reasoning under the open answer set semantics is decidable, but which is still expressive enough to represent useful knowledge.

Satisfiability checking for such *conceptual logic programs* (CoLPs) is reduced to checking non-emptiness of two-way alternating tree automata (2ATA) [45]. The reduction yields an EXPTIME-upper bound for satisfiability checking w.r.t. CoLPs. CoLPs turn out to be useful for expressing conceptual knowledge, hence their naming, as they can simulate expressive *description logics* (DLs). Description logics [3] 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. The basic language features of such languages 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.

Reasoning in the particular DL $\mathcal{SHIQ}$ can be 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.

A promising area of application for open answer set programming is the envisioned *Semantic Web*. The Semantic Web [7] 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' [40] of certain domains.

Although DLs are being heavily promoted as an ontology language standard (see the ontology language OWL DL [6]), they are by no means a synonym for *ontology language*. Possible alternatives to DL ontologies include, e.g., ORM [21] ontologies as illustrated in the DOGMA framework [27], 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 [31]. 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., [1, 13–15, 20, 22, 25, 26, 28, 31, 35, 38, 41].

[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.

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 features in one unifying decidable framework, allowing for both negation as failure and open domains in a rule-based formalism.

The remainder of the paper is organized as follows. We introduce the basic definitions and properties of open answer set programming in Section 2.1. Section 2.2 shows that open answer set programming is undecidable in general, while Section 2.3 identifies conceptual logic programs as a decidable fragment. In Section 3.1 we recall the DL $\mathcal{SHIQ}$. The simulation of $\mathcal{SHIQ}$ reasoning with CoLPs can be found in Section 3.2. Section 3.3 discusses the advantages of CoLPs over $\mathcal{SHIQ}$ and Section 4 contains an overview of related work. Finally, we conclude and give directions for further research in Section 5.

2 Open answer set programming

2.1 Basic definitions and properties

We define the language of OASP. *Constants*, *variables*, *terms*, and *atoms* are defined as usual. A *literal* is an atom $p(\mathbf{t})$ or a *naf-atom* not $p(\mathbf{t})$.[3] The *positive part* of a set of literals α is $\alpha^+ = \{p(\mathbf{t}) \mid p(\mathbf{t}) \in \alpha\}$ and the *negative part* of α is $\alpha^- = \{p(\mathbf{t}) \mid \text{not } p(\mathbf{t}) \in \alpha\}$. We assume the existence of binary predicates $=$ and $\neq$, where $t = s$ is considered as an atom and $t \neq s$ as not $t = s$. E.g. for $\alpha = \{X \neq Y, Y = Z\}$, we have $\alpha^+ = \{Y = Z\}$ and $\alpha^- = \{X = Y\}$. A *regular* atom is an atom that is not an equality atom. For a set X of atoms, not $X = \{\text{not } l \mid l \in X\}$.

A *program* is a countable set of rules $\alpha \leftarrow \beta$, where α and β are finite sets of literals, and $\forall t, s \cdot t = s \notin \alpha^+$, i.e., the positive part of α does not contain equality atoms. The set α is the *head* of the rule and represents a disjunction of literals, while β is called the *body* and represents a conjunction of literals. If $\alpha = \emptyset$, the rule is called a *constraint*. Atoms, literals, rules, and programs that do not contain variables are *ground*.

For a program P, let cts(P) be the constants in P, vars(P) its variables, preds(P) its predicates, upreds(P) its unary and bpreds(P) its binary predicates. Let $\mathcal{B}_P$ be the set of regular atoms that can be formed from a ground program P.

An *interpretation* I of a grounded program P is any subset of $\mathcal{B}_P$. For a ground regular atom $p(\mathbf{t})$, we write $I \models p(\mathbf{t})$ if $p(\mathbf{t}) \in I$; For an equality atom $p(\mathbf{t}) \equiv t = s$, we have $I \models p(\mathbf{t})$ if s and t are equal terms, while we have $I \models \text{not } p(\mathbf{t})$ if $I \not\models p(\mathbf{t})$. For a set of ground literals X, $I \models X$ if $I \models l$ for every $l \in X$. A ground rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. For a ground program P without not, an interpretation I of P is a *model* of P if I satisfies every rule in P; it is an *answer set* of P if it is a subset minimal model of P. For ground programs P containing not, the

[3] We have no negation $\neg$, however, programs with $\neg$ can be reduced to programs without it, see e.g. [29].

GL-reduct [18] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $I \models \text{not } \beta^-$ and $I \models \alpha^-$. I is an *answer set* of a ground P if I is an answer set of P^I.

Definition 1 A *universe* U for a program P is a non-empty countable superset of the constants in P: $\text{cts}P \subseteq U$. We call P_U the ground program obtained from P by substituting every variable in P by every possible element from U.

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.

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

Definition 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 1 Take a program P with rules $p(X) \leftarrow \text{not } q(X)$ and $q(a) \leftarrow$. Then $\text{cts}(P) = \{a\}$ 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

$$\begin{array}{ll} p(a) \leftarrow \text{not } q(a) & p(x_1) \leftarrow \text{not } q(x_1) \\ p(x_2) \leftarrow \text{not } q(x_2) & \ldots \\ q(a) \leftarrow & \end{array}$$

such that $(\{a, x_1, x_2, \ldots\}, \{q(a), p(x_1), p(x_2), \ldots\})$ 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 procedure we consider for the open answer set semantics is *satisfiability checking*.

Definition 3 For an n-ary predicate p, appearing in a program 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 predicate p in the program P in Example 1 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, while in the open answer set semantics p is satisfiable.

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 2 Take the program

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

Rule r_1 represents the knowledge that a system that has crashed on a particular day X (crash(X)), can be restored on that day (restore(X)) if a backup of the system on the day Y before ($y(X, Y)$, y stands for *yesterday*) succeeded (backSucc(Y)). Backups succeed if the system does not crash and it cannot be established that the backups at previous dates failed (r_2) and a backup fails if it does not succeed (r_3). Rule r_4 ensures that for a particular day there can be only one day after. Rules r_5 and r_6 allow to freely introduce y and crash literals. 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_6 (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 must be infinite. An example of such an open answer set is (we omit U if it is clear from M) $M \equiv \{$restore(x), crash(x), backFail(x), $y(x, x_1)$, backSucc(x_1), $y(x_1, x_2)$, backSucc(x_2), $y(x_2, x_3), \ldots\}$. One can check 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.

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 1 *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 1 to hold.

Example 3 Take the non-ground program with rules $q(a) \leftarrow \text{not } p(X)$ and $p(a) \leftarrow$. Then $(\{a, x\}, \{p(a), q(a)\})$ is an open answer set, while the open interpretation $(\{a\}, \{p(a), q(a)\})$ is not.

In order to be able to define an *immediate consequence operator*, we restrict ourselves in the rest of this paper 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, 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* [42] 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, $T_P^{(U,M)} : \mathcal{B}_P \to \mathcal{B}_P$ 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 $T^{n+1}(B) = T(T^n(B))$.[4] Although we allow for infinite universes, we can motivate the presence of atoms in open answer sets in a finite way, where the motivation of an atom is formally expressed by the immediate consequence operator.

Theorem 2 *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 Sketch Assume, by contradiction, $\exists a_1 \in M \cdot \forall n < \infty \cdot a_1 \notin T^n$. We then know that for every rule with head a_1 that has a true body, there must be a body literal that satisfies the same conditions as a_1, i.e., it cannot be in a finite application of T to the empty set. One can continue this way and define an interpretation that equals M without those identified literals. One can show that this new interpretation is a model of P_U^M, contradicting the minimality of M. □

We restrict ourselves in the remainder of this paper to programs with unary and binary predicates only. This allows us to introduce, similar to some description logics (DLs, see Section 3), *inverted predicates* f^{i} for a binary predicate f.[5] Intuitively, $f(x, y)$ will hold iff its inverse $f^{\mathrm{i}}(y, x)$ holds. For a set X of binary (possibly inverted) predicate names, $X^{\mathrm{i}} \equiv \{f^{\mathrm{i}} \mid f \in X\}$ where $f^{\mathrm{ii}} \equiv f$. We call atoms $f^{\mathrm{i}}(s, t)$, 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^{i} or a binary f in the program, the Herbrand Base contains atoms with predicate f^{i} and f. We further have that bpreds(P) includes both f and f^{i} for a f or f^{i} in P.

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

Definition 4 Let P be a ground program and M an interpretation of P. Then IWA(P, M) is the formula

$$\forall f \in \text{bpreds}(P) \cdot f(x, y) \in M \iff f^{\mathrm{i}}(y, x) \in M\,. \tag{1}$$

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 IWA(P, M) holds. Models, minimal models, and answer sets under IWA of a ground program

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

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

P are then defined as usual but with interpretations under IWA, instead of just interpretations.

Definition 5 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 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 $p(\mathbf{x}) \in M$.

Example 4 Modify the program of Example 2 by adding inverted predicates, i.e., replace $\leftarrow y(Y_1, X), y(Y_2, X), Y_1 \neq Y_2$ by its counterpart $\leftarrow y^{\mathrm{i}}(X, Y_1), y^{\mathrm{i}}(X, Y_2), Y_1 \neq Y_2$ with inverses. An open answer set under IWA is then {restore(x), crash(x), backFail (x), $y(x, x_1)$, $y^{\mathrm{i}}(x_1, x)$, backSucc (x_1), $y(x_1, x_2)$, $y^{\mathrm{i}}(x_2, x_1)$, backSucc (x_2), y (x_2, x_3), $y^{\mathrm{i}}(x_3, x_2), \ldots$}.

Satisfiability under IWA does not imply (normal) satisfiability.

Example 5 Take the program containing the rules $q(X) \leftarrow f(X, Y)$ and $f^{\mathrm{i}}(X, Y) \vee$ not $f^{\mathrm{i}}(X, Y) \leftarrow$. 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, and thus q is not satisfiable.

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

Example 6 Take the program P:

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

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 $p(X) \leftarrow$ 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 *Let P be a program and p a unary 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^{i} and f). □

For programs that do not contain inverted predicates satisfiability is equivalent to satisfiability under IWA. Theorem 3 shows that the IWA does not add extra expressiveness, however, it allows for an elegant definition of conceptual logic programs, see below. Note that for a program without inverted predicates and p a n-ary predicate, $1 \leq n \leq 2$, we have that p is satisfiable w.r.t. P iff p is satisfiable under IWA w.r.t. P.

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, ${T^{\mathrm{i}}}_P^{(U,M)} : \mathcal{B}_P \to \mathcal{B}_P$ 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 ${T^{\mathrm{i}}}^0(B) = B$,[6] 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 4 *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 The proof is similar to the proof of Theorem 2. □

2.2 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.

Intuitively, the domino (or tiling) problem asks whether, given a set of dominoes D and a $d \in D$, there is a tiling of the plane $\mathbb{N} \times \mathbb{N}$ that contains d. 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).

A domino *is present* in a tiling τ if there is some $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that $\tau(x, y) = d$; the domino problem is then *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?* The domino problem is undecidable [9].

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

We can reduce the domino problem to satisfiability checking under the open answer set semantics. 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 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 $\mathbb{N} \times \mathbb{N}$ 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 one domino type is assigned to each position. Finally, f_3^i introduces the dominoes itself.

Theorem 5 *Let $\mathcal{D}$ be a domino system and d_i some domino in $\mathcal{D}$. Then, $\mathcal{D}$ tiles the plane $\mathbb{N} \times \mathbb{N}$ such that the domino d_i is present in the tiling iff the corresponding predicate d_i is satisfiable w.r.t.* $[\mathcal{D}]$.

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

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

We have that d_i is satisfied in M: d_i is present in τ, such that there is a $(x, y) \in \mathbb{N} \times \mathbb{N}$ with $\tau(x, y) = d_i$. By definition of M, we have that $d_i(x, y) \in M$. One can show that (U, M) is an open answer set of $[\mathcal{D}]$.

Table 1 Domino program.

$\mathbb{N} \times \mathbb{N}$	$h: \leftarrow h(U, V_1), h(U, V_2), V_1 \neq V_2$	
	$v: \leftarrow v(U, V_1), v(U, V_2), V_1 \neq V_2$	
	$s: \leftarrow h(U, X), v(X, V_1), v(U, Y), h(Y, V_2), V_1 \neq V_2$	
	$\mathrm{hh}: \mathrm{hh}(U) \leftarrow h(U, X)$	
	$\mathrm{hv}: \mathrm{hv}(U) \leftarrow v(U, X)$	
	$\mathrm{hhc}: \leftarrow \mathrm{not}\ \mathrm{hh}(U)$	
	$\mathrm{hvc}: \leftarrow \mathrm{not}\ \mathrm{hv}(U)$	
	$f_1: h(U, V) \vee \mathrm{not}\ h(U, V) \leftarrow$	
	$f_2: v(U, V) \vee \mathrm{not}\ v(U, V) \leftarrow$	
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 $(d_i, d_j) \notin V$
	$d_3: \leftarrow \mathrm{not}\ d_1(X), \ldots, \mathrm{not}\ d_k(X)$	
	$d_4^{i,j}: \leftarrow d_i(X), d_j(X)$	for $i \neq j$
	$f_3^i: d_i(U) \vee \mathrm{not}\ d_i(U) \leftarrow$	for $1 \leq i \leq k$

For the 'if' direction, assume that (U, M) is an open answer set of $[\mathcal{D}]$ containing a $d_i(u_0)$ for $u_0 \in U$. For each $(x, y) \in \mathbb{N} \times \mathbb{N}$, define τ 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)). One can show that τ is well-defined and that the domino conditions are satisfied.

First, we show that τ is well-defined:

- Every element in $\mathbb{N} \times \mathbb{N}$ has an image through τ. 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 τ.
- 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 τ, 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 $h(t_y, t'_y) \in M$.

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

Corollary 1 Satisfiability checking is undecidable.

Proof This is an immediate consequence of the undecidability of the domino problem and Theorem 5. □

Corollary 2 Satisfiability checking under IWA is undecidable.

Proof The domino program in Table 1 contains only unary and binary predicates and no inverted predicates. The result follows from the undecidability of satisfiability checking that was established in Corollary 1. □

2.3 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, then it can be satisfied by an open answer set with a tree structure. I.e., CoLPs have the *tree model property*. In [44], 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 two-way alternating tree automaton (2ATA).

2.3.1 *Conceptual logic programs*

We first give some preliminary definitions of (infinite) trees as in [45]. A *(finite) tree* T is a (finite) subset of $\mathbb{N}_0^{*}$[7] such that if $xc \in T$ for $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, we have that $x \in T$. A tree T is *complete* if for $xc \in T$ also $xc' \in T$ for $0 < c' < c$.

Elements of T are called nodes and the empty word ε is the root of T. For a node $x \in T$ we call xc, $c \in \mathbb{N}_0$, *successors* of x. By convention, $x0 = x$ and $(xc) - 1 = x$ ($\varepsilon - 1$ is undefined). If every node x in a tree has k successors we say that the tree is *k-ary*. An *infinite path* P of T is a prefix-closed subset of T such that for every $0 \leq i$ there is a unique $x \in P$ such that the length of x is i ($|x| = i$). A *labeled tree* over an alphabet Σ is a function $t : T \to \Sigma$ where T is a tree, labeling the nodes of T with elements from the alphabet.

[7] $\mathbb{N}_0^*$ is the set of finite sequences that can be formed using the natural numbers, excluding 0.

Recall the program in Example 2, which has an open answer set (U, M) with $U \equiv \{x, x_1, \ldots\}$ and $M \equiv \{\text{restore}(x), \text{crash}(x), \text{backFail}(x), y(x, x_1), \text{backSucc}(x_1), y(x_1, x_2), \text{backSucc}(x_2), y(x_2, x_3), \ldots\}$.

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

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

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 $t : U' \to 2^{\text{preds}(P)}$: 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 $y = xi \wedge f \in t(y)$.

Intuitively, unary literals $a(x)$ can be encoded in the label of node x and binary literals $f(x, xi)$ can be encoded in the label of xi, the ith successor of x. A particular f in the label of a node xi indicates that $f(x, xi) \in M$ since each node xi has the unique predecessor x. The open answer set (U', M') can be encoded as the linear tree in figure 1.

If we consider open answer sets under the IWA, we can also encode literals $f(xi, x)$, 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 $f(xi, x)$ also contain $f^{\mathrm{i}}(x, xi)$. Similarly to the above, we place f^{i} in the label of xi. Since xi has only one predecessor, x, such a label uniquely identifies $f^{\mathrm{i}}(x, xi)$ and thus also $f(xi, x)$.

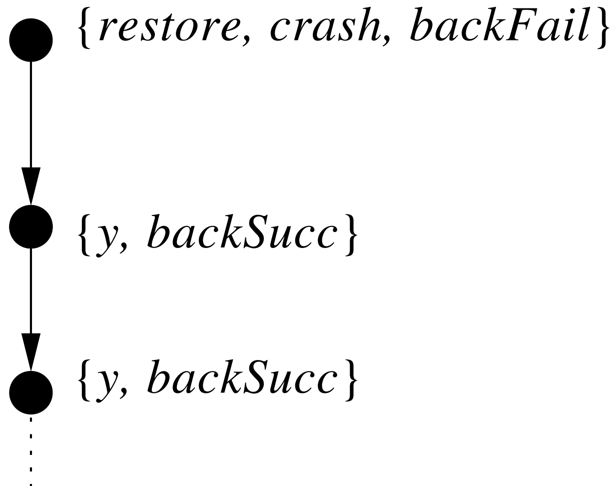

Figure 1 Backup example tree.

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

Example 7 Modify the program in Example 4 by adding the rule

$$\text{tomor}^{\mathrm{i}}(X, Y) \leftarrow y(X, Y)\ .$$

The modified program then has an open answer set under IWA (U, M) that can be encoded as the labeled tree t in figure 2.

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 $y = xi \wedge f \in t(y)$ or $x = yi \wedge f^{\mathrm{i}} \in t(x)$.

Further note that the encoded trees in both of the above examples are minimal, in the sense that for every node zi in the tree-shaped universe there is some $f(z, zi)$ in the open answer set under the IWA where f is possibly inverted. Intuitively, the tree cannot contain *dangling* nodes. A unary predicate p is *tree satisfiable under IWA* if there is an open answer set (U, M) under the IWA 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 6 Let P be a program. A $p \in \text{upreds}(P)$ is *tree satisfiable under IWA* w.r.t. P if there exists

- An open answer set (U, M) under IWA of P such that U is a tree of bounded arity, and
- A labeling function $t : U \rightarrow 2^{\text{preds}(P)}$ such that

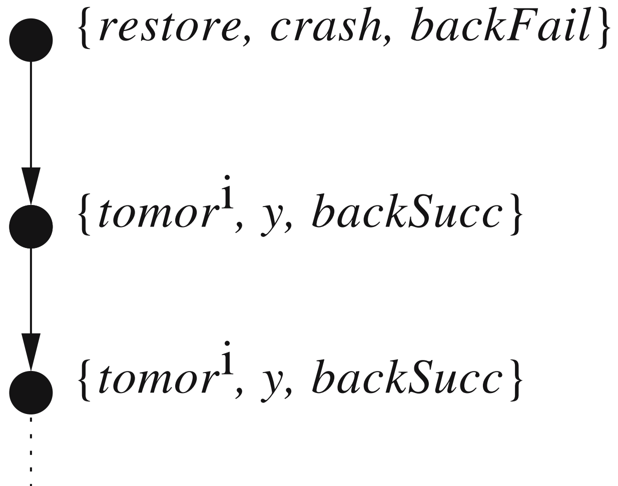

Figure 2 Modified backup example tree.

- $p \in t(\varepsilon)$ and $t(\varepsilon)$ does not contain (possibly inverted) binary predicates, and
- $zi \in U, i > 0$, iff there is some $f(z, zi) \in M$ where f is possibly inverted, and
- for $y \in U, q \in \text{upreds}(P)$, $f \in \text{bpreds}(P)$,
  - $q(y) \in M$ iff $q \in t(y)$, and
  - $f(x, y) \in M$ iff $y = xi \wedge f \in t(y)$ or $x = yi \wedge f^{\mathrm{i}} \in t(x)$, 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 $p \in \text{upreds}(P)$ is satisfiable under IWA w.r.t. P then p is tree satisfiable under IWA w.r.t. P.

We will often denote a set like, e.g., $\{a(X), \text{not } b(X)\}$ as $\alpha(X)$ with $\alpha = \{a, \text{not } b\}$; similarly for sets of binary (possibly inverted) literals, e.g., $\{f(X, Y), \text{not } g^{\mathrm{i}}(X, Y)\}$ will be denoted as $\alpha(X, Y)$ for $\alpha = \{f, \text{not } g^{\mathrm{i}}\}$. If we only write $\alpha(X)$, without specifying α, it is assumed that α 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 7 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* $a(X) \vee \text{not } a(X) \leftarrow$ or $f(X, Y) \vee \text{not } f(X, Y) \leftarrow$, where f is possibly inverted (similarly for the subsequent rule types),
- *Unary rules*

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

  with k a finite natural number and 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. $\forall Y_i \in \text{vars}(r) \cdot \gamma_i^+ \neq \emptyset$, 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 $\leftarrow f(X, Y)$.

Intuitively, unary rules 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.,

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

deduces a at X if X has two different neighbors Y_1 and Y_2. E.g.,

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

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. The restriction $\forall Y_i \in \text{vars}(r) \cdot \gamma_i^+ \neq \emptyset$ is necessary to have the tree model property: 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[8]

$$\begin{aligned} q(X) &\leftarrow a(X), \text{not } p(X) \\ p(X) &\leftarrow \text{not } r(X, Y), a(Y) \end{aligned}$$

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.

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 [2].

A unary rule 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 $\text{degree}(r) \equiv |\{m \mid \gamma_m \neq \emptyset\}|$. The *degree of a live predicate a* in P is $\text{degree}(a) \equiv \max\{\text{degree}(r) \mid a \in \text{head}(r)\}$. The *rank of a* CoLP P is the sum of the degrees of the live predicates in P: $\sum_{a \in \text{live}(P)} \text{degree}(a)$. Intuitively, given a node in an encoded tree with a certain label that contains some unary predicates, every live unary predicate in label of 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 label. 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.

Theorem 6 *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, θ associates some of the nodes in the complete tree with elements in the domain.

[8]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., [30].

Initially, assume that θ is undefined for the whole tree $\{1,\ldots,n\}^*$. If θ is defined on some node x, we will call the node x *defined*. θ is then constructed by first defining $\theta(\varepsilon)=u$. Subsequently, assume we have considered, as in [45], every node in $\{1,\ldots,n\}^k$, for some k, as well as every successor node of the defined $z'\in\{1,\ldots,n\}^k$ with $|z'|=k$ until[9] zm for some defined $z\in\{1,\ldots,n\}^k$ with $|z|=k$. Consequently, we have considered the nodes $z1,\ldots,zm$.

Since θ is defined on z, we have that $\theta(z)\in U$. For every $q(\theta(z))\in M$, there is, by Theorem 2, some $l<\infty$ s.t. $q(\theta(z))\in T^{\mathbf{i}^l}$. By definition of the immediate consequence operator, we have that there is a rule $r_{q(\theta(z))}: q(\theta(z))\leftarrow\beta^+[]\in P_U^M$ with $M\models\beta^+[]$, originating from $r: q(X)\vee\alpha\leftarrow\beta\in P$ such that $M\models\alpha^-[]$, $M\models$ not $\beta^-[]$, and $T^{\mathbf{i}^{l-1}}\models\beta^+[]$. 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 $zj\in\{z-1,z1,\ldots,zm,\ldots,z(m+i-1))\}$ with $\theta(zj)=y_i$ then θ remains undefined on $z(m+i)$, otherwise $\theta(z(m+i))=y_i$. Intuitively, if θ is already defined on a neighbor of z as equal to y_i, there is no need to define θ on another successor as equal to y_i.

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

- $t(\varepsilon)\equiv\{q\mid q(u)\in M\}$,
- $t(zi)\equiv\{q\mid q(\theta(zi))\in M\}\cup\{f\mid f(\theta(z),\theta(zi))\in M,\ f \text{ poss. inv.}\}$.

Define the open interpretation (V,N) such that $V\equiv\mathrm{dom}(\theta)$ and $N\equiv\{q(z)\mid q\in t(z)\}\cup\{f(z,zi),\ f^{\mathbf{i}}(zi,z)\mid f\in t(zi),\ f \text{ poss. inverted}\}$. It is then straightforward to check that (V,N) is a tree model under IWA satisfying p according to Definition 6.

□

2.3.2 *Decidability of conceptual logic programs*

Two-way alternating tree automata (2ATA) [45] 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 it 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 δ 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

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

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.

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 ϕ if assigning **true** to the elements in J and **false** to the elements in $I \setminus J$ makes ϕ true according to the standard inductive semantics for boolean formulas. A *two-way alternating tree automaton* (2ATA) [45] over k-ary infinite trees is a tuple $(\Sigma, Q, q_0, \delta, \Omega)$ where Σ 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 Ω is the acceptance condition.

A *run* over a tree $t : T \to \Sigma$ is a tree[10] $r : R \to T \times Q$ 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 ϕ, 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 Ω. 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 Ω if for every path π in r, there exists an even i such that $\text{In}(\pi) \cap G_i \neq \emptyset$ and $\text{In}(\pi) \cap G_{i-1} = \emptyset$, where $\text{In}(\pi)$ are the states that appear infinitely often in the labels of nodes in π. A tree is accepted by a 2ATA A if there is an accepting run. We denote the set of trees that are accepted by A as $\mathcal{L}(A)$, i.e., the language of A. Non-emptiness checking of a 2ATA amounts then to checking whether $\mathcal{L}(A) \neq \emptyset$.

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. Note that the tree model property ensures that open answer sets can be written as trees and subsequently fed as input to a tree 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.

Definition 8 An infinite k-ary tree $t : T \to 2^{\text{preds}(P)} \cup \{\{\text{dummy}\}\}$ for a program P with rank k is well-behaved if the root label does not contain binary predicates (possibly inverted) from P, and, if the label of a node is $\{\text{dummy}\}$, then the labels of all its successors are $\{\text{dummy}\}$.

[10] Note that the alphabet of r is infinite.

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 Σ. The alphabet of the automaton is $2^{\mathrm{preds}(P)} \cup \{\{\mathrm{dummy}\}\}$, 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 δ. 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

$$\delta(q_0, n) = p \in n \wedge (0, q_1) \tag{2}$$

for any $n \in 2^{\mathrm{preds}(P)} \cup \{\{\mathrm{dummy}\}\}$. 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

$$\delta(q_1, n) = \left(\bigwedge_{a \in n} (0, q_a) \wedge \bigwedge_{a \notin n} (0, \overline{q_a}) \wedge \bigwedge_{c \text{ constraint}} (0, q_c) \wedge \bigwedge_{1 \le i \le k} (i, q_1) \right) \vee (n = \{\mathrm{dummy}\}) \,, \tag{3}$$

where $a \in \mathrm{preds}(P)$. 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* : $\mathrm{preds}(P) \to \{\mathbf{true}, \mathbf{false}\}$ such that *free(q)* returns true if q (or its inverse) is free. For unary predicates $a \in \mathrm{preds}(P)$ and binary (possibly inverted) predicates $f \in \mathrm{preds}(P)$, we have the transitions:

$$\delta(q_a, n) = a \in n \wedge \left(\mathit{free}(a) \vee \bigvee_{r:(a)(X) \leftarrow \beta} (0, q_r) \right) \tag{4}$$

and

$$\delta(q_f, n) = f \in n \wedge \left(\mathit{free}(f) \vee \bigvee_{r: f(X,Y) \leftarrow \beta} (0, q_r) \vee \bigvee_{r: f^{\mathrm{i}}(X,Y) \leftarrow \beta} (0, q_{r^{\mathrm{i}}}) \right) . \tag{5}$$

The transitions q_a and q_f need to motivate the presence of a and f in the label. They check that a and f 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^{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 $I = \{i_{Y_i} \mid Y_i \in \text{body}(r), i_{Y_i} \in \{0, \ldots, k\}\}$ satisfies ψ 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 ψ 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

$$\delta(q_r, n) = (0, q_\beta) \wedge \exists I \text{ satisfies } \psi \cdot \left(\bigwedge_{m_{Y_m} \in I} (m_{Y_m}, q'_{\gamma_m}) \wedge (m'_{Y_m}, q_{\delta_m}) \right) , \tag{6}$$

with

$$q'_{\gamma_m} = \begin{cases} q_{\gamma_m^{\mathrm{i}}} & \text{if } m_{Y_m} = 0 \\ q_{\gamma_m} & \text{else} \end{cases} \quad \text{and} \quad m'_{Y_m} = \begin{cases} -1 & \text{if } m_{Y_m} = 0 \\ m_{Y_m} & \text{else} \end{cases} .$$

Intuitively, when reading a label with a at node X, one has to verify that β 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 ψ and check γ_m and δ_m. If a direction m_{Y_m} is such that $0 < m_{Y_m}$, i.e., down the tree, then one has to check γ_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 $X2$ of X and checks whether f is in the label of $X2$ (recall that a f in a label of zi indicates a connection $f(z, zi)$). If $m_{Y_m} = 0$, we assume the Y_m is the predecessor of X and we check that γ_m^{i} holds at X itself and we go one node up (direction -1) to check δ_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^{i} is in the label of X and b is in the label of the predecessor Y_m of X (recall that a f^{i} in a label of z indicates a connection $f^{\mathrm{i}}(z-1, z)$ or $f(z, z-1)$).

- The transition for a binary $r : f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$ includes

$$\delta(q_r, n) = (-1, q_\beta) \wedge (0, q_\gamma) \wedge (0, q_\delta) \tag{7}$$

and

$$\delta(q_{r^{\mathrm{i}}}, n) = (-1, q_\delta) \wedge (0, q_{\gamma^{\mathrm{i}}}) \wedge (0, q_\beta) . \tag{8}$$

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

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

$$\delta(q_\gamma, n) = \bigwedge_{a \in \gamma} (0, q_a) \wedge \bigwedge_{\text{not } a \in \gamma} a \notin n , \tag{9}$$

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 $a \in \text{preds}(P)$ and binary (possibly inverted) predicates $f \in \text{preds}(P)$, we have the transitions:

$$\delta(\overline{q_a}, n) = a \notin n \wedge \left(\neg \text{free}\,(a) \wedge \bigwedge_{r: a(X) \leftarrow \beta} (0, \overline{q_r}) \right) \tag{10}$$

  and

$$\delta(\overline{q_f}, n) = f \notin n \wedge \left(\neg \text{free}\,(f) \wedge \bigwedge_{r: f(X,Y) \leftarrow \beta} (0, \overline{q_r}) \wedge \bigwedge_{r: f^{\text{i}}(X,Y) \leftarrow \beta} (0, \overline{q_{r^{\text{i}}}}) \right) . \tag{11}$$

– For a unary rule $r : a(X) \leftarrow \beta(X), \gamma_m(X, Y_m), \delta_m(Y_m), \psi$ we have the transition

$$\delta(\overline{q_r}, n) = (0, \overline{q_\beta}) \vee \forall I \text{ satisfies } \psi \cdot \left(\bigvee_{m_{Y_m} \in I} (m_{Y_m}, \overline{q_{\gamma_m}}') \vee (m'_{Y_m}, \overline{q_{\delta_m}}) \right) \tag{12}$$

  with

$$\overline{q_{\gamma_m}}' = \begin{cases} \overline{q_{\gamma_m^{\text{i}}}} & \text{if } m_{Y_m} = 0 \\ \overline{q_{\gamma_m}} & \text{else} \end{cases} \quad \text{and} \quad m'_{Y_m} = \begin{cases} -1 & \text{if } m_{Y_m} = 0 \\ m_{Y_m} & \text{else} \end{cases} .$$

– The transition for a binary $r : f(X, Y) \leftarrow \beta(X), \gamma(X, Y), \delta(Y)$ comprises

$$\delta(\overline{q_r}, n) = (-1, \overline{q_\beta}) \vee (0, \overline{q_\gamma}) \vee (0, \overline{q_\delta}) \tag{13}$$

  and

$$\delta(\overline{q_{r^{\text{i}}}}, n) = (-1, \overline{q_\delta}) \vee (0, \overline{q_{\gamma^{\text{i}}}}) \vee (0, \overline{q_\beta}) . \tag{14}$$

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

$$\delta(\overline{q_\gamma}, n) = \bigvee_{a \in \gamma} (0, \overline{q_a}) \vee \bigvee_{\text{not } a \in \gamma} a \in n , \tag{15}$$

  where a is unary or (possibly inverted) binary.

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

$$\delta(q_c, n) = a \notin n . \tag{16}$$

  A constraint c is 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

$$\delta(q_{c_1}, n) = \delta(q_{c_2}, n) = f \notin n \wedge f^{\mathrm{i}} \notin n \,. \tag{17}$$

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

Note that we do not need qualifiers in the transition function Definitions (6) and (12); 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 Ω. Take $\Omega = (Q^+, Q)$. Then, an infinite path π 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 reduces 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 7 *Let P be a CoLP and $p \in \mathrm{upreds}(P)$. p is satisfiable under IWA w.r.t. P iff $\mathcal{L}(A_{p,P}) \neq \emptyset$.*

Proof For the 'only if' direction, assume that p is satisfiable under IWA w.r.t. P, then, by Theorem 6, p is tree satisfiable under IWA w.r.t. P. By Definition 6, there exists a tree model 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 corresponding labeling function $t : U \to 2^{\mathrm{preds}(P)}$.

The tree U may be finite, however, a 2ATA demands for an infinite tree input. We take the infinite complete k-ary tree U' and define $t' : U' \to 2^{\mathrm{preds}(P)} \cup \{\{\mathrm{dummy}\}\}$ such that for $x \in U$, $t'(x) \equiv t(x)$, and for $x \in U' \backslash U$, $t'(x) \equiv \{\mathrm{dummy}\}$. 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. One can then check that t' is accepted by $A_{p,P}$ such that $\mathcal{L}(A_{p,P}) \neq \emptyset$.

For the 'if' direction, assume $t : T \to 2^{\mathrm{preds}(P)} \cup \{\{\mathrm{dummy}\}\}$ is an infinite labeled k-ary tree that is accepted by $A_{p,P}$. Denote the corresponding run with r. Define (U, M) with $U \equiv \{x | x \in T, t(x) \neq \{\mathrm{dummy}\}\}$ and $M \equiv \{q(x) \mid q \in t(x) \cap \mathrm{upreds}(P)\} \cup \{f(x, xi), f^{\mathrm{i}}(xi, x) \mid f \in t(xi) \cap \mathrm{bpreds}(P)\}$. 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 (2) 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 $p(\varepsilon) \in M$. One can show that (U, M) is an open answer set under IWA of P. □

The reduction yields a complexity upper bound for satisfiability checking under IWA w.r.t. CoLPs.

Theorem 8 *Satisfiability checking under IWA w.r.t. CoLPs is decidable and in* EXPTIME.

Proof With Theorem 7, 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 $A_{p,P}$ [45]. One can see that the number of states of $A_{p,P}$ is polynomial in the size of P such that the result follows. □

Satisfiability checking w.r.t. CoLPs is more efficient than normal (finite) answer set programming for arbitrary programs, which is NEXPTIME-complete if the head contains at most one positive literal, see [12]. A final note regarding the formal properties of CoLPs is that the syntax of CoLPs can be loosened up without loss of generality. One can unfold the bodies of unary and binary rules yielding, instead of tree-shaped rules of one level deep, tree-shaped rules of arbitrary finite depth. We can also allow for constraints $\leftarrow \beta$, where β is a body as in a unary or binary CoLP rule. Such general constraints can be easily rewritten as the CoLP rules $a(X) \leftarrow \beta$ and $\leftarrow a(X)$ in the unary case, or as $f(X, Y) \leftarrow \beta$ and $\leftarrow f(X, Y)$ in the binary case.

3 Simulating the description logic $\mathcal{SHIQ}$

3.1 The DL $\mathcal{SHIQ}$

Description logics (DLs) constitute a family of logical formalisms useful for knowledge representation, e.g., the representation of taxonomies in certain application domains [33].

The 'Semantic Web' [7] 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' [40] of certain domains. In order to describe ontologies, one can use ontology languages, such as DAML+OIL, OIL [5, 16, 17], or, more recently, OWL [6]. 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 or whether classes are subclasses of other classes [4, 24].

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.

We define the syntax and semantics of $\mathcal{SHIQ}$ [23] expressions in Table 2, where $\mathcal{SHIQ}$ is the formal DL underlying OIL. In the table, R is a role, R^{i} its inverse, and A is a concept name which is the basic concept expression; C and D are concept expressions that can be used to build more complex concept expressions such as conjunction, disjunction, exists restriction, value restriction, at least restriction, and at most restriction. The latter two expressions will be referred to as qualified number restrictions.

A DL *knowledge base* is a set of *axioms*, where an axiom is either a *terminological axiom* $C \sqsubseteq D$ with C and D concept expressions, a *role axiom* $R \sqsubseteq S$ where R, S may be inverse roles, or a *transitivity axiom* $\mathsf{Trans}(R)$ for an (inverse) role R. We often

Table 2 Syntax and semantics of $\mathcal{SHIQ}$ constructs.

Construct	Syntax	Semantics
Concept name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role name	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Inverse role	R^-	$(R^-)^{\mathcal{I}} = \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$
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 restr.	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
Value restr.	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
At least restr.	$\geq nS.C$	$(\geq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$
At most restr.	$\leq nS.C$	$(\leq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x, y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$

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 reflexive-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, for $R \sqsubseteq S$ a role axiom with (possibly inverted) 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.

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 Σ if it satisfies every axiom in Σ. A concept C is *satisfiable* w.r.t. Σ if there is a model $\mathcal{I}$ of Σ 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., [24]).

Example 8 Consider the knowledge base Σ with axioms

$$\text{Personnel} \equiv \text{Management} \sqcup \text{Workers} \sqcup \exists \text{boss.Management}$$
$$\text{Management} \sqsubseteq (\forall \text{take_orders.Management}) \sqcap (\geq 3\ \text{boss.Workers})$$

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 three workers. Additionally, we assume Σ contains the axiom $\mathsf{Trans}(boss)$, 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 $\mathcal{I} = (\{j, w_1, w_2, w_3, m\}, ^{\mathcal{I}})$, with $^{\mathcal{I}}$ defined by $\text{Workers}^{\mathcal{I}} = \{w_1, w_2, w_3\}$, $\text{Management}^{\mathcal{I}} = \{m\}$, $\text{Personnel}^{\mathcal{I}} = \{j, w_1, w_2, w_3, m\}$, $\text{boss}^{\mathcal{I}} = \{(j, m), (m, w_1), (m, w_2), (m, w_3), (j, w_1),$
$(j, w_2), (j, w_3)\}$, and $\text{take_orders}^{\mathcal{I}} = \emptyset$.

Satisfiability checking of $\mathcal{SHIQ}$ concept expressions w.r.t. $\mathcal{SHIQ}$ knowledge bases is EXPTIME-complete [39].

3.2 Simulating $\mathcal{SHIQ}$ with conceptual logic programs

Consider the knowledge base Σ from Example 8. We translate the two axioms as three CoLP constraints (the first axiom actually corresponds to two terminological axioms):[11]

$$\leftarrow \text{Per}(X), \text{not } (\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(X)$$
$$\leftarrow \text{not Per}(X), (\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(X)$$
$$\leftarrow \text{Man}(X), \text{not } ((\forall \text{tak.Man}) \sqcap (\geq 3\, \text{boss.Wor}))(X)$$

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 $\text{Man}(x)$ holds, but $((\forall \text{tak.Man}) \sqcap (\geq 3\, \text{boss.Wor}))(x)$ does not, we have a contradiction. This corresponds to the DL behavior of the corresponding axiom: if $x \in \text{Management}^{\mathcal{I}}$ and $x \notin ((\forall \text{tak.Man}) \sqcap (\geq 3\text{ boss.Wor}))^{\mathcal{I}}$, we have a contradiction as the axiom requires that $\text{Man}^{\mathcal{I}} \subseteq ((\forall \text{tak.Man}) \sqcap (\geq 3\text{ boss.Wor}))^{\mathcal{I}}$ for models $\mathcal{I}$.

Note that we do not encode the transitivity of boss directly as a constraint $\leftarrow \text{boss}(X, Y), \text{boss}(Y, Z), \text{not boss } (X,Z)$, 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. We define *Per* with the free $\text{Per}(X) \vee \text{not Per}(X) \leftarrow$. 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 $\text{Man}(X) \vee \text{not Man}(X) \leftarrow$, $\text{Wor}(X) \vee \text{not Wor}(X) \leftarrow$, and $\text{boss}(X, Y) \vee \text{not boss}(X, Y) \leftarrow$. Note that *boss* is a role name, so we introduce it as a binary predicate. The predicate $(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})$ can be defined by the rules:

$$(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(X) \leftarrow \text{Man}(X)$$
$$(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(X) \leftarrow \text{Wor}(X)$$
$$(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(X) \leftarrow (\exists \text{boss.Man})(X)$$

Intuitively, if $(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(x)$ is in an open answer set, then, by minimality of open answer sets, there has to be either a $\text{Man}(x)$, $\text{Wor}(x)$, or a $(\exists \text{boss.Man})(x)$. Vice versa, if $\text{Man}(x)$, $\text{Wor}(x)$, or $(\exists \text{boss.Man})(x)$ holds, then $(\text{Man} \sqcup \text{Wor} \sqcup \exists \text{boss.Man})(x)$ holds as well since the rules must be satisfied. This corresponds exactly to the DL semantics for concept disjunction.

The predicate $(\exists \text{boss.Man})$ is defined by the rules

$$(\exists \text{boss.Man})(X) \leftarrow \text{boss}(X, Y), \text{Man}(Y)$$
$$(\exists \text{boss.Man})(X) \leftarrow \text{boss}(X, Y), (\exists \text{boss.Man})(Y)$$

[11] 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 only appear as arguments of predicates.

The rules explicitly say that $(\exists \text{boss.Man})(x)$ holds in an open answer set iff there is some chain $\text{boss}(x, u_0), \ldots, \text{boss}(u_n, y)$, and $\text{Man}(y)$ that hold in that open answer set. By transitivity of boss, we should indeed have then that $(x, y) \in \text{boss}^{\mathcal{I}}$ such that $x \in (\exists \text{boss.Man})^{\mathcal{I}}$.

The second axiom does not yield any new rules. The last axiom introduces a new free rule $\text{tak}(X, Y) \vee \text{not tak}(X, Y) \leftarrow$ and a rule that defines concept conjunction as conjunction in the body of a rule:

$$((\forall \text{tak.Man}) \sqcap (\geq 3\ \text{boss.Wor}))(X) \leftarrow (\forall \text{tak. Man})(X), (\geq 3\ \text{boss.Wor})(X)$$

The predicate $(\forall \text{tak. Man})$ is defined corresponding to the DL equivalence $\forall \text{tak. Man} \equiv \neg \exists \text{tak.} \neg \text{Man}$:

$$\begin{aligned}(\forall \text{tak. Man})(X) &\leftarrow \text{not } (\exists \text{tak.} \neg \text{Man})(X)\\ (\exists \text{tak.} \neg \text{Man})(X) &\leftarrow \text{tak}(X, Y), (\neg \text{Man})(Y)\\ (\neg \text{Man})(X) &\leftarrow \text{not Man}(X)\end{aligned}$$

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 $\exists \text{tak.} \neg \text{Man}$ like for $\exists \text{boss.Man}$: $\exists \text{tak.} \neg \text{Man}$ should hold only when there is a direct tak-connection with a *Man* element.

Finally, the number restriction is defined as follows:

$$\begin{aligned}(\geq 3\ \text{boss.Wor})(X) \leftarrow\ & \text{boss}(X, Y_1), \text{boss}(X, Y_2), \text{boss}(X, Y_3),\\ & \text{Wor}(Y_1), \text{Wor}(Y_2), \text{Wor}(Y_3),\\ & Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3\end{aligned}$$

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 $(\geq 3\ \text{boss.Wor})(x)$ is in the open answer set.

Before giving the formal translation, define the *closure* $\text{clos}(C, \Sigma)$ of a $\mathcal{SHIQ}$ concept expression C and a $\mathcal{SHIQ}$ knowledge base Σ.

Definition 9 The closure $\text{clos}(C, \Sigma)$ of a $\mathcal{SHIQ}$ concept expression C and a $\mathcal{SHIQ}$ knowledge base Σ is the smallest set satisfying the following conditions:

- $C \in \text{clos}(C, \Sigma)$,
- for each $C \sqsubseteq D$ an axiom in Σ (role or terminological), $\{C, D\} \subseteq \text{clos}(C, \Sigma)$,
- for each $\mathsf{Trans}(R)$ in Σ, $\{R\} \subseteq \text{clos}(C, \Sigma)$,
- for every D in $\text{clos}(C, \Sigma)$, we have
  - if $D = \neg D_1$, then $\{D_1\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = \exists R.D_1$, then $\{R, D_1\} \cup \{\exists S.D_1 \mid S \underline{\sqsubseteq} R, S \neq R, \mathsf{Trans}(S) \in \Sigma\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = \forall R.D_1$, then $\{\exists R. \neg D_1\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = (\leq n\ Q.D_1)$, then $\{(\geq n + 1\ Q.D_1)\} \subseteq \text{clos}(C, \Sigma)$,
  - if $D = (\geq n\ Q.D_1)$, then $\{Q, D_1\} \subseteq \text{clos}(C, \Sigma)$.

Note that for a $R^- \in \text{clos}(C, \Sigma)$, 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^{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 $S \underline{\sqsubseteq}^* R$ 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 Σ and the concept expression C:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add $\leftarrow C(X), \text{not } D(X)$.
- For each role axiom $R \sqsubseteq S \in \Sigma$, add $\leftarrow r(X, Y), \text{not } s(X, Y)$ where r is Q^{i} for $R = Q^-$ or Q for $R = Q$, Q a role name. Similarly for s, i.e., replace $(\cdot)^-$ by $(\cdot)^{\mathrm{i}}$.
- Next, we distinguish between the types of concept expressions that appear in $\mathrm{clos}(C, \Sigma)$. For each $D \in \mathrm{clos}(C, \Sigma)$:
  - If D is a concept name, add $D(X) \vee \text{not } D(X) \leftarrow$,
  - If D is a role name, add $D(X, Y) \vee \text{not } D(X, Y) \leftarrow$,
  - If D is an inverted role name R^- for a role name R, add the free rule $R^{\mathrm{i}}(X, Y) \vee \text{not } R^{\mathrm{i}}(X, Y) \leftarrow$,
  - If $D = \neg E$, add $D(X) \leftarrow \text{not } E(X)$,
  - If $D = E \sqcap F$, add $D(X) \leftarrow E(X), F(X)$,
  - If $D = E \sqcup F$, add $D(X) \leftarrow E(X)$ and $D(X) \leftarrow F(X)$,
  - If $D = \exists Q.E$, add $D(X) \leftarrow q(X, Y), E(Y)$ where q is defined from Q similarly to the above definition for role axioms, and for all $S \underline{\sqsubseteq}^* Q$, $S \neq R$, with $\mathsf{Trans}(S) \in \Sigma$, add rules $D(X) \leftarrow (\exists S.E)(X)$. If $\mathsf{Trans}(Q) \in \Sigma$, we further add the rule $D(X) \leftarrow q(x, y), D(Y)$,
  - If $D = \forall R.E$, add $D(X) \leftarrow \text{not } (\exists R.\neg E)(X)$,
  - If $D = (\leq n\ Q.E)$, add $D(X) \leftarrow \text{not } (\geq n+1\ Q.E)(X)$,
  - If $D = (\geq n\ Q.E)$, add $D(X) \leftarrow q(X, Y_1), \ldots, q(X, Y_n)$, $E(Y_1), \ldots, E(Y_n), \cup_{i \neq j}\{Y_i \neq Y_j\}$, where q is as above.

Rule $D(X) \leftarrow q(X, Y), E(Y)$ 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 the rule $D(X) \leftarrow q(x, y), D(Y)$ 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 rules $D(X) \leftarrow (\exists S.E)(X)$.

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^{i}, which have, under the IWA, a similar semantics.

Theorem 9 *Let Σ 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 Σ.*

Proof Observing the rules in $\Phi(C, \Sigma)$, it is clear that this program is a CoLP. Moreover, if we assume, as is not uncommon in DLs (see, e.g., [39]), that the number n in number restrictions is represented in unary notation, then the size of the CoLPs is polynomial. □

Theorem 10 *A $\mathcal{SHIQ}$ concept expression C is satisfiable w.r.t. a $\mathcal{SHIQ}$ knowledge base Σ 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. Σ, 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

$$
\begin{aligned}
M \equiv\ & \{C(x) \mid x \in C^{\mathcal{I}}, C \in \text{clos}(C, \Sigma), C \text{ a concept expression}\} \\
& \cup \{R^{\text{i}}(x, y) \mid (x, y) \in (R^{-})^{\mathcal{I}}, R^{-} \text{ or } R \text{ in clos}(C, \Sigma), R \text{ a role name}\} \\
& \cup \{R(x, y) \mid (x, y) \in R^{\mathcal{I}}, R^{-} \text{ or } R \text{ in clos}(C, \Sigma), R \text{ a role name}\}\ .
\end{aligned}
$$

One can show 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 $R(x, y) \in M \iff R^{\text{i}}(y, x) \in M$ such that the IWA 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 $\Phi(C, \Sigma)^M_U$. One can check that every rule in $\Phi(C, \Sigma)^M_U$ is satisfiable.
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, Σ).

   - If $R \in \text{clos}(C, \Sigma)$, 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 $R^{-}\text{clos}(C, \Sigma)$, then R^{i} is free. Since $(x, y) \in R^{\mathcal{I}}$, we have that $(y, x) \in (R^{-})^{\mathcal{I}}$ and thus $R^{\text{i}}(y, x) \in M$. Then, $R^{\text{i}}(y, x) \leftarrow \in \Phi(C, \Sigma)^M_U$ such that $R^{\text{i}}(y, x) \in N$. 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* $E \in \text{clos}(C, \Sigma)$. One can prove this by induction on the structure of E.

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 $Q^{\mathbf{i}}$ 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 Σ. One can show that $\mathcal{I}$ is a model of Σ and, since $C(u) \in M$, we have that $u \in C^{\mathcal{I}}$. □

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 11 *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 [39]). By Theorem 10 and Theorem 9, we can polynomially reduce such satisfiability checking to satisfiability checking under IWA w.r.t. CoLPs. □

Theorem 12 *Satisfiability checking under IWA w.r.t. CoLPs is* EXPTIME-*complete.*

Proof Membership follows from Theorem 8 and hardness from Theorem 11. □

3.3 Discussion: OASP vs. DLs

In this section, we discuss some of the advantages and disadvantages of open answer set programming versus description logics in the context of knowledge representation and reasoning.

Using CoLPs instead of $\mathcal{SHIQ}$ has the advantage of nonmonotonicity by means of negation as failure.

Example 9 Add a rule to the company example knowledge base, expressing that if Persons are not married, they work late at the office: $\text{works_late}(X) \leftarrow \text{not} \, \text{married}(X)$. Adding such a rule to our knowledge will have the effect that every open answer set includes the literal $\text{works_late}(x)$, i.e., everybody always works late. However, consecutively adding the newly acquired knowledge that everybody is actually married with a rule $\text{married}(X) \leftarrow$, will make sure that nobody ever works late according 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 [10] 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 DL axiom $\neg\text{Married} \sqsubseteq \text{Works_late}$. However, interpretations satisfying this axiom have a choice in making persons work later or not, such that adding that everybody is married monotonically reduces the number of possible models.

DLs have only a limited set of constructs while CoLPs have a flexible rule presentation which often allows for a more compact representation of knowledge than would be possible in DLs.

Example 10 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 rule

$$\text{team}(X) \leftarrow \text{has_member}(X, Y_1), \text{tech}(Y_1), \text{has_member}(X, Y_2), \text{secret}(Y_2), \text{leader}(X, Y_3), Y_1 \neq Y_3.$$

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 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). Representing such *generalized number restrictions* using DLs would be significantly harder while arguably less succinct.

Currently, a clear 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 $\mathcal{SHIQ}$, is EXPTIME-complete, practical tableau algorithms run in 2-NEXPTIME in the worst case [39]. 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., [11]) 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.

4 Related work

In [19], the language $\mathcal{L}_0$ of a program P is expanded 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 a *k-belief set* of P 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 k-belief sets are always finite. Nonetheless, the other direction is valid: every k-belief set can be written as an open answer set.

Defining k-belief sets, or open answer sets for that matter, easily leads to undecidability as was argued for k-belief sets in [37]. Interestingly, [37] 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 [37]. Consider, in this light, $\Phi(C, \Sigma)$, which basically consists of a stratified part, defining the DLs constructors, and a disjunctive part, the free rules. However, we still have decidability, emphasizing the importance of the tree model property.

Another approach to infinite reasoning, besides infinite open domains, is presented in [8], where function symbols are included in the language. *Finitary programs* are identified as a class of programs for which ground query answering is decidable, and lead to elegant formulations of, e.g., plans with unbounded planning length. Formally, they are defined as programs that are finitely recursive, i.e., every ground atom may only depend on a finite number of other ground atoms, and such that only a finite number of odd-cycles may occur in the grounded program. Neither conditions are necessary for CoLPs: the CoLP containing rules $a(X) \leftarrow f(X, Y), \text{not } b(Y)$ and $b(X) \leftarrow a(X)$, when grounded with an infinite universe, is not finitely recursive and contains infinitely many odd-cycles. Since not all finitary programs are CoLPs, both classes of programs are not directly related, and the tree model property appears to be an alternative indication of 'finitary' reasoning with possibly infinite knowledge. While ground query answering with finitary programs is decidable, unground query answering is only semi-decidable [8]. Since unground query answering (satisfiability checking) is decidable for CoLPs, CoLPs are arguably more suited for conceptual modeling. Moreover, checking whether a program is finitary is itself undecidable, in contrast with CoLPs, which are a syntactic restriction.

There are basically two lines of research that try to reconcile description logics with logic programming. The approaches in [1, 20, 26, 32, 38, 41] simulate DLs with LP, possibly with a detour to FOL, while [13, 15, 34] attempt to unite the strengths of DLs and LP by letting them coexist and interact.

In [41], the simulation of a DL with acyclic axioms in *open logic programming* is shown. An open logic program is a program with possibly undefined predicates and a FOL-theory; the semantics is the completion semantics, which is only complete for a restrictive set of programs. The openness lies in the use of undefined predicates, which are comparable to free predicates with the difference that free predicates can be expressed within the CoLP framework. 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. Note that $\mathcal{ALCN}$ is a subclass of $\mathcal{SHIQ}$, the DL that we simulated with open answer set programming.

Grosof et al. [20] imposes restrictions on the occurrence of DL constructs in terminological axioms to enable a simulation using 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, yielding that, so-called *Description Logic Programs* are incapable of handling expressive DLs. However, the forte of [20] lies in the identification of a subclass of DLs that make efficient reasoning through LPs possible.

In [1], the DL $\mathcal{ALCQI}$ is successfully translated into a disjunctive logic program. However, to take into account infinite interpretations [1] presumes, for technical reasons, the existence of function symbols, which leads, in general, to undecidability of reasoning.

Hustadt et al. [26] and Swift [38] simulate reasoning in DLs with a LP formalism by using an intermediate translation to first-order clauses. In [26], $\mathcal{SHIQ}^-$ knowledge bases, i.e., $\mathcal{SHIQ}$ knowledge bases with the requirement that roles S in $(\leq nS.C)$ have no subroles, 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.

Swift [38] translates $\mathcal{ALCQI}$ concepts 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 for $\mathcal{ALCQI}$-concept expressions; in the presence of terminological axioms this is no longer possible. The resulting program is, however, not declarative anymore such that its main contribution is that it provides an alternative reasoner for DLs, whereas CoLPs can be used both for reasoning with DLs and for a direct and elegant expression of knowledge. Furthermore, CoLPs are also interesting from a pure LP viewpoint since they constitute a decidable class of programs under the open answer set semantics.

Along the second line of research, an $\mathcal{AL}$-log [13] system consists of two subsystems: a DL knowledge base and a Datalog program, where in the latter variables may range over DL concept instances, thus obtaining a flow of information from the structural DL part to the relational Datalog part. This is extended in [34] for disjunctive Datalog and the $\mathcal{ALC}$ DL. A further generalization is attained in [15] where the particular DL can be the expressive $\mathcal{SHIF}$, $\mathcal{F}$ stands for functional restrictions, or $\mathcal{SHOIN}$. The DL knowledge base is considered as a black box that can be queried from the rules. Moreover, inferences made by rules can serve as input to the DL knowledge base as well, leading to a bidirectional flow of information.

A notable approach, which cannot be categorized in one of the two lines of research described above, although it tends towards the coexisting approach, is the SWRL [25] initiative. SWRL is a *Semantic Web Rule Language* and extends the syntax and semantics of the ontology language OWL DL with unary/binary Datalog RuleML [36], i.e., Horn-like rules. This extension is undecidable [22] but lacks, nevertheless, interesting knowledge representation mechanisms such as negation as failure.

5 Conclusions and directions for further 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 subsequently identified CoLPs for which reasoning under the open answer set semantics is decidable and EXPTIME-complete. Furthermore, CoLPs can simulate reasoning in the expressive description logic $\mathcal{SHIQ}$. CoLPs 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. 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 predicates cannot be reduced to finite answer set programming 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.

Similar to tableau algorithms for $\mathcal{SHIQ}$, we want to look into practical algorithms for CoLP satisfiability checking. However, due to minimality of open answer sets, this is expected to be more intricate than the blocking techniques used in DLs.

Acknowledgements The work is funded by the European Commission under the projects ASG, DIP, enIRaF, InfraWebs, Knowledge Web, Musing, Salero, SEKT, Seemp, SemanticGOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/I13 ; by the FFG (Österreichische ForschungsfÃűrderungsgesellschaft mbH) under the projects Grisino, RW^2, SemNetMan, SeNSE, TSC, OnTourism. Davy Van Nieuwenborgh is supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

References

1. Alsaç, G., Baral, C.: Reasoning in description logics using declarative logic progamming. Technical Report, Arizona State University, Phoenix, Arizona (2002)
2. Andréka, H., Németi, I., Van Benthem, J.: Modal languages and bounded fragments of predicate logic. J. Philos. Logic **27**(3), 217–274 (1998)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, UK (2003)
4. Baader, F., Sattler, U.: Tableau algorithms for description logics. In: Proc. of Tableaux 2000, vol. 1847 of LNAI, pp. 1–18. Springer, Berlin Heidelberg New York (2000)
5. Bechhofer, S., Goble, C., Horrocks, I.: DAML+OIL is not Enough. In: Proc. of SWWS'01, pp. 151–159. CEUR, Stanford, California (2001)
6. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.:(OWL) Web Ontology Language Reference, Stanford University, California (2004)
7. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. 34–43 (May 2001)
8. Bonatti, P.A.: Reasoning with infinite stable models. Artif. Intell. **156**, 75–111 (2004)
9. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives of Mathematical Logic. Springer, Berlin Heidelberg New York (1997)
10. Bry, F., Schaffert, S.: An entailment relation for reasoning on the web. In: Proc. of RuleML, LNCS, pp. 17–34. Springer, Berlin Heidelberg New York (2003)
11. Calvanese, D., De Giacomo, G., Lenzerini, M.: 2ATAs make DLs easy. In: Proc. of the 2002 Description Logic Workshop (DL'02). Toulouse, France (2002)
12. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
13. Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.: AL-log: Integrating datalog and description logics. J. Intell. Syst. **10**, 227–252 (1998)
14. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Nonmonotonic description logic programs: implemenation and experiments. In: Proc. of LPAR 2004, 3452 in LNAI, pp. 511–527. Springer, Berlin Heidelberg New York (2005)
15. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with DLs for the semantic web. In: Proc. of KR 2004, pp. 141–151. Morgan Kaufmann, San Mateo, California (2004)

16. Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., Klein, M.: OIL in a Nutshell. In: Proc. of EKAW 2000, LNAI. Springer, Berlin Heidelberg New York (2000)
17. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.F.: OIL: An ontology infrastructure for the semantic web. IEEE Intell. Syst. **16**(2), 38–45 (2001)
18. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of ICLP 1988, pp. 1070–1080. MIT, Cambridge, Massachusetts (1988)
19. Gelfond, M., Przymusinska, H.: Reasoning in open domains. In: Logic Programming and Non-Monotonic Reasoning, pp. 397–413. MIT, Cambridge, Massachusetts (1993)
20. Grosof, B., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proc. of Twelfth International World Wide Web Conference (WWW 2003), pp. 48–57. ACM, Budapest, Hungary (2003)
21. Halpin, T.: Information Modeling and Relational Databases. Morgan Kaufmann, San Mateo, California (2001)
22. Horrocks, I., Patel-Schneider, P.F.: A proposal for an OWL rules language. Proc. of WWW 2004. ACM, New York (2004)
23. Horrocks, I., Sattler, U.: A description logic with transitive and converse roles and role hierarchies. LTCS-Report 98–05 (1998)
24. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Proc. of LPAR'99, LNCS, pp. 161–180. Springer, Berlin Heidelberg New York (1999)
25. Horrocks, I., Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, (May 2004)
26. Hustadt, U., Motik, B., Sattler, U.: Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. FZI-Report 1-8-11/03 (2003)
27. Jarrar, M., Meersman, R.: Formal ontology engineering in the DOGMA approach. Proc. of CoopIS/DOA/ODBASE, vol. 2519 of LNCS, pp. 1238–1254. Springer, Berlin Heidelberg New York (2002)
28. Levy, A.Y., Rousset, M.: CARIN: A representation language combining horn rules and description logics. In: Proc. of ECAI'96, pp. 323–327, Budapest, Hungary (1996)
29. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Syst. **2**(4), 526–541 (2001)
30. Lutz, C., Sattler, U.: Mary likes all cats. In: Baader, F., Sattler, U. (eds.) Proc. of DL 2000, number 33 in CEUR-WS, pp. 213–226, Aachen, Germany (2000)
31. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. In: Proc. of ISWC 2004, number 3298 in LNCS, pp. 549–563. Springer, Berlin Heidelberg New York (2004)
32. Motik, B., Volz, R., Maedche, A.: Optimizing query answering in description logics using disjunctive deductive databases. In: Proc. of KRDB'03, pp. 39–50, Humberg, Germany (2003)
33. Rector, A.L., Wroe, C., Rogers, J., Roberts, A.: Untangling taxonomies and relationships: personal and practical problems in loosely coupled development of large ontologies. In: Proc. of K-CAP 2001, pp. 139–146. ACM, USA (2001)
34. Rosati, R.: Towards expressive KR systems integrating datalog and description logics: preliminary report. In: Proc. of DL'99, pp. 160–164, Linkping, Sweden (1999)
35. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. J. Web Sem. **3**(1), pp. 41–60 (2005)
36. The Rule Markup Initiative. http://www.ruleml.org
37. Schlipf, J.: Some remarks on computability and open domain semantics. In Proc. of the Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming, Washington, District of Columbia (1993)
38. Swift, T.: Deduction in ontologies via answer set programming. In: Lifschitz, V., Niemelä, I. (eds) Proc. of LPNMR 2004, vol. 2923 of LNCS, pp. 275–288. Springer, Berlin Heidelberg New York (2004)
39. Tobies S.: Complexity results and practical algorithms for logics in knowledge representation. PhD thesis, RWTH-Aachen, Germany (2001)
40. Uschold, M., Grüninger, M.: Ontologies: Principles, methods, and applications. Knowl. Eng. Rev. **11**(2), 93–155 (1996)
41. Van Belleghem, K., Denecker, M., De Schreye, D.: A strong correspondence between DLs and open logic programming. In: Proc. of ICLP 1997, pp. 346–360. MIT, Cambridge, Massachusetts (1997)
42. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. J. ACM **23**(4), 733–742 (1976)

43. Van Gelder, A., Schlipf, J.: Commonsense axiomatizations for logic programs. J. Log. Program. **17**, 161–195 (1993)
44. Vardi, M.Y.: Why is modal logic so robustly decidable? Technical Report TR97-274, Rice University, Houston, Texas, April 12 (1997)
45. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Proc. of ICALP'98, pp. 628–641. Springer, Berlin Heidelberg New York (1998)