I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ARBEITSBEREICH WISSENSBASIERTE SYSTEME

HYBRID REASONING WITH FOREST LOGIC PROGRAMS

Cristina Feier Stijn Heymans

INFSYS RESEARCH REPORT 184-08-14.

DECEMBER 2008

Institut für Informationssysteme
AB Wissensbasierte Systeme
Technische Universität Wien
Favoritenstrassße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TU WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 184-08-14., DECEMBER 2008

HYBRID REASONING WITH FOREST LOGIC PROGRAMS

Cristina Feier[1] and Stijn Heymans[2]

Abstract. Open Answer Set Programming (OASP) is an attractive framework for integrating ontologies and rules. Although several decidable fragments of OASP have been identified, few reasoning procedures exist. In this paper, we provide a sound, complete, and terminating algorithm for satisfiability checking w.r.t. forest logic programs, a fragment where rules have a tree shape and allow for inequality atoms and constants. We further introduce f-hybrid knowledge bases, a hybrid framework where $\mathcal{SHOQ}$ knowledge bases and forest logic programs co-exist, and we show that reasoning with such knowledge bases can be reduced to reasoning with forest logic programs only. We note that f-hybrid knowledge bases do not require the usual (weakly) DL-safety of the rule component, providing thus a genuine alternative approach to hybrid reasoning.

[1]Institute of Information Systems, Knowledge-Based Systems Group, Vienna University of Technology, Favoritenstraße 9-11, A-1040 Vienna, Austria. E-mail: feier@kr.tuwien.ac.at.

[2]Institute of Information Systems, Knowledge-Based Systems Group, Vienna University of Technology, Favoritenstraße 9-11, A-1040 Vienna, Austria. E-mail: heymans@kr.tuwien.ac.at.

Acknowledgements: This work is partially supported by the Austrian Science Fund (FWF) under the projects *Distributed Open Answer Set Programming (FWF P20305)* and *Reasoning in Hybrid Knowledge Bases (FWF P20840)*.

This Report is a Preliminary version.

Contents

1 Introduction

Integrating Description Logics (DLs) with rules for the Semantic Web has received considerable attention with approaches such as *Description Logic Programs* [11], *DL-safe rules* [27], $\mathcal{DL}$+*log* [29], *dl-programs* [6], *Description Logic Rules* [21], and Open Answer Set Programming (OASP) [16]. OASP combines attractive features from the DL and the Logic Programming (LP) world: an open domain semantics from the DL side allows for stating generic knowledge, without the need to mention actual constants, and a rule-based syntax from the LP side supports nonmonotonic reasoning via *negation as failure*.

Several decidable fragments of OASP were identified by syntactically restricting the shape of logic programs, while carefully safe-guarding enough expressiveness for integrating rule- and ontology-based knowledge. Notable fragments are *Conceptual Logic Programs (CoLPs)* [13] that are able to simulate reasoning in the DL $\mathcal{SHIQ}$ and *Forest Logic Programs (FoLPs)* [14] that are expressive enough to deal with $\mathcal{SHOQ}$. Note that both $\mathcal{SHOQ}$ and $\mathcal{SHIQ}$ are close family of $\mathcal{SHOIN}(\mathbf{D})$, the DL underlying the Web Ontology language OWL-DL [31]. A serious shortcoming of these decidable fragments is their lack of effective reasoning procedures. In [7], we took a first step in mending this by providing a sound and complete algorithm for *simple CoLPs*. Simple CoLPs are a particular type of CoLPs that disallow the use of inverse predicates, inequality and allow just a restricted form of literal cyclity, but that are still expressive enough to simulate the DL $\mathcal{ALCH}$.

In this report, we extend the algorithm of [7] to *Forest Logic Programs*, an extension of simple CoLPs with constants and inequality and no cyclicity restriction. FoLPs are able to simulate the DL $\mathcal{SHOQ}$. Furthermore, they serve well as an underlying integration vehicle for ontologies and rules. In order to illustrate this, we define *f-hybrid knowledge bases (fKBs)*, consisting of a $\mathcal{SHOQ}$ knowledge base and a rule component that is a FoLP, with a nonmonotonic semantics similar to the semantics of *g-hybrid knowledge bases* [12], *r-hybrid knowledge bases* [30], and $\mathcal{DL}$+*log* [29]. Our approach differs in two points with current other proposals:

- In contrast with Description Logic Programs, DL-safe rules, and Description Logic Rules, f-hybrid knowledge bases have, in line with traditional logic programming paradigms, a minimal model semantics for the rule component, thus allowing for nonmonotonic reasoning.
- To ensure effective reasoning, our approach does not rely on a (weakly) DL-safeness condition such as [27, 29, 30], which restricts the interaction of the rule component with the DL component. Instead, we rely on a translation of the hybrid knowledge to FoLPs.

The major contributions of the paper can be summarized as follows:

- We define in Section 4 a nondeterministic algorithm for deciding satisfiability w.r.t. FoLPs, inspired by tableaux-based methods from DLs. We show that this algorithm is terminating, sound, complete, and runs in 2-NEXPTIME. The algorithm is non-trivial from two perspectives: both the minimal model semantics of OASP, compared to the model semantics of DLs, as well as the open domain assumption, compared to the closed domain assumption of ASP [8], pose specific challenges.
- We show in Section 5 that FoLPs are expressive enough to simulate the DL $\mathcal{SHOQ}$ with *fKBs* an alternative characterization for hybrid representation and (nonmonotonic) reasoning of knowledge, that supports a tight integration of ontologies and rules.

2 Preliminaries

We recall the open answer set semantics from [16]. *Constants* $a, b, c, \ldots$, *variables* $x, y, \ldots$, *terms* $s, t, \ldots$, and *atoms* $p(t_1, \ldots, t_n)$ are defined as usual. We further allow for *equality atoms* $s = t$. A *literal* is an atom L or a negated atom $not\ L$. An *inequality literal* $not\ (s = t)$ will often be denoted with $s \neq t$. An atom (literal) that is not an equality atom (inequality literal) will be called a *regular atom (literal)*. For a set α of literals or (possibly negated) predicates, $\alpha^+ = \{l \mid l \in \alpha, l \text{ an atom or a predicate}\}$ and $\alpha^- = \{l \mid not\ l \in \alpha, l \text{ an atom or a predicate}\}$. For example, $\{a, not\ b, c \neq d\}^+ = \{a\}$ and $\{a, not\ b, c \neq d\}^- = \{b, c = d\}$. For a set X of atoms, $not\ X = \{not\ l \mid l \in X\}$. For a set of (possibly negated) predicates α, we will often write $\alpha(x)$ for $\{a(x) \mid a \in \alpha\}$ and $\alpha(x, y)$ for $\{a(x, y) \mid a \in \alpha\}$.

A *program* is a countable set of rules $\alpha \leftarrow \beta$, where α is a finite set of regular literals and β is a finite set of literals. The set α is the *head* of the rule and represents a disjunction, while β is called the *body* and represents a conjunction. If $\alpha = \emptyset$, the rule is called a *constraint*. *Free rules* are rules $q(t_1, \ldots, t_n) \vee not\ q(t_1, \ldots, t_n) \leftarrow$ for terms $t_1, \ldots, t_n$; they enable a choice for the inclusion of atoms. We call a predicate q *free* in a program if there is a free rule $q(x_1, \ldots, x_n) \vee not\ q(x_1, \ldots, x_n) \leftarrow$ in the program, where $x_1, \ldots, x_n$ are variables. Atoms, literals, rules, and programs that do not contain variables are *ground*. For a rule or a program X, let $cts(X)$ be the constants in X, $vars(X)$ its variables, and $preds(X)$ its predicates with $upreds(X)$ the unary and $bpreds(X)$ the binary predicates. A *universe* U for a program P is a non-empty countable superset of the constants in P: $cts(P) \subseteq U$. We call P_U the ground program obtained from P by substituting every variable in P by every possible element in U. Let $\mathcal{B}_P$ ($\mathcal{L}_P$) be the set of regular atoms (literals) that can be formed from a ground program P.

An *interpretation* I of a ground P is any subset of $\mathcal{B}_P$. We write $I \models p(t_1, \ldots, t_n)$ if $p(t_1, \ldots, t_n) \in I$ and $I \models not\ p(t_1, \ldots, t_n)$ if $I \not\models p(t_1, \ldots, t_n)$. Furthermore, for ground terms s and t we write $I \models s = t$ if $s = t$ and $I \models not\ s = t$ or $I \models s \neq t$ if $s \neq t$. 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 *GL-reduct* [8] w.r.t. I is defined as P^I, where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P, $I \models not\ \beta^-$, and $I \models \alpha^-$. I is an *answer set* of a ground P if I is an answer set of P^I.

In the following, a program is assumed to be a finite set of rules; infinite programs only appear as byproducts of grounding a finite program with an infinite universe. An *open interpretation* of a program P is a pair (U, M) where U is a universe for P and M is an interpretation of P_U. An *open answer set* of P is an open interpretation (U, M) of P with M an answer set of P_U. An n-ary predicate p in P is *satisfiable* if there is an open answer set (U, M) of P and a $(x_1, \ldots, x_n) \in U^n$ such that $p(x_1, \ldots, x_n) \in M$.

We introduce some notations for trees which extend those in [33]. Let $\cdot$ be a concatenation operator between different symbols such as constants or natural numbers. A *tree* T with root c (also denoted as T_c), where c is a specially designated constant, has as nodes sequences of the form $c \cdot s$, where s is a (possibly empty) sequence of positive integers formed with the help of the concatenation operator; for $x \cdot d \in T$, $d \in \mathbb{N}^*$[1], we must have that $x \in T$. For example a tree with root c and 2 successors will be denoted as $\{c, c \cdot 1, c \cdot 2\}$ or $\{c, c1, c2\}$.

For a node $x \in T$, we call $succ_T(x) = \{x \cdot n \in T \mid n \in \mathbb{N}^*\}$, *successors* of x. As the successorship relation is captured in the codification of the nodes, a tree is literally the set of its nodes. The *arity* of a tree is the maximum amount of successors any node has in the tree. The set $A_T = \{(x, y) \mid x, y \in T, \exists n \in \mathbb{N}^* :$

[1] $\mathbb{N}^*$ is the set of positive integers

$y = x \cdot n\}$ denotes the set of arcs of a tree T. We define a partial order $\leq_T$ on a tree T such that for $x, y \in T$, $x \leq_T y$ iff x is a prefix of y. As usual, $x <_T y$ if $x \leq_T y$ and $y \not\leq_T x$. A *branch* IB in a tree T is a prefix-closed subset of T such that $\forall x \neq y \in IB : |x| \neq |y|$. A *path from* x *to* y in T, where $x <_T y$ denoted with $path_T(x, y)$, is a subset of T which contains all nodes which are at the same time greater or equal to x in T and lesser or equal to y in T according to the partial order relation. A branch B in a tree T_c is a maximal path (there is no path in T_c which strictly contains it) which contains the root c. We denote the *subtree* of T at x by $T[x]$, i.e., $T[x] = \{y \in T \mid x \leq_T y\}$.

A *labeled tree* is a pair (T, t) where T is a tree and $t : T \to \Sigma$ is a labeling function; sometimes we will identify the tree (T, t) with t. For a labeled tree $t : T \to \Sigma$, the subtree of t at $x \in T$ is $t[x] : T[x] \to \Sigma$ such that $t[x](y) = t(y)$ for $y \in T[x]$.

A *forest* F is a set of trees $\{T_c \mid c \in C\}$, where C is a set of distinguished constants. The set of nodes N_F of a forest F and the set of arcs A_F of F are defined as follows: $N_F = \cup_{T \in F} T$ and $A_F = \cup_{T \in F} A_T$. For a node $x \in F$, we denote with $succ_F(x) = succ_T(x)$ for $x \in T$, the set of successors of x in F. Also, as for trees, we define a partial order relationship $\leq_F$ on the nodes of a forest F where $x \leq_F y$ iff $x \leq_T y$ for some tree T in F.

A labeled forest f is a tuple (F, f) where F is a forest and $f : N_F \to \Sigma$ is a labeling function; sometimes we will identify the forest (F, f) with f. A labeled forest (F, f), with $F = \{T_c \mid c \in C\}$, induces a set of labeled trees $\{(T_c, t_c) \mid c \in C\}$, where $t_c : T_c \to \Sigma$ and $t_c(x) = f(x)$, for any $x \in T_c$. Figure 1 depicts a labeled forest which contains two labeled trees t_a and t_b (their roots are a and b respectively).

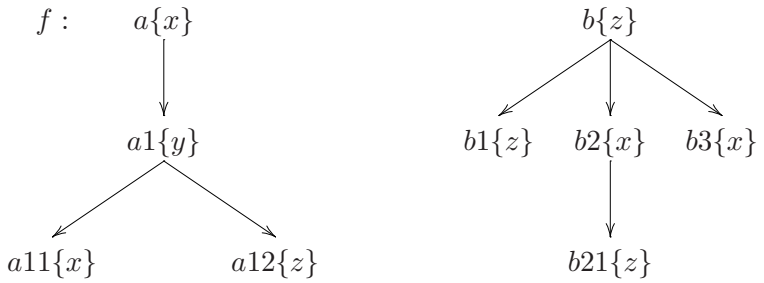

Figure 1: A Simple Labeled Forest

An extended forest EF is a tuple $\langle F, ES \rangle$ where $F = \{T_c \mid c \in C\}$ is a forest and ES is a binary relation which contains tuples of the form (x, y) where $x \in N_F$ and $y \in C$, i.e., ES relates nodes of the forest with roots of trees in the forest. ES extends the successorship relation: $succ_{EF}(x) = \{y \mid y \in succ_F(x)$ or $(x, y) \in ES\}$.

Figure 2 depicts an extended forest.

The presence of ES gives rise to so-called extended trees in EF, where such a tree (actually, a particular type of graph) is one of $T_c \in F$, extended with the arcs $\{(x, d) \mid (x, d) \in ES, x \in T\}$ and with the nodes $\{d \mid (x, d) \in ES, x \in T\}$. The extension of T_c in EF is denoted with T_c^{EF}. For example, the extension of T_a in EF from Figure 2 contains the extra arc $(a12, b)$ and the extension of T_b in EF contains the extra arcs (b, a) and $(b2, a)$. An extended subtree of T^{EF} with root x is denoted with $T^{EF}[x]$: it is defined (as a graph) as the extension of $T[x]$ with the arcs $\{(y, c) \mid (y, c) \in ES, y \in T[x]\}$ and with the nodes $\{c \mid (y, c) \in ES, y \in T[x]\}$. Finally, by $N_{EF} = N_F$ we denote the set of nodes of an extended forest EF

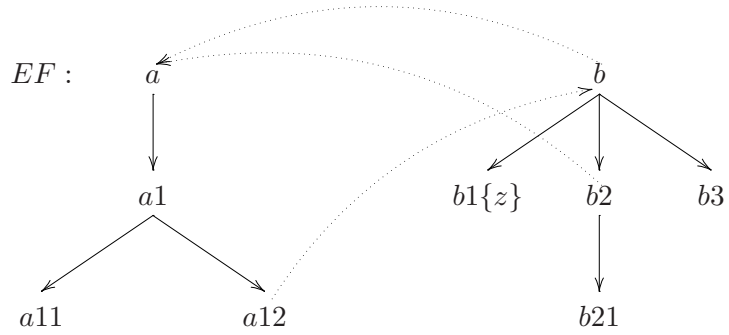

Figure 2: An extended forest

and by $A_{EF} = A_F \cup ES$ the set of arcs of EF.

A labeled extended forest is a tuple $\langle EF, ef \rangle$ where EF is an extended forest and $ef : N_{EF} \to \Sigma$ is a labeling function; sometimes we will identify the extended forest $\langle EF, ef \rangle$ with ef. A labeled extended forest can be seen as a set of labeled extended trees, where a labeled extended tree is a tuple (T^{ef}, t^{ef}), where T^{ef} is an extended tree and $t^{ef} : T^{ef} \to \Sigma$ is a labeling function defined such that $t^{ef}(x) = ef(x)$, for $x \in T^{ef}$. For a labeled extended tree $t^{ef} : T^{ef} \to \Sigma$, the subtree of t^{ef} at $x \in T$ is $t^{ef}[x] : T^{ef}[x] \to \Sigma$ such that $t^{ef}[x](y) = t^{ef}(y)$ for $y \in T^{ef}[x]$.

Figure 3 depicts an extended labeled forest (a labeled version of the extended forest from Figure 2).

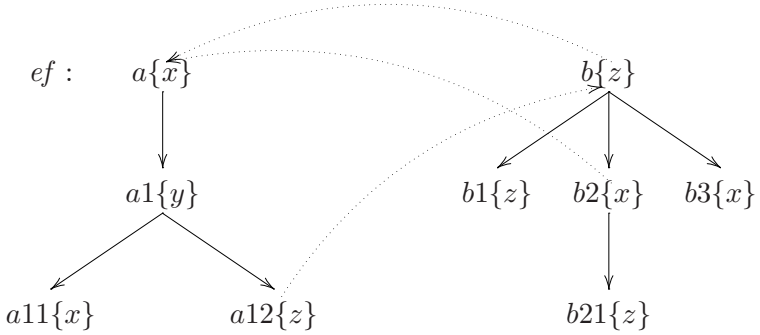

Figure 3: A labeled extended forest

We introduce the operation of replacing in a labeled extended forest ef an extended subtree $t^{ef}[x]$ with another extended subtree $t^{ef}[y]$, where both x and y are from N_{EF}, and denote this operation with $replace_{ef}(x, y)$. Figure 4 describes the result of applying the replace operation on the extended forest from Figure 2 with two different sets of operators. In the first case, $t_b^{ef}[b2]$ is replaced with $t_a^{ef}[a1]$, while in the second case $t_a^{ef}[a1]$ is replaced with $t_a^{ef}[a12]$. Note that the names of nodes of the subtree which is replaced are not changed with the names of the nodes from the replacing subtree, but new names are generated for the new nodes in concordance with the naming scheme for nodes of that tree. Also, observe how in the first replacement one of the 'extra' arcs of t_b, $(b2, a)$, is dropped (it was part of the replaced extended subtree) and a new 'extra' arc is introduced, $(b22, b)$, which mirrors the arc $(a12, b)$ from the replacing extending

subtree. Similarly, in the second transformation, $(a12, b)$ is dropped and $(a1, b)$ is introduced.

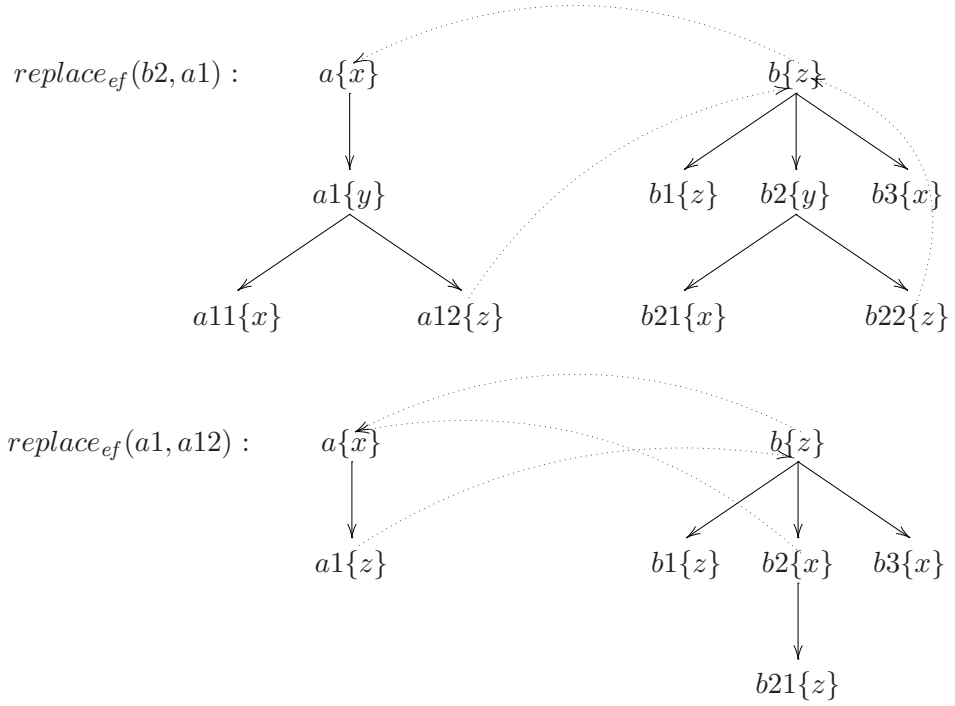

Figure 4: Two applications of the replace operator on ef

Finally, a directed graph G is defined as usually by its sets of nodes V and arcs A. We introduce two graph-related notations: $paths_G$ denotes the set of paths in G, where each path is a tuple of nodes from V: $paths_G = \{(x_1, \ldots, x_n) \mid ((x_i, x_{i+1}) \in A)_{1 \leq i < n}\}$, and $connected_G$ denotes the set of pairs of connected nodes from V: $connected_G = \{(x, y) \mid \exists Pt = (x_1, \ldots, x_n) \in paths_G : x_1 = x \wedge x_n = y\}$.

3 Forest Logic Programs

Forest Logic Programs (FoLPs) were introduced in [14] as a syntactical fragment of OASP. FoLPs are a generalization of Conceptual Logic Programs [13] which are logic programs with tree-shaped rules for which satisfiability checking under the open answer set semantics is decidable. FoLPs impose the same structure for rules as CoLPs, but also allow for constants.

Definition 3.1 A *forest logic program (FoLP)* is a program with only unary and binary predicates, and such that a rule is either a *free rule*

$$a(s) \vee not\ a(s) \leftarrow \text{ or } f(s,t) \vee not\ f(s,t) \leftarrow \tag{1}$$

where s and t are terms such that if s and t are both variables, they are different[2], a *unary rule*

$$r : a(s) \leftarrow \beta(s), (\gamma_m(s, t_m), \delta_m(t_m))_{1 \leq m \leq k}, \psi \tag{2}$$

[2]A rule $f(X, X) \vee not\ f(X, X) \leftarrow$ is not allowed.

where s and t_m, $1 \leq m \leq k$, are terms (again, if both s and t_m are variables, they are different; similarly for t_i and t_j), where

1. $\psi \subseteq \bigcup_{1 \leq i \neq j \leq k} \{t_i \neq t_j\}$ and $\{\neq\} \cap \gamma_m = \emptyset$ for $1 \leq m \leq k$,
2. $\forall t_i \in vars(r) : \gamma_i^+ \neq \emptyset$, i.e., for variables t_i there is a positive atom that connects s and t_i,

or a *binary rule*

$$f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t) \tag{3}$$

with $\{\neq\} \cap \gamma = \emptyset$ and $\gamma^+ \neq \emptyset$ if t is a variable (s and t are different if both are variables), or a *constraint*

$$\leftarrow a(s) \text{ or } \leftarrow f(s,t) \tag{4}$$

where s and t are different if both are variables).

The constraints can be left out of the fragment without losing expressivity. Indeed, a constraint $\leftarrow body$ can be replaced by a rule of the form $constr(x) \leftarrow not\ constr(x), body$, for a new predicate $constr$. As their name suggests FoLPs have the *forest model property*: every open answer set can be written as a set of trees.

Definition 3.2 Let P be a program. A predicate $p \in upreds(P)$ is *forest satisfiable* w.r.t. P if there is an open answer set (U, M) of P and there is an extended forest $EF \equiv (\{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\}, ES)$, where ε is a constant, possibly one of the constants appearing in P[3], and a labeling function $\mathcal{L} : \{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\} \cup A_{EF} \rightarrow 2^{preds(P)}$ such that

- $U = N_{EF}$, and
- $p \in \mathcal{L}(\varepsilon)$,
- $z \cdot i \in T, T \in EF, i > 0$, iff there is some $f(z, z \cdot i) \in M$, $z \in T$, and
- for $y \in T, T \in EF$, $q \in upreds(P)$, $f \in bpreds(P)$, we have that
  - $q(y) \in M$ iff $q \in \mathcal{L}(y)$, and
  - $f(y,u) \in M$ iff $(u = y \cdot i \vee u \in cts(P)) \wedge f \in \mathcal{L}(y,u)$.

We call such a (U, M) a *forest model* and a program P has the *forest model property* if the following property holds:

If $p \in upreds(P)$ is satisfiable w.r.t. P then p is forest satisfiable w.r.t. P.

Proposition 3.3 *FoLPs have the forest model property [15].*

In [7], we introduced the class of simple Conceptual Logic Programs. It is easy to see that every simple CoLP is an FoLP. As satisfiability checking w.r.t. simple Conceptual Logic Programs is EXPTIME-hard, the following property follows:

Proposition 3.4 *Satisfiability checking w.r.t. FoLPs is* EXPTIME*-hard.*

[3]Note that in this case $T_\varepsilon \in \{T_a \mid a \in cts(P)\}$. Thus, the extended forest contains for every constant from P a tree which has as root that specific constant and possibly, but not necessarily, an extra tree with unidentified root node.

4 An Algorithm for Forest Logic Programs

In this section, we define a sound, complete, and terminating algorithm for satisfiability checking w.r.t. FoLPs. In [15] it has been shown that several restrictions of FoLPs which have the finite model property are decidable, but there was no result so far regarding the whole fragment. Thus, the algorithm described in this section also establishes a decidability result for FoLPs.

For every non-free predicate q and a FoLP P, let P_q be the rules of P that have q as a head predicate. For a predicate p, $\pm p$ denotes p or $not\ p$, whereby multiple occurrences of $\pm p$ in the same context will refer to the same symbol (either p or $not\ p$). The negation of $\pm p$ (in a given context) is $\mp p$, that is, $\mp p = not\ p$ if $\pm p = p$ and $\mp p = p$ if $\pm p = not\ p$.

The basic data structure for our algorithm is a *completion structure*.

Definition 4.1 A *completion structure for a FoLP* P is a tuple $\langle EF, G, \mathrm{CT}, \mathrm{ST}, bl\rangle$ where $EF = \langle F, ES\rangle$ is an extended forest which together with the labeling functions CT and ST and with the set bl of blocking pairs is used to represent/construct a tentative forest model. $G = \langle V, A\rangle$ is a directed graph with vertices $V \subseteq \mathcal{B}_{P_{N_{EF}}}$ and arcs $A \subseteq \mathcal{B}_{P_{N_{EF}}} \times \mathcal{B}_{P_{N_{EF}}}$ which is used to keep track of dependencies between elements of the constructed model (the atom dependency graph of $P_{N_{EF}}$): V represents the tentative model, while N_{EF} represents the tentative universe. Below the signature and the role for each labeling function is given:

- The *content* function $\mathrm{CT} : N_{EF} \cup A_{EF} \rightarrow 2^{preds(P) \cup not\ (preds(P))}$ maps a node of the extended forest to a set of (possibly negated) unary predicates and an arc of the extended forest to a set of (possibly negated) binary predicates such that $\mathrm{CT}(x) \in upreds(P) \cup not(upreds(P))$ if $x \in N_{EF}$, and $\mathrm{CT}(x) \in bpreds(P) \cup not(bpreds(P))$ if $x \in A_{EF}$. Every presence of a non-negated predicate symbol p in the content of some node/arc x of EF indicates that $p(x)$ is part of the tentative model represented by EF.

- The *status* function $\mathrm{ST} : \{(x, q) \mid q \in \mathrm{CT}(x), x \in N_{EF} \cup A_{EF}\} \cup \{(x, q) \mid not\ q \in \mathrm{CT}(x), x \in A_{EF}\} \cup \{(x, not\ q, r) \mid not\ q \in \mathrm{CT}(x), x \in N_{EF}, r \in P_q\} \rightarrow \{exp, unexp\}$ attaches to every (possibly negated) predicate which appears in the content of a node/arc x of EF a status value which indicates whether the predicate has already been expanded in that node/edge. As it will be indicated later, the completion structure is evolved such that the presence of any (possibly negated) predicate symbol in the content of some node/arc is justified, so it is necessary to keep track which predicate symbols have already been justified in every node/arc of EF. As negative unary predicates have to be justified by showing that no rule which defines them can be applied and this will be done in potentially more than one step, the function takes as an argument also a rule for such negated predicate symbols.

The last component of a completion structure, bl, is a set of pairs of elements from N_{EF}, which contains the so-called blocking pairs of the completion structure. The presence of a pair of nodes (x, y) in bl indicates that the predicate symbols present in $\mathrm{CT}(y)$ can be justified in a similar way as the predicate symbols in $\mathrm{CT}(x)$.

An *initial completion structure* for checking satisfiability of a unary predicate p w.r.t. a FoLP P is a completion structure $\langle EF, G, \mathrm{CT}, \mathrm{ST}, bl\rangle$ with $EF = \langle F, ES\rangle$, $F = \{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\}$, where ε is a constant, possibly in $cts(P)$, and $T_x = \{x\}$, for every $x \in cts(P) \cup \{\varepsilon\}$, $ES = \emptyset$, $G = \langle V, A\rangle$, $V = \{p(\varepsilon)\}$, $A = \emptyset$, $\mathrm{CT}(\varepsilon) = \{p\}$, $\mathrm{ST}(\varepsilon, p) = unexp$, and $bl = \emptyset$.

In the definition above the forest is initialized with the set of single-node trees having as root a constant appearing in P and possibly a new single-node tree with an anonymous root. The root of the anonymous tree,

in case this exists, should contain p, the predicate which one tries to prove that it is satisfiable. Otherwise the root of one of the other trees should contain p. G is initialized to the graph with a single vertex $p(\varepsilon)$. There are no blocking pairs in the initial completion structure.

In the following, we will show how to expand an initial completion structure to prove the satisfiability of a unary predicate p w.r.t. a FoLP P, how to determine when no more expansion is needed (e.g., like in the case of *blocking*), and under what circumstances a *clash* occurs. In particular, *expansion rules* will evolve a completion structure, starting with a guess for an initial completion structure for checking satisfiability of p w.r.t. P, to a complete clash-free structure that corresponds to a finite representation of an open answer set in case p is satisfiable w.r.t. P. *Applicability rules* state the necessary conditions such that these expansion rules can be applied.

4.1 Expansion Rules

The expansion rules will need to update the completion structure whenever in the process of justifying a literal l in the current model, a new literal $\pm p(z)$ has to be considered (either as being part of the model, in case the literal is an atom, or as not being part of the model, in case the literal is a negated atom). This means that $\pm p$ has to be inserted in the content of z in case it is not already there and marked as unexpanded, and in case $\pm p(z)$ is an atom, it has to be ensured that it is a node in G and if l is also an atom, a new arc from l to $\pm p(z)$ should be created to capture the dependencies between the two elements of the model. More formally:

- if $\pm p \notin \text{CT}(z)$, then $\text{CT}(z) = \text{CT}(z) \cup \{\pm p\}$ and $\text{ST}(z, \pm p) = unexp$ in case $\pm p = p$ or $p \in bpreds(P)$ and $\text{ST}(z, \pm p, r) = unexp$, for all $r \in P_p$ in case $\pm p = not\ p$ and $p \in upreds(P)$,
- if $\pm p = p$ and $\pm p(z) \notin V$, then $V = V \cup \{\pm p(z)\}$,
- if $l \in \mathcal{B}_{P_{N_{EF}}}$ and $\pm p = p$, then $A = A \cup \{(l, \pm p(z))\}$.

As a shorthand, we denote this sequence of operations as $update(l, \pm p, z)$; more general, $update(l, \beta, z)$ for a set of (possibly negated) predicates β, denotes $\forall \pm a \in \beta, update(l, \pm a, z)$.

In the following, for a completion structure $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$, let $x \in N_{EF}$ and $(x, y) \in A_{EF}$ be the node, respectively arc, under consideration.

4.1.1 (i) Expand unary positive.

For a unary positive predicate (non-free) $p \in \text{CT}(x)$ such that $\text{ST}(x, p) = unexp$,

- nondeterministically choose a rule $r \in P_p$ of the form (2) such that s (the term in the head of the rule) unifies with x. The rule will be used to motivate the presence of $p(x)$ in the tentative open answer set.
- for the β in the body of r, $update(p(x), \beta, x)$,
- pick up (or define when needed) k successors for x, $(y_m)_{1 \le m \le k}$, such that:
  - for every $1 \le i, j \le k$ such that $t_i \neq t_j \in \psi$: $y_i \neq y_j$;
  - for every $1 \le m \le k$:
    - $y_m \in succ_{EF}(x)$, or

  - y_m is defined as a new successor of x in the tree T_c, where $x \in T_c$: $y_m = x \cdot n$, where $n \in \mathbb{N}^*$ s.t. $x \cdot n \notin succ_{EF}(x)$, and $T_c = T_c \cup \{y_m\}$, or
  - y_m is defined as a new successor of x in EF in the form of a constant: $y_m = a$, where a is a constant from $cts(P)$ s.t. $a \notin succ_{EF}(x)$. In this case also add (x, a) to ES: $ES = ES \cup (x, a)$.

- for every successor y_m of x, $1 \leq m \leq k$: $update(p(x), \gamma_m, (x, y_m))$ and $update(p(x), \delta_m, y_m)$.
- set $\mathrm{ST}(x, p) = exp$.

4.1.2 (ii) Expand unary negative.

In general, justifying a negative unary literal $not\ p \in \mathrm{CT}(x)$ (or in other words, the absence of $p(x)$ in the constructed model) implies that the body of every ground rule which defines $p(x)$ has to be refuted. The body of such a ground rule can be either:

- locally refuted (a literal from $\beta(x)$ has to be refuted which amounts to the fact that a certain $\pm q \in \beta$ does not appear in $\mathrm{CT}(x)$)
- depending on the particular grounding of the rule, it has to be refuted in one of the outgoing arcs of x, or in some successor of x. In other words a certain $\pm f \in \delta_m$ should not appear in $\mathrm{CT}(x, y_j)$, where y_j is a successor of x or a $\pm q \in \gamma_m$ should not appear in $\mathrm{CT}(y_j)$, where again y_j is a successor of x.
- refuted by the fact that there is no valid assignment of successors of x in the completion to successors of s in the rule which fulfill the inequalities in the rule, due to an insufficient number of such successors.

Formally, for a unary negative predicate (non-free) $not\ p \in \mathrm{CT}(x)$ and a rule $r \in P_p$ of the form (2) such that x unifies with s (s is the term from the head of the rule) and $st(x, not\ p, r) = unexp$ do one of the following:

- choose a $\pm q \in \beta$, and $update(not\ p(x), \mp q, x)$. Also set $\mathrm{ST}(x, not\ p, r) = exp$, or
- if
  - for all $p \in upreds(P)$, $p \in \mathrm{CT}(x)$ or $not\ p \in \mathrm{CT}(x)$, and
  - for all $p \in \mathrm{CT}(x)$, $\mathrm{ST}(p, x) = exp$

  then for all $y_{i_1}, \ldots, y_{i_k}$ such that $(1 \leq i_j \leq n)_{1 \leq j \leq k}$, where $succ_{EF}(x) = \{y_1, \ldots y_n\}$: if for all $1 \leq j, l \leq k$, $t_j \neq t_l \in \psi \Rightarrow y_{i_j} \neq y_{i_l}$, then do one of the following:
  - for some m, $1 \leq m \leq k$, pick up a binary (possibly negated) predicate symbol $\pm f$ from δ_m and $update(not\ p(x), \mp f, (x, y_{i_m}))$, or
  - for some m, $1 \leq m \leq k$, pick up a unary negated predicate symbol $not\ q$ from γ_m and $update(not\ p(x), q, y_{i_m})$.

  Set $\mathrm{ST}(x, not\ p, r) = exp$.

One can see that once the body of a ground unary rule derived from a unary rule $r \in P_p$, which is grounded such that s, the term in the head, is substituted with x, the current node (whenever possible), is locally refuted, the bodies of all similar groundings (similar in the sense that s is substituted with x) of this rule are locally refuted, too. This is not the case for the other two refutation cases. All possible groundings have to be considered and this is not possible until all the successors of x are known. The local refutation case is captured in the first part of the rule, while the other two cases, are captured in the second part of the rule. Note that a condition for the second part of the rule to be applicable is that all positive predicates in the content of the current node have been expanded and there is no possibility for a new positive predicate to be inserted in $\text{CT}(x)$. If this condition is met, an iteration over all possible groundings of rule r is triggered. For every possible grounding it is first checked whether the inequality constraints are not violated, and if this is not the case, one of the resulting literals from the non-local part of the rule (γ-s or δ-s) is refuted.

4.1.3 (iii) Choose a unary predicate.

If for all $a \in \text{CT}(x)$, $\text{ST}(x, a) = exp$ or a is free, and for all $(x, y) \in A_{EF}$, and for all $\pm f \in \text{CT}(x, y)$ (both positive and negative predicates) $\text{ST}((x, y), \pm f) = exp$ or f is free, and there is a $p \in upreds(P)$ such that $p \notin \text{CT}(x)$ and $not\ p \notin \text{CT}(x)$, then add p to $\text{CT}(x)$ with $\text{ST}(x, p) = unexp$ or add $not\ p$ to $\text{CT}(x)$ with $\text{ST}(x, not\ p, r) = unexp$, for every rule $r \in P_p$.

This rule says that in case there is a node x for which all the positive predicate symbols in its content and all the predicate symbols in the contents of its outgoing arcs are free or have already been expanded and there are still unary predicate symbols which do not appear in the content of the current node, one has to pick such a unary predicate symbol p and to inject either p or $not\ p$ in $\text{CT}(x)$. This is needed for consistency reasons: it is not enough to find a justification for the predicate we want to prove that is satisfiable, but one has to show also that this justification makes part from an actual model, which is done by actually constructing such a model. We do not impose that all negative predicate symbols are expanded as that would constrain all the ensuing literals to be locally refuted (the second part for the expand unary negative rule has as precondition the fact that all predicate symbols appear in the content of the current node - see above).

4.1.4 (iv) Expand binary positive.

For a binary positive predicate symbol (non-free) p in $\text{CT}(x, y)$ such that $\text{ST}((x, y), p) = unexp$ nondeterministically choose a rule $r \in P_p$ of the form (3) such that x unifies with s and y unifies with t (s and t are the terms from the head of the rule) to motivate p. For β, γ, and δ corresponding to r do: $update(p(x, y), \beta, x)$, $update(p(x, y), \gamma, (x, y))$, and $update(p(x, y), \delta, y)$. Finally, set $\text{ST}((x, y), p) = exp$.

4.1.5 (v) Expand binary negative.

For a binary negative predicate symbol (non-free) $not\ p$ in $\text{CT}(x, y)$ such that $\text{ST}((x, y), not\ p) = unexp$, and for every rule $r \in P_p$ of the form (3) such that x unifies with s and y unifies with t (s and t are the terms from the head of the rule) do one of the following:

- nondeterministically choose a $\pm q$ from β and $update(not\ p(x, y), \mp q, x)$, or
- nondeterministically choose a $\pm f$ from γ and $update(not\ p(x, y), \mp f, (x, y))$, or
- nondeterministically choose a $\pm q$ from δ and $update(not\ p(x, y), \mp q, y))$.

Finally, set $\text{ST}((x,y), not\ p) = exp$. Note that a binary rule is always local in the sense that a binary literal $\pm f(x,y)$ can always be justified using a component from x, y, and/or (x,y).

4.1.6 (vi) Choose a binary predicate.

There is an $x \in N_{EF}$ for which none of $a \in \text{CT}(x)$ can be expanded with rules (i-ii), and for all $(x,y) \in A_{EF}$ none of $\pm f \in \text{CT}(x,y)$ can be expanded with rules (iv-v), and there is an arc $(x,y) \in A_{EF}$ and a $p \in bpreds(P)$ such that $p \notin \text{CT}(x,y) \wedge not\ p \notin \text{CT}(x,y)$. Then, add p to $\text{CT}(x,y)$ with $\text{ST}((x,y),p) = unexp$ or add $not\ p$ to $\text{CT}(x,y)$ with $\text{ST}((x,y), not\ p) = unexp$.

4.2 Applicability Rules

A second set of rules is not updating the completion structure under consideration, but restricts the use of the expansion rules. We refer to these rules as so-called applicability rules.

4.2.1 (vii) Saturation

We call a node $x \in N_{EF}$ *saturated* if

- for all $p \in upreds(P)$ we have $p \in \text{CT}(x)$ or $not\ p \in \text{CT}(x)$ and none of $\pm q \in \text{CT}(x)$ can be expanded according to the rules (i-iii) ,
- for all $(x,y) \in A_{T^{EF}}$, $T \in EF$ and $p \in bpreds(()P)$, $p \in \text{CT}(x,y)$ or $not\ p \in \text{CT}(x,y)$ and none of $\pm f \in \text{CT}(x,y)$ can be expanded according to the rules (iv-vi).

We impose that no expansions can be performed on a node from N_{EF} which does not belong to $cts(()P)$ until its predecessors are saturated (we exclude constants as they can have more then one predecessor in the completion, including themselves).

4.2.2 (viii) Blocking

A node $x \in N_{EF}$ is *blocked* if there is an ancestor y of x in F, $y <_F x$, $y \notin cts(P)$, s.t. $\text{CT}(x) \subseteq \text{CT}(y)$ and the set $paths_G(y,x) = \{(p,q) \mid (p(y), q(x)) \in connected_G\}$ is empty. We call (y,x) a *blocking pair* and update bl: $bl = bl \cup \{(y,x)\}$. No expansions can be performed on a blocked node. Intuitively, if there is an ancestor y of x which is not a constant, whose content includes the content of x, and there are no paths in G from a positive literal $p(y)$ to another positive literal $q(x)$ one could reuse the justification for y when dealing with x.

4.2.3 (ix) Redundancy

A node $x \in N_{EF}$ is *redundant* if it is saturated, it is not blocked, and there are k ancestors of x in F, $(y_i)_{1 \le i \le k}$, where $k = 2^p(2^{p^2} - 1) + 2$, and $p = |upreds(P)|$, such that $\text{CT}(x) = \text{CT}(y_i)$. In other words, a node is redundant if there are other k nodes on the same branch with the current node which all have content equal to the content of the current node. The presence of a redundant node stops the expansion process. In the completeness proof we show that any forest model of a FoLP P which satisfies p can be reduced to another forest model which satisfies p and has at most $k+1$ nodes with equal content on any branch of a tree from the forest model, and furthermore the $(k+1)st$ node, in case it exists, is blocked. One can thus

search for forest models only of the latter type. This rule exploits that result: we discard models which are not in this shrunk search space.

4.3 Termination, Soundness, and Completion

We call a completion structure *contradictory*, if for some $x \in N_{EF}$ and $a \in upreds(P)$, $\{a, not\ a\} \subseteq \text{CT}(x)$ or for some $(x, y) \in A_{EF}$ and $f \in bpreds(P)$, $\{f, not\ f\} \subseteq \text{CT}(x, y)$. A *complete completion structure* for a FoLP P and a $p \in upreds(P)$, is a completion structure that results from applying the expansion rules to the initial completion structure for p and P, taking into account the applicability rules, such that no expansion rules can be further applied. Furthermore, a complete completion structure $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$ is *clash-free* if:

- (1) CS is not contradictory
- (2) EF does not contain redundant nodes
- (2) G does not contain positive cycles.

We show that an initial completion structure for a unary predicate p and a FoLP P can always be expanded to a complete completion structure (*termination*), that, if p is satisfiable w.r.t. P, there is a clash-free complete completion structure (*soundness*), and, finally, that, if there is a clash-free complete completion structure, p is satisfiable w.r.t. P (*completeness*).

Proposition 4.2 (termination) *Let P be a FoLP and $p \in upreds(P)$. Then, one can construct a finite complete completion structure by a finite number of applications of the expansion rules to the initial completion structure for p w.r.t. P, taking into account the applicability rules.*

Proof Sketch. Assume one cannot construct a complete completion structure by a finite number of applications of the expansion rules, taking into account the applicability rules. Clearly, if one has a finite completion structure that is not complete, a finite application of expansion rules would complete it unless successors are introduced. However, one cannot introduce infinitely many successors: every infinite path in the extended forest will eventually contain $|k + 1|$ saturated nodes with equal content, where k is as in the redundancy rule, and thus either a blocked or a redundant node, which is not further expanded. Furthermore, the arity of the trees in the completion structure is bound by the number of predicates in P and the degrees of the rules. □

Proposition 4.3 (soundness) *Let P be a FoLP and $p \in upreds(P)$. If there exists a complete clash-free completion structure for p w.r.t. P, then p is satisfiable w.r.t. P.*

Proof. From a clash-free complete completion structure, we will construct an open interpretation, and show that this interpretation is an open answer set of P that satisfies p. Let $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$ be such a clash-free complete completion structure with $EF = \langle F, ES\rangle$ the extended forest and $G = (V, A)$ the corresponding dependency graph.

1. *Construction of open interpretation.*

   We construct a new graph $G_{ext} = (V_{ext}, A_{ext})$ by extending G in the following way: first, we set $V_{ext} = V$ and $A_{ext} = A$, and then for every pair $(x, y) \in bl$ do the following:

- (a) for every p such that $p(x) \in V$, add $p(y)$ to V_{ext}: $V_{ext} = V_{ext} \cup \{p(y)\}$;
- (b) for every f and z such that $f(x,z) \in V$, add $f(y,z)$ to V_{ext}: $V_{ext} = V_{ext} \cup \{f(y,z)\}$;
- (c) for every p, q such that $(p(x), q(x)) \in A_{ext}$, add $(p(y), q(y))$ to A_{ext}: $A_{ext} = A_{ext} \cup \{(p(y), q(y))\}$;
- (d) for every p, q, z such that $(p(x), q(z)) \in A_{ext}$, and $z \neq x$ add $(p(y), q(z))$ to A_{ext}: $A_{ext} = A_{ext} \cup \{(p(y), q(z))\}$;
- (e) for every p, f, z such that $(p(x), f(x,z)) \in A_{ext}$, add $(p(y), f(y,z))$ to A_{ext}: $A_{ext} = A_{ext} \cup \{(p(y), f(y,z))\}$;
- (f) for every f, q, z such that $(f(x,z), q(x)) \in A_{ext}$, add $(f(y,z), q(y))$ to A_{ext}: $A_{ext} = A_{ext} \cup \{(f(y,z), q(y))\}$;
- (g) for every f, q, z such that $(f(x,z), q(z)) \in A_{ext}$, add $(f(y,z), q(z))$ to A_{ext}: $A_{ext} = A_{ext} \cup \{(f(y,z), q(z))\}$;
- (h) for every f, g, z such that $(f(x,z), g(x,z)) \in A_{ext}$, add $(f(y,z), g(y,z))$ to A_{ext}: : $A_{ext} = A_{ext} \cup \{(f(y,z), g(y,z))\}$;

Basically, we replicate the content of the blocking node as the content of the blocked node, and also all the connections from/within the blocking node as connections from/within the blocked node (or, in other words, the content of the blocked node is identical with the content of the blocking node and it is justified in a similar way).

Define the open interpretation (U, M) then as (N_{EF}, V_{ext}), i.e., the universe is the set of nodes in the extended forest, and the interpretation corresponds to the nodes in V_{ext}.

2. *M is a model of P_U^M.* All free rules are trivially satisfied.

   Take a ground unary rule: $r' : a(x) \leftarrow \beta^+(x), (\gamma_m^+(x, y_m), \delta_m^+(y_m))_{1 \leq m \leq k}$ (the rule was grounded using individuals from U) originating from $r : a(s) \leftarrow \beta(s), (\gamma_m(s, t_m), \delta_m(t_m))_{1 \leq m \leq k}, \psi$, with $\beta^-(x) \not\subseteq M$, for all $1 \leq m \leq k$: ${\gamma_m}^-(x, y_m) \not\subseteq M$ and ${\delta_m}^-(y_m) \not\subseteq M$, and for all $t_i \neq t_j \in \psi$: $y_i \neq y_j$. Assume $M \models \beta^+(x) \cup \bigcup_{1 \leq m \leq k} \gamma_m^+(x, y_m) \cup \bigcup_{1 \leq m \leq k} \delta_m^+(y_m)$ (together with the assumptions about the negative part of the rule, this amounts to $M \models \beta(x) \cup \bigcup_{1 \leq m < \leq k} \gamma_m(x, y_m) \cup \bigcup_{1 \leq m \leq k} \delta_m(y_m) \cup \psi$) and $a(x) \notin M$ (the rule is not satisfied).

   Depending on x there are two cases:

   - Assume x is not a blocked node. Then $not\ a \in \text{CT}(x)$, x is saturated, and no expansion rules can be further applied to $not\ a$. This means that for every ground rule derived from a rule $r \in P_a$ for which s (the term in the head) unifies with x and grounded in such a way that s was substituted with x, the *expand unary negative* rule has been applied. Such a rule is r'. The application of the *expand unary negative* rule to $not\ a \in \text{CT}(x)$ and r' leads to one of the following situations:
     - there is a unary predicate symbol $\pm q \in \beta$, such that $\mp q \in \text{CT}(x)$ (the result of $update(not\ a(x), \mp q, x)$), or in other words, $\mp q(x) \in M$. This contradicts with $M \models \beta(x)$.
     - there are two successors of x, y_j and y_l such that $y_j = y_l$ and $t_i \neq t_j \in \psi$. This contradicts the assumption that for all $t_i \neq t_j \in \psi$: $y_i \neq y_j$.
     - for some m $1 \leq m \leq k$, there is a binary/unary predicate symbol $\pm f \in \gamma_m / \pm q \in \delta_m$ such that $\mp f \in \text{CT}(x, y_m) / \mp q \in \text{CT}(y_m)$ (the result of $update(not\ a(x), \mp f, (x, y_m))$ / $update(not\ a(x), \mp q, y_m)$), or in other words, $\mp f(x, y_m) \in M / \mp q(y_m) \in M$. This contradicts with $M \models \gamma_m(x, y_m) / M \models \delta_m(y_m)$.

- In case x is a blocked node, by replacing x with y in r', where y is the corresponding blocking node, one obtains a ground rule r'' which again should not be satisfied because due to the construction of M, $M \models \beta(x) \cup \bigcup_{1 \leq m < \leq k} \gamma_m(x, y_m) \cup \bigcup_{1 \leq m \leq k} \delta_m(y_m) \cup \psi$ implies $M \models \beta(y) \cup \bigcup_{1 \leq m < \leq k} \gamma_m(y, y_m) \cup \bigcup_{1 \leq m \leq k} \delta_m(y_m) \cup \psi$ and $a(x) \notin M$ implies $a(y) \notin M$. Thus, this case is reduced to the case above.

In all cases we get a contradiction, so the original assumption that the rule r' is not satisfied by M was false. Thus, every unary rule is satisfied by M.

The proof for the satisfiability of binary rules is similar.

3. *M is a minimal model of P_U^M.* Before proceeding with the actual proof we introduce a notation and a lemma which will prove useful in the following. Let EF' be the directed graph (N_{EF}, A') which has as nodes all the nodes from EF and as arcs all the arcs of EF plus some 'extra' arcs which point from blocked nodes to successors of corresponding blocking nodes to reflect on the changes made to construct an actual model from a completion structure: $A' = A_{EF} \cup \{(y, z) \mid \exists x \text{ s. t. } (x, y) \in bl \wedge z \in succ_{EF}(x)\}$. Figure **??** gives an example of constructing the graph EF' from an extended forest EF by addition of extra arcs: (x, y) is a blocking pair, $z_1, \ldots, z_n$, and b are the successors of x, so extra arcs from y to each of these successors are added (the dotted arrows). Among the successors of x the one which is on the same path with y is singled out and denoted with z (this will be relevant later).

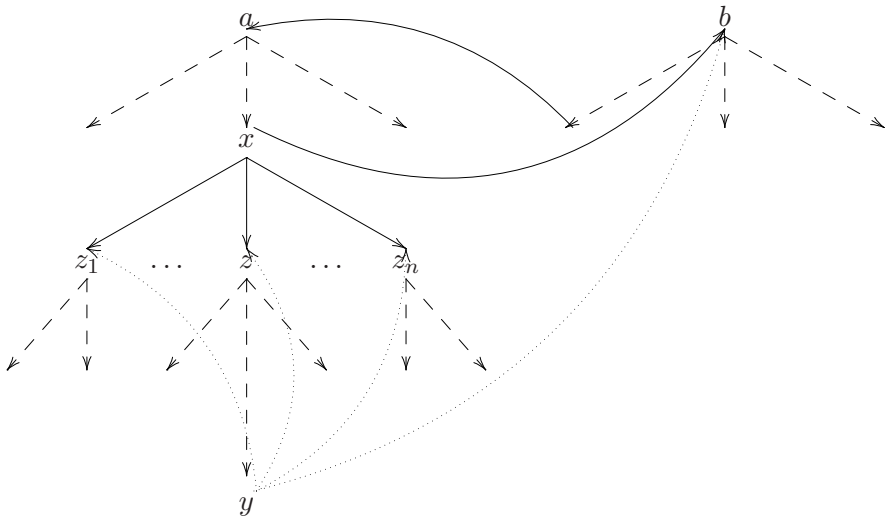

Figure 5: Constructing EF': (x, y) is a blocking pair

Lemma 4.4 *Let Pt be a path from a literal L_1 to a literal L_2 in G/G_{ext}. If $L_1 = p(x)$ for some $p \in upreds(P)$ and $L_2 = q(y)$ for some $q \in upreds(P)$ (q is not necessarily different from p) or $L_2 = g(y, z)$ for some $g \in bpreds(P)$ and $x \neq y$ then there is a path from x to y in EF/EF': $(x_1 = x, x_2, \ldots, x_n = y)$; furthermore, for every $1 \leq i \leq n$ there is a unary literal l_i in Pt with argument x_i and there is a path from l_i to l_{i+1} in G/G_{ext} for every $1 \leq i < n$.*

Proof.

Let $S = (x_1 = x, x_2, \ldots, x_n)$ be a tuple of nodes from $EF/EF^{'}$ constructed in the following way: for every unary literal l in Pt add $arg(l)$ to the tuple if it is not already there (we assume that the elements of Pt are considered in order of their apparition in Pt in the process of this construction). We will show that S is a path in $EF/EF^{'}$ and further more that $x_n = y$.

For every two consecutive elements of the tuple, x_i and x_{i+1}, with $1 \leq i < n$, there must be two unary literals $l^{'}$ and $l^{''}$ in Pt, with arguments x_i and x_{i+1} respectively, such that there is no other unary literal l in the sub-path of Pt: $(l^{'}, \ldots, l^{''})$. It is easy to see that such a sub-path has the form: $(r(x_i), f_1(x_i, x_{i+1}), \ldots, f_m(x_i, x_{i+1}), s(x_{i+1}))$ (this is the only way to reach a unary literal from another in G/G_{ext} without passing through another unary literal), and thus $(x_i, x_{i+1}) \in A/A^{'}$ for every $1 \leq i < n$: $(x_1, \ldots, x_n)$ is a path in $EF/EF^{'}$.

To see that $x_n = y$, consider the opposite: $x_n \neq y$. Then there must be a unary literal $l = r(x_n)$ in Pt with argument x_n such that there is no other unary literal in the sub-path of Pt: $(r(x_n), \ldots, g(y, z))$. This would imply that the sub-path has the form $r(x_n), f_1(x_n, t), \ldots, f_m(x_n, t), g(y, z)$, where t is some successor of x_n in $EF/EF^{'}$: $(x_n, t) \in A/A^{'}$. But there is no arc of the form $(f_m(x_n, t), g(y, z))$ in $A/A^{'}$ with $x_n \neq y$, so we obtain a contradiction.

That there is a path from l_i to l_{i+1} in G/G_{ext} for every $1 \leq i < n$, where l_i-s are the unary literals identified above with $arg(l_i) = x_i$, $1 \leq i \leq n$, is obvious from the way S was constructed.

□

Now we can proceed to the actual proof of statement. Assume there is a model $M' \subset M$ of $Q = P_U^M$. Then $\exists l_1 \in M : l_1 \notin M'$. Take a rule $r_1 \in Q$ of the form $l_1 \leftarrow \beta_1$ with $M \models \beta_1$; note that such a rule always exists by construction of M and expansion rule (i) . If $M' \models \beta_1$, then $M' \models l_1$ (as M' is a model), a contradiction. Thus, $M' \not\models \beta_1$ such that $\exists l_2 \in \beta_1 : l_2 \notin M'$. Continuing with the same line of reasoning, one obtains an infinite sequence $\{l_1, l_2, \ldots\}$ with $(l_i \in M)_{1 \leq i}$ and $(l_i \notin M')_{1 \leq i}$. M is finite (the complete clash-free completion structure has been constructed in a finite number of steps, and when constructing $M(V_{ext})$ we added only a finite number of atoms to the ones already existing in V), thus there must be $1 \leq i, j$, $i \neq j$, such that $l_i = l_j$. We observe that $(l_i, l_{i+1})_{1 \leq i} \in E_{ext}$ by construction of E_{ext} and expansion rule (i), so our assumption leads to the existence of a cycle in G_{ext}.

Claim 4.5 Let $C = (l_1, l_2, \ldots, l_n = l_1)$ be a cycle in G_{ext}. If one of the following holds:

- (i) there is no unary literal in C and for every $l_i = f_i(x_i, y_i)$, $1 \leq i \leq n$, x_i is not blocked
- (ii) there is at least one unary literal in C and for every unary literal in C: l_j, $j \in \{1, 2, \ldots, n\}$, its argument $args(l_j)$ is not a blocked node in CS

then C is a cycle in G.

Proof. From the construction of G_{ext} one can see that any arc which is added to G is of the form $(p(x), l)$ or $(f(x, y), l)$, where p is some unary predicate, f is some binary predicate, and x is a blocked node. It is clear that when condition (i) or condition (ii) holds there is no arc of the first form in C. As concerns arcs of the latter type, it is again obvious that there are no such arcs if condition (i) is fulfilled. In case condition (ii) holds, assume there is an arc $(f(x, y), l)$ where x is a blocking node. We know that there must be at least one unary literal in the cycle. Let this be $p(z)$. In this case

there is a path in G (and also in G_{ext}) from $p(z)$ to $f(x,y)$ and z is different from x by virtue of (ii). According to lemma 4.4 this path contains a unary literal with argument x (as any path in EF from z to x contains x). However this contradicts with condition (ii) which says that there is no such literal in C. □

Claim 4.6 Let $C = (l_1, l_2, \ldots, l_n = l_1)$ be a cycle in G_{ext}. If one of the following holds:

- (i) there is no unary literal in C and for some $l_i = f_i(x_i, y_i)$, $1 \leq i \leq n$, x_i is blocked
- (ii) there is at least one unary literal in C and all unary literals have the same argument y which is a blocked node

then G contains a cycle. *Proof.* We will treat the two cases separately:

(i) First, notice that in this case (when there is no unary literal in the cycle), $args(l_1) = args(l_2) = \ldots = args(l_n) = (x,y)$ as there is no arc in A_{ext} from a binary literal $f(x,y)$ to another binary literal $g(z,t)$, with $x \neq z$ or $y \neq t$ (by construction of G_{ext}). So the cycle can be written as $C = (f_1(x,y), f_2(x,y), \ldots, f_n(x,y) = f_1(x,y))$, where $(f_i \in bpreds(P))_{1 \leq i \leq n}$. Let z be the blocking node corresponding to x: $(z,x) \in bl$. As $((f_i(x,y), f_{i+1}(x,y)) \in A_{ext})_{1 \leq i < n}$, it follows that $((f_i(z,y), f_{i+1}(z,y)) \in A)_{1 \leq i < n}$, so $C' = (f_1(z,y), f_2(z,y), \ldots, f_n(z,y) = f_1(z,y))$ is a cycle in G.

(ii) Let $p_1(y), p_2(y), \ldots, p_n(y)$ be the unary literals in C with y being a blocked node. W.l.o.g. we consider $p_n = p_1$. Then the cycle can be written as: $C = (p_1(y), f_{11}(y,z_1), \ldots, f_{1m_1}(y,z_1), p_2(y), f_{21}(y,z_2), \ldots, f_{2m_2}(y,z_2)), \ldots p_n(y) = p_1(y)$ where $(f_{ij} \in bpreds(P))_{1 \leq i < n, 1 \leq j \leq m_i}$, $((y,z_i) \in A')_{1 \leq i < n}$ (as the only binary literals reachable from $p(y)$ are of the form $f(y,z)$, where $(y,z) \in A'$). Similar with the previous case one can show that $C' = (p_1(x), f_{11}(x,z_1), \ldots, f_{1m_1}(x,z_1), p_2(x), f_{21}(x,z_2), \ldots, f_{2m_2}(x,z_2)), \ldots p_n(x) = p_1(x)$, where x is the corresponding blocking node for y: $(x,y) \in bl$ is a cycle in G. □

Claim 4.7 Let $C = (l_1, l_2, \ldots, l_n = l_1)$ be a cycle in G_{ext}. If there are at least two unary literals in C with different arguments and at least one unary literal has as argument a blocked node y then there is a path in G from a literal l_1 to a literal l_2 where $args(l_1) = x$, $args(l_2) = y$, and x is the corresponding blocking node for y: $(x,y) \in bl$.

Proof. Let t be the argument of a unary literal in the cycle different from y. As there is a path in G_{ext} from some $p(t)$ to some $q(y)$ and also viceversa from some $q(y)$ to some $p(t)$ according to lemma 4.4 there must also be a path in EF' from t to y and a path from y to t. In other words there exists a cycle in EF' which involves both y and t. Furthermore for every element of the cycle in EF', there is a unary literal in C which has this element as an argument. From the way EF' was constructed (see also Figure 5), one can see that any cycle in EF' which involves a blocked node y which makes part from a tree T in the corresponding simple forest contains the path in T from z to y, where z is the node which is a successor of x in T, and is on the same path in T as x and y, x being the corresponding blocking node for y: formally, $(x,y) \in bl$, $z \in succ_T(x)$, $z \in path_T(x,y)$. There are two kinds of cycles in EF':

- cycles which contain x, z, and y (these cycles will contain also elements from other trees than T): in this case there is a unary literal l_1 with argument x in C and there is as well a unary literal l_2 with argument y in C (from the condition of the claim) - so the claim is satisfied
- cycles which contain z, and y, but do not contain x (actually, this is a unique such cycle which has all elements from $path_T(z, y)$): in this case there are two unary literals l_2, and l_3 in C, with arguments y, and z respectively, such that there is no other unary literal on the path induced by C in G_{ext} from l_2 to l_3. In this case this path has the form: $p(y), f_1(y, z), \ldots, f_n(y, z), q(z)$. Due to the construction of G_{ext}, the existence of the path $(p(y), f_1(y, z), \ldots, f_n(y, z), q(z))$ in G_{ext} implies the existence of the path $(p(x), f_1(x, z), \ldots, f_n(x, z), q(z))$ in G. At the same time note that there is a path in G from $q(z)$ to $p(y)$. So, $(p(x), q(z)) \in connected_G$ and $(q(z), p(y)) \in connected_G$, thus $(p(x), p(y)) \in connected_G$ and the claim is satisfied.

□

One can see that the hypotheses of the three claims cover all possible types of cycles C in G_{ext} and that the consequences of having such a cycle are contradicting in each case with the fact that $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$ is a complete clash-free completion structure (in the case of the first two claims, one obtains that there must be a cycle in G, while the conclusion of the third claim contradicts with the blocking condition for a pair of blocking nodes from bl). Thus, there cannot be such a cycle C in G_{ext} and M is minimal.

□

Proposition 4.8 (completeness) *Let P be a FoLP and $p \in upreds(P)$. If p is satisfiable w.r.t. P, then there exists a clash-free complete completion structure for p w.r.t. P.*

Proof. If p is satisfiable w.r.t. P then p is forest-satisfiable w.r.t. P (Proposition 3.3). We construct a clash-free complete completion structure for p w.r.t. P, by guiding the nondeterministic application of the expansion rules with the help of a forest model of P which satisfies p and by taking into account the constraints imposed by the saturation, blocking, redundancy, and clash rules. The proof is inspired by completeness proofs in Description Logics for tableaux, for example in [20], but requires additional mechanisms to eliminate redundant parts from Open Answer Sets.

In order to proceed we need to introduce the notion of *relaxed completion structure* which is a tuple $\langle EF, G, \text{CT}, \text{ST}, bl\rangle$, where EF is an extended forest, and G, CT, st, bl represent the same kind of entities as their homonym counterparts in the definition of a completion structure. An *initial relaxed completion structure for checking satisfiability of a unary predicate p w.r.t. a FoLP P* is defined similarly as an initial completion structure for checking satisfiability of p w.r.t. P. A relaxed completion structure is evolved using the expansion rules (i)-(vi) and the applicability rules (vii)-(viii). Note that the *redundancy* rule is left out. A complete clash-free relaxed completion structure is a relaxed completion structure evolved from an initial relaxed completion structure for p and P, such that no expansion rules can be further applied, which is not contradictory and for which G does not contain positive cycles.

The first step of the proof consists in constructing a complete clash-free relaxed completion structure starting from a forest model of a FoLP P which satisfies p. Note that in the general case, constructing a complete clash-free relaxed completion structure might be a non-terminating process (the termination for the construction of complete clash-free completion structures was based on the application of the redundancy

rule), but as we will see in the following, the process does terminate when a forest model is used as a guidance.

So, let (U, M) be an open answer set of a FoLP P which satisfies p which at the same time is a forest model of P. Then there exists an extended forest $EF = \langle\{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\}, ES\rangle$, where ε is a constant, possibly one of the constants appearing in P, and a labeling function $\mathcal{L} : \{T_\varepsilon\} \cup \{T_a \mid a \in cts(P)\} \cup A_{EF} \to 2^{preds(P)}$ which fulfill the conditions from definition 3.2.

We define an initial relaxed completion structure $CS_0 = \langle EF', G, \text{CT}, \text{ST}, bl\rangle$ for p and P such that $EF' = \langle F', ES'\rangle$, $F' = \{T'_\varepsilon\} \cup \{T'_a \mid a \in cts(P)\}$, where ε is the same ε used to define EF, and $T_x = \{x\}$, for every $x \in cts(P) \cup \{\varepsilon\}$, and $ES' = \emptyset$, $G = \langle V, A\rangle$, $V = \{p(\varepsilon)\}$, $A = \emptyset$, and $\text{CT}(\varepsilon) = \{p\}$, $\text{ST}(\varepsilon, p) = unexp$, $bl = \emptyset$. We will evolve this completion structure using rules (i)-(viii). To this purpose we inductively define a function $\pi : N_{EF'} \to U$ that relates nodes in the relaxed completion structure to nodes in the forest model satisfying the following properties:

$$\ddagger \begin{cases} \{q \mid q \in \text{CT}(z)\} \subseteq \mathcal{L}(\pi(z)), \text{ for all } z \in N_{EF'} \\ \{q \mid not\ q \in \text{CT}(z)\} \cap \mathcal{L}(\pi(z)) = \emptyset, \text{ for all } z \in N_{EF'} \end{cases}$$

Intuitively, the positive content of a node/edge in the completion structure is contained in the label of the corresponding forest model node, and the negative content of a node/edge in the completion structure cannot occur in the label of the corresponding forest model node.

Claim 4.9 Let CS be a relaxed completion structure derived from CS_0 and π a function that satisfies (‡). If an expansion rule is applicable to CS then the rule can be applied such that the resulting relaxed completion structure CS' and an extension π' of π still satisfies (‡).

We start by setting $\pi(x) = x$, for every $x \in cts(P) \cup \{\varepsilon\}$ (the roots of the trees in the relaxed completion structure correspond to the roots of the trees in the forest model). It is clear that (‡) is satisfied for CS_0. By induction let CS be a relaxed completion structure derived from CS_0 and π a function that satisfies (‡). We consider the expansion rules and the applicability rules saturation and blocking:

1. *Expand unary positive.* As $q \in \text{CT}(x)$, we have, by the induction hypothesis, that $q \in \mathcal{L}(\pi(x))$. Since M is a minimal model there is an $r \in P_q$ of the form (2) and a ground version $r' : q(\pi(x)) \leftarrow \beta^+(\pi(x)), (\gamma^+_m(\pi(x), z_m))_{1\le m\le k}, (\delta^+_m(z_m))_{1\le m\le k} \in (P_q)^M_U$ such that $M \models \beta^+(\pi(x)) \cup (\gamma^+_m(\pi(x), z_m))_{1\le m\le k} \cup (\delta^+_m(z_m))_{1\le m\le k}$. Set $\text{RL}(q, x) = r$ and $update(q(x), \beta, x)$. Next, for each $1 \le m \le k$:

   - If $z_m = \pi(z)$ for some z already in EF', take $y_m = z$; also, if $z \in cts(P)$ and $(x, z) \notin ES'$ then $ES' = ES' \cup \{(x, z)\}$,
   - if $z_m = \pi(x) \cdot s$ and z_m is not yet the image of π of some node in EF', then add $x \cdot s$ as a new successor of x in F': $T'_c = T'_c \cup \{x \cdot s\}$, where $x \in T'_c$, set $\pi(x \cdot s) = \pi(x) \cdot s$ and $\pi(x, x \cdot s) = (\pi(x), \pi(x) \cdot s)$.
   - $update(q(x), \gamma_m, (x, y_m))$,
   - $update(q(x), \delta_m, y_m)$.

   In other words we removed the nondeterminism from the *expand unary positive rule*, by choosing the rule r and the successors corresponding to the open answer set (U, M). One can verify that (‡) still holds for π.

2. One can deal with the rules (ii-vi) in a similar way, making the nondeterministic choices in accordance with (U, M).

3. *Saturation*. No expansion rule can be applied on a node from EF' which is not a constant until its predecessor is saturated. This rule is independent of the particular open answer set which guides the construction, so it is applied as usually.

4. *Blocking*. Consider a node $x \in N_{EF'}$ which is selected for expansion. If there is a saturated node $y \in N_{EF'}$ which is not a constant, $y <_{T_c} x$, where $T_c \in F'$, $\text{CT}(x) \subseteq \text{CT}(y)$, and $paths_G(y, x) = \emptyset$ then x is blocked and (y, x) is added to the set of blocking pairs: $bl = bl \cup \{(y, x)\}$. Furthermore, we impose that if there are more nodes y which satisfy the condition we will consider as the blocking node for x the one which is closest to the root of the tree T_c (the tree from which x makes part), so the node y for which there is no node z such that $z <_{T_c} y$, $\text{CT}(x) \subseteq \text{CT}(z)$, and $paths_G(z, y) = \emptyset$. This choice over possible blocking nodes is relevant for the next stage of the proof, where a complete clash-free relaxed completion structure is transformed into a complete clash-free completion structure. The condition (‡) still holds for π as we have not modified the content of nodes, but just removed some unexpanded nodes.

So, (‡) holds for CS' which was evolved from CS, no matter which expansion rule or applicability rule was used. It is easy to see, that if (‡) holds for a particular relaxed completion structure CS then this fact together with the fact that (U, M) is an open answer set of P guarantees that CS is clash-free. So, in order to obtain a complete clash-free relaxed completion structure one has just to apply rules (i-viii) in the manner described above. To see that the process terminates, assume it does not. Then, for every $x, y \in N_{EF'}$ such that $x <'_F y$ and $\text{CT}(x) = \text{CT}(y)$, the blocking rule cannot be applied, so there is a path from a $p(x)$ to some $q(y)$. This suggests the existence of an infinite path in G (as on any infinite branch in a tree from F' there would be an infinite number of nodes with equal content - there is a finite amount of values for the content of a node), which contradicts with the fact that any atom in an open answer set is justified in a finite number of steps[13, Theorem 2].

At this point we have constructed a complete clash-free relaxed completion structure CS for p w.r.t P starting from a forest open answer set for P which satisfies p.

The preference relation over different blocking nodes choices in the construction above has several consequences described by the following results:

Lemma 4.10 *Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$ be a complete clash-free relaxed completion structure constructed in the manner described above ($EF = \langle F, ES\rangle$). Then, for every x such that there exists a y so that $(x, y) \in bl$ (x is a blocking node in CS), there is no node $z <_{T_c} x$, $T_c \in F$ such that $\text{CT}(z) = \text{CT}(x)$.*

Proof. Assume by contradiction that x is a blocking node in CS, so, there is a y such that $(x, y) \in bl$, and that there exists also $z <_{T_c} x$, $T_c \in F$ such that $\text{CT}(z) = \text{CT}(x)$. Observe that $paths_G(z, y) = \{(p(z), q(y)) \mid p \in \text{CT}(z) \wedge q \in \text{CT}(y) \wedge (\exists r \in \text{CT}(x) \text{ s. t. } (p(z), r(x)) \in paths_G(z, x) \wedge (r(x), q(y)) \in paths_G(x, y))\}$ (according to lemma 4.4 the existence of a path from a $p(z)$ to a $q(y)$ in G implies the existence of a path from z to y in EF; all paths from z to y in EF include the path from z to y in T_c and conversely x, and then according to the same lemma there must be a literal in the initial path in G with argument x: $r(x)$ in this case). But $paths_G(x, y) = \emptyset$ as $(x, y) \in bl$, so $paths_G(z, y) = \emptyset$. Additionally, $\text{CT}(z) = \text{CT}(x) \supseteq \text{CT}(y)$, so the existence of z is in contradiction with the preference condition over potentially blocking nodes. Thus, the lemma holds. □

Corollary 4.11 *Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$ be a complete clash-free relaxed completion structure constructed in the manner described above ($EF = \langle E, ES\rangle$) and IB a branch of a tree T_c from F. Then there are at most 2^p distinct blocking nodes in IB where $p = |upreds(P)|$.*

Proof. The result follows from the fact that there cannot be two blocking nodes with equal content on the same path in a tree according to the previous lemma and the finite number of values for the content of a node which is given by the cardinality of the power set of $upreds(P)$. □

The next step is to transform a relaxed clash-free complete completion structure $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$, where $EF = \langle F, ES\rangle$, into a complete clash-free completion structure, that is, a complete clash-free relaxed completion structure which has no redundant nodes. This is done by applying a series of successive transformations on the relaxed completion structure - each transformation "shrinks" the completion structure in the sense that the newer returned relaxed completion structure has a lesser number of nodes than the original one and is still complete and clash-free. The result of applying the transformation is a relaxed clash-free complete completion structure which has a bound on the number of nodes on any branch which matches the bound k from the redundancy condition, which is thus a clash-free complete completion structure. A way to shrink a (relaxed) completion structure is that whenever two nodes u and v in a tree T_c from F are on the same path, $u <_{T_c} v$, and they have equal content, $\text{CT}(u) = \text{CT}(v)$, the subtree $T_c[u]$ is replaced with the subtree $T_c[v]$. We call such a transformation $collapse_{CS}(u, v)$ and its results is a new relaxed completion structure $CS^{'} = \langle EF^{'}, G^{'}, \text{CT}^{'}, \text{ST}^{'}, bl^{'}\rangle$, where the elements of this new completion structure are defined in the following. Let $ef : N_{EF} \rightarrow C$ be a labeled extended forest which associates to every node of EF a label from a set of distinguished constants C such that $ef(x) \neq ef(y)$ for every x and y in N_{EF} such that $x \neq y$. Let $ef^{'} = replace_{ef}(u, v)$ be a new labeled extended forest and $EF^{'}$ be the corresponding unlabeled extended forest. For every $x \in EF^{'}$ let $\overline{x}$ be the counterpart of x in EF in the sense that: $ef^{'}(x) = ef(\overline{x})$. Note that for every $x \in EF^{'}$ there is a unique such counterpart in EF. For simplicity we also introduce the notation $\overline{S}$ to refer to the counterpart tuple (the tuple of counterpart nodes) corresponding to the tuple of nodes from S from $T^{'}$. Formally, $\overline{(x_1, \ldots, x_n)} = (\overline{x_1}, \ldots, \overline{x_n})$. With the help of this notion of counterpart node we will define also the other components of the resulted completion structure ($EF^{'}$ has already been defined):

- $G^{'} = (V^{'}, A^{'})$. The set of nodes $V^{'}$ of the new graph $G^{'}$ contains all literals l for which there is a literal in V formed with the same predicate symbol as l and having as arguments the counterpart of the arguments of l. Additionally, $V^{'}$ contains binary literals which connect the predecessor of u (it is the same both in EF and $EF^{'}$) with the new node u which were also present in V - this is necessarily as $\overline{u} = v$, so otherwise these connections would be lost:

$$\begin{aligned} V^{'} =& \{l_1 \mid \exists l_2 \in V \text{ s. t. } pred(l_1) = pred(l_2) \wedge \overline{arg(l_1)} = arg(l_2)\} \cup \\ & \{f(z, u) \mid z \in T^{'} \wedge f(z, u) \in V\}. \end{aligned}$$

  The set of arcs $A^{'}$ of the new graph $G^{'}$ contains all pair of literals (l_1, l_2) for which there is a corresponding pair in E, (l_3, l_4), such that l_3 and l_4 have the same predicate symbols as l_1 and l_2, respectively, and their argument tuples are the counterpart of the argument tuples of l_1, and l_2, respectively. Additionally, $A^{'}$ contains arcs from A which connect literals whose arguments include the predecessor of u (it is the same both in T and $T^{'}$) with literals whose arguments include the new node u - this

is necessarily as $\overline{u} = v$, so otherwise these connections would be lost:

$$A^{'} = \{(l_1, l_2) \mid \exists (l_3, l_4) \in A \text{ s. t. } pred(l_1) = pred(l_3) \wedge pred(l_2) = pred(l_4) \\ \wedge \overline{arg(l_1)} = arg(l_3) \wedge \overline{arg(l_2)} = arg(l_4)\} \cup \\ \{(l_1, l_2) \mid (l_1, l_2) \in E \wedge u \in arg(l_2) \wedge z \in arg(l_1) \wedge z < u\}.$$

- $\text{CT}^{'}(x) = \text{CT}(\overline{x})$, for every $x \in ef^{'}$;
- $\text{ST}^{'}(x) = \text{ST}(\overline{x})$, for every $x \in ef^{'}$;
- $bl^{'} = \{(x, y) \mid (\overline{x}, \overline{y}) \in bl \wedge paths_{G'}(x, y) = \emptyset\}$. We maintain those blocking pairs whose counterparts in EF formed a blocking pair, and which further more still fulfill the blocking condition.

Note that the result of applying the transformation on a complete clash-free relaxed completion structure might be an incomplete clash-free relaxed completion structure. If completeness of the original structure was achieved by applying among others the blocking rule, the transformation might leave some branches "unfinished" in case the blocking node is eliminated or simply because two nodes who formed a blocking pair are still found in the new structure, but they do not longer fulfill the blocking condition. We will describe two cases in which the transformation can be applied without losing the completeness of the resulted structure by means of two lemmas. Before that, however, we need to state a general result which will prove useful in the demonstration of the two lemmas. The result states that if as a result of applying the $collapse$ transformation on a complete clash-free relaxed completion structure one obtains a completion structure in which the path between a blocking pair of nodes remains untouched (every node in the original path is the counterpart of some node in the new structure), then the nodes which have as counterparts the nodes of the blocking pair form a blocking pair in the new completion structure.

Lemma 4.12 *Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$, $EF = \langle F, ES\rangle$ be a complete clash-free relaxed completion structure and $CS^{'} = \langle EF^{'}, G^{'}, \text{CT}^{'}, \text{ST}^{'}, bl^{'}\rangle$ the result returned by $collapse_C S(u, v)$, where u and v are two nodes from EF which fulfill the usual conditions necessary for the application of $collapse$. Then, for every $(x, y) \in bl$: if for every $z \in path_{T_c}(x, y)$ $(x, y \in T_c)$, exists $z^{'} \in EF^{'}$ such that $\overline{z^{'}} = z$, then $(x^{'}, y^{'}) \in bl^{'}$, where $x^{'}, y^{'} \in EF^{'}$, $\overline{x^{'}} = x$ and $\overline{y^{'}} = y$.*

Proof. Let EF, $EF^{'}$, x, y, $x^{'}$, and $y^{'}$ be as defined in the lemma. The conditions for the two nodes $x^{'}$ and $y^{'}$ from $EF^{'}$ to form a blocking pair: $(x^{'}, y^{'}) \in bl^{'}$, are that $(\overline{x}, \overline{y}) \in bl$ and $paths_{G'}(x^{'}, y^{'}) = \emptyset$. The first condition is part of the prerequisites of the lemma, so it remains to be proved that $paths_{G'}(x^{'}, y^{'}) = \emptyset$. Assume by contradiction that there exists a path in $G^{'}$ from a $p(x^{'})$ to a $q(y^{'})$. Then according to lemma 4.4 there is a path Pt in $EF^{'}$ from $x^{'}$ to $y^{'}$ such that for every $z \in P$ there exists a unary literal with argument z in the path in $G^{'}$ from $p(x^{'})$ to $q(y^{'})$. Any path in $EF^{'}$ from $x^{'}$ to $y^{'}$ includes the path in $T_c^{'}$ (the tree from which both $x^{'}$ to $y^{'}$ make part) from $x^{'}$ to $y^{'}$. Assume $path_{T_c^{'}}(x^{'}, y^{'}) = (x_1^{'} = x^{'}, x_2^{'}, \ldots, x_n^{'} = y^{'})$: then Pt contains the unary literals $l_1{}^{'}, l_2{}^{'}, \ldots, l_n{}^{'}$ with $arg(l_i{}^{'}) = x_i^{'}$, for $1 \leq i \leq n$ such that $(l_i{}^{'}, l_{i+1}^{'}) \in paths_{G'}$, for every $1 \leq i < n$. Let $\overline{x_i^{'}} = x_i$. As every node on the path $path_{T_c}(x, y)$ is the counterpart of some node in $path_{T_c^{'}}(x^{'}, y^{'})$ and every node in $path_{T_c^{'}}(x^{'}, y^{'})$ has the some counterpart in $path_{T_c}(x, y)$, one can conclude that $path_{T_c}(x, y) = (x_1, x_2, \ldots, x_n)$. Also, from the definition of $collapse$ one can see that the presence of unary literals $l_i{}^{'}$ with $arg(l_i{}^{'}) = x_i^{'}$ in $Pt/G^{'}$ implies the presence of literals l_i with $arg(l_i) = x_i$ and $pred(l_i) = pred(l_i^{'})$ in G, for every $1 \leq i \leq n$. Furthermore $(l_i{}^{'}, l_{i+1}^{'}) \in paths_{G^{'}}$ implies $(l_i, l_{i+1}) \in$

$paths_G$, for every $1 \leq i < n$. The latter results leads to: $(l_1, l_n) \in paths_G$ with $arg(l_1) = x_1 = \overline{x_1}^{'} = x$ and $arg(l_n) = x_n = \overline{x_n}^{'} = y$, or in other words to $(pred(l_1), pred(l_n)) \in paths_G(x, y)$. This contradicts with the fact that $(x, y) \in bl$, and thus $paths_G(x, y) = \emptyset$. □

Lemma 4.13 *Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$, $EF = \langle F, ES\rangle$ be a complete clash-free relaxed completion structure. If there are two nodes u and v in a tree T_c in F such that $u <_{T_c} v$, $\text{CT}(u) = \text{CT}(v)$, and there is no blocking node x', $x' <_{T_c} v$, $collapse_{CS}(u, v)$ returns a complete clash-free relaxed completion structure.*

Proof. We have to show that $CS^{'} = collapse_{CS}(u, v)$ is complete, that is, no expansion rule further applies to this completion structure. We will consider every leaf node x of $EF^{'}$ and show that no rule can be applied to further expand such a node. There are three possible cases as concerns the counterpart of x in EF, $\overline{x}$ (which at its turn is a leaf node in EF):

- $\overline{x}$ is a blocked node in CS, which does not make part from the tree T_c from which u and v make part. Let T_d be the tree from which $\overline{x}$ makes part: then there is a node $y^{'} \in T_d$ such that $(y^{'}, \overline{x}) \in bl$. No node was eliminated from T_d as a result of the transformation so for every $z \in path_{T_c}(\overline{x}, y')$, exists $z^{'} \in EF^{'}$ such that $\overline{z^{'}} = z$. Thus lemma 4.12 can be applied: $(x, y) \in bl^{'}$, where y is the node in $EF^{'}$ for which $\overline{y} = y^{'}$. So x is a blocked node in CS.

- $\overline{x}$ is a blocked node in CS which makes part from the same tree T_c from which u and v also make part: then there is a node $y^{'} \in T_c$ such that $(y^{'}, \overline{x}) \in bl$. Depending on the location of $y^{'}$ in T_c one can distinguish between the following situations :

  - $y^{'} \not>_{T_c} u$ (Figure 6 a)): in this case $y^{'}$ is on a branch which does not contain u and v (as it is also the case that $y^{'} \not< u$ due to the fact that there is no blocking node x' such that $\varepsilon \leq x' < v$) and it is not eliminated as a result of applying the transformation, so the path from $\overline{x}$ to $y^{'}$ in T_c is preserved as a result of the transformation. Lemma 4.12 can be applied with the result that $(x, y) \in bl$ where y is the node in $EF^{'}$ for which $\overline{y} = y^{'}$
  - $y^{'} \geq_{T_c} u$ and $y^{'} \not\geq v$ (Figure 6 b)): in this case $y^{'}$ is eliminated as a result of applying the transformation, but $\overline{x}$ is also eliminated which contradicts with the existence of x in $CS^{'}$. To see why $\overline{x}$ is also eliminated notice that $y^{'} \not\leq v$ (as again this would contradict with the fact that there is no blocking node x' such that $\varepsilon \leq x' < v$) and $\overline{x} > y^{'}$. This implies that $\overline{x} > u$ and $\overline{x} \not\leq v$ which suggests that $\overline{x}$ is one of the eliminated nodes, too.
  - $y^{'} \geq v$ (Figure 6 c)): in this case $y^{'}$ is not eliminated as a result of applying the transformation, so the path from $\overline{x}$ to $y^{'}$ in T_c is preserved as a result of the transformation. Lemma 4.12 can be applied with the result that $(x, y) \in bl$ where y is the node in $EF^{'}$ for which $\overline{y} = y^{'}$

  So the conclusion of the analysis above is the existence of a node $y \in T^{'}$ such that $(\overline{y}, \overline{x}) \in bl$. As $paths_G(\overline{y}, \overline{x}) = \emptyset$, $paths_{G'}(y, x) = \emptyset$ as the subtree $T[\overline{y}]$ can be found in $T^{'}$ intact in the form of the subtree $T^{'}[y]$: the eliminated nodes were not part of this subtree as, again, there is no blocking node x' in T, such that $\varepsilon \leq x' < v$.

- $\overline{x}$ is not a blocked node in CS; as CS is complete, no expansion rule can be applied to $\overline{x}$ in CS and, by transfer neither to x in $CS^{'}$ (as they are two nodes which have equal contents which are justified in a similar way).

□

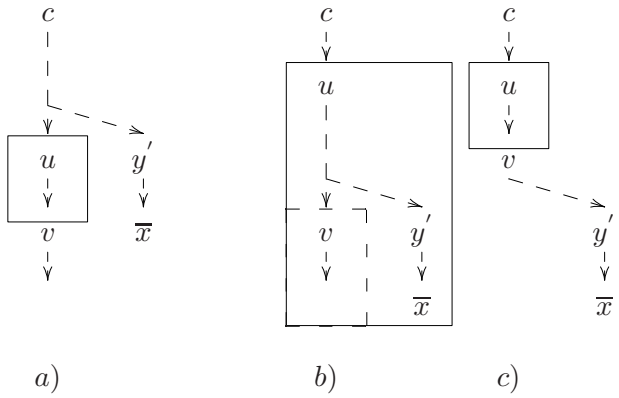

Figure 6: Shrinking a completion structure by eliminating a subtree with a root above any blocking node (the eliminated part is highlighted with continuous line; the part highlighted with dashed line is still kept in)

Lemma 4.14 *Let $CS = \langle T, G, \text{CT}, \text{ST}, bl\rangle$ be a complete clash-free relaxed completion structure. If there are three nodes z, u, and v in T such that $z < u < v$ and there is no blocking node x' such that $z < x' < v$, and $paths_G(z,u) \subseteq paths_G(z,v)$, $collapse_{CS}(u,v)$ returns a complete clash-free relaxed completion structure.*

Proof. Like for the lemma above we show that any leaf node in the completion structure $CS^{'} = collapse_{CS}(u,v)$ (or more precisely in the corresponding tree $T^{'}$) cannot be further expanded. Again we consider every such leaf x and we distinguish between three cases as concerns its counterpart in T, $\overline{x}$:

- $\overline{x}$ is a blocked node in CS, which does not make part from the tree T_c from which u and v make part.This case is similar with the first case in the previous lemma.

- $\overline{x}$ is a blocked node in CS which makes part from the same tree T_c from which u and v make part: then there is a node $y^{'} \in T_c$ such that $(\overline{x}, y^{'}) \in bl$. Using a similar argument as for the previous lemma one concludes that there is a node $y \in T^{'}$ such that $y^{'} = \overline{y}$, or in other words $y^{'}$ has not been eliminated as a result of applying the transformation. In the following we will show that $(y,x) \in bl^{'}$ and x is not further expanded. We will do this on a case-basis considering different locations of $\overline{y}$ and $\overline{x}$ in T_c w.r.t. the nodes z, u, an v (we consider only those cases in which after the transformation both $\overline{y}$ and $\overline{x}$ are maintained in the structure):

  - $\overline{y} \leq_{T_c} z$ and there is a node $z^{'}$ such that $z^{'} <_{T_c} u$, $z^{'} \geq_{T_c} \overline{y}$, and $\overline{x} >_{T_c} z^{'}$ (Figure 7 a)): in this case the transformation does not remove any node from $path_{T_c}(\overline{y}, \overline{x})$ so lemma 4.12 can be applied with the result that $(y,x) \in bl^{'}$.

  - $\overline{y} >_{T_c} v$ (Figure 7 b)): in this case no nodes from the subtree $T_c[\overline{y}]$ are removed during the transformation so using the same argument as above we obtain that $(y,x) \in bl^{'}$.

  - $\overline{y} \not> _{T_c} z$ and $\overline{y} \not\leq_{T_c} z$ (Figure 7 c)): in this case $\overline{y}$ is not on the same path as z, u, and v and again the subtree $T_c[\overline{y}]$ is copied intact into $T_c^{'}$, so $(y,x) \in bl^{'}$.

– $\overline{y} \leq_{T_c} z$ and $\overline{x} \geq_{T_c} v$: in this case $\overline{y}$, z, u, v and $\overline{x}$ are all on the same path in T_c. Assume by contradiction that $paths_{G'}(y,x) \neq \emptyset$, or in other words there is a path in $G^{'}$ from a $p(y)$ to some $q(x)$. By lemma 4.4 one obtains that there must be a path Pt between y and x in EF': note that every such path contains $path_{T_c^{'}}(y,x)$. From the same lemma and the previous observation one obtains that there exists a set of unary literals $l_1, l_2, \ldots, l_n$ in $G^{'}$ with arguments $x_1, x_2, \ldots x_n$, where $path_{T_c^{'}}(y,x) = (x_1 = y, x_2, \ldots x_n = y)$ such that $(l_i, l_{i+1}) \in paths_{G'}$, for $1 \leq i < n$. Note that $(l_i, l_{i+1}) \in paths_{G'}$, for $1 \leq i < n$ implies that $(l_i, l_j) \in paths_{G'}$, for $1 \leq i < j \leq n$. Observe that the counterpart of z from T_c in $T_c^{'}$ is still z and the counterpart of v from T_c in $T_c^{'}$ is u, or in other words $\overline{z} = z$ and $\overline{u} = v$. So, $z, u \in path_{T_c^{'}}(x,y)$, or in other words exists $1 \leq j < k \leq n$ such that $x_j = z$ and $x_k = u$. As $(l_1, l_j), (l_j, l_k), (l_k, l_n) \in paths_{G'}$: $(pred(l_1), pred(l_j)) \in paths_{G'}(y,z)$, $(pred(l_j), pred(l_k)) \in paths_{G'}(z,u)$, and $(pred(l_k), pred(l_n)) \in paths_{G'}(u,x)$. By definition of $collapse$: $paths_{G'}(y,u) = paths_G(\overline{y},u)$, $paths_{G'}(z,u) = paths_G(z,u)$ and $paths_{G'}(u,y) = paths_G(v,\overline{x})$, so: $(pred(l_1), pred(l_j)) \in paths_G(\overline{y},z)$, $(pred(l_j), pred(l_k)) \in paths_G(z,u)$, and $(pred(l_k), pred(l_n)) \in paths_{G'}(v,\overline{x})$. From the lemma condition $paths_G(z,u) \subseteq paths_G(z,v)$, thus $(pred(l_j), pred(l_k)) \in paths_{G'}(z,v)$. Finally, $(pred(l_1), pred(l_j)) \in paths_G(\overline{y},z)$, $(pred(l_j), pred(l_k)) \in paths_G(z,v)$, and $(pred(l_k), pred(l_n)) \in paths_{G'}(v,\overline{x})$ implies $(pred(l_1), pred(l_n)) \in paths_G(\overline{y},\overline{x})$, which is a contradiction with the fact that $paths_G(\overline{y},\overline{x}) = \emptyset$ as $(\overline{y},\overline{x}) \in bl$. Thus, our assumption is false: $paths_{G'}(y,x) = \emptyset$, and $(y,x) \in bl^{'}$.

- $\overline{x}$ is not a blocked node in CS (Figure 7 d)); using a similar argument as for the previous lemma one can show that no expansion rule applies to x in $CS^{'}$.

□

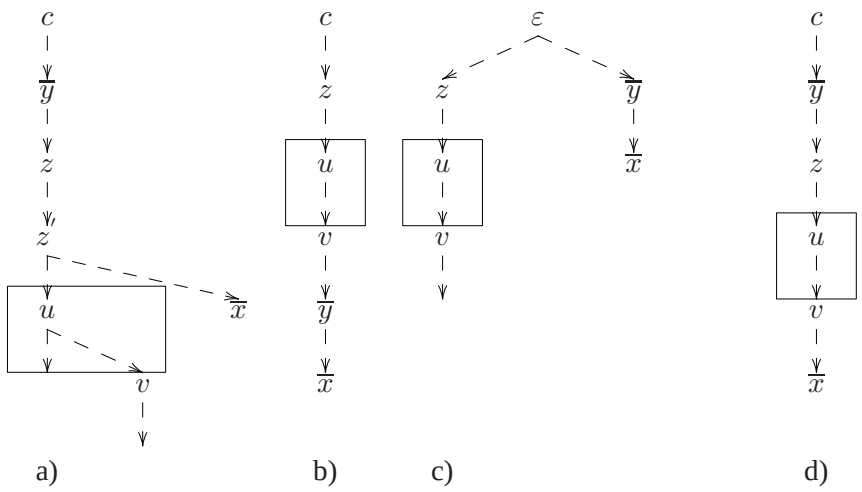

Figure 7: Shrinking a completion structure by eliminating a subtree with a root below a blocking node (the eliminated part is highlighted)

Now, we will describe a sequence of transformations on a relaxed clash-free complete completion structure $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$, $EF = \langle F, ES\rangle$, which returns a complete clash-free completion structure. The transformations which have to be applied to CS are the following (the order in which they are applied is irrelevant):

- for every two nodes u and v in a tree $T_c \in F$ such that $c <_{T_c} u <_{T_c} v$, $\text{CT}(u) = \text{CT}(v)$, and there is no blocking node x, $c \leq_{T_c} x <_{T_c} v$, $collapse_{CS}(u, v)$ (we will call such a transformation a transformation of type 1) ;
- for every two nodes u, and v in a tree $T_c \in F$ for which there exists a node z in T_c such that $z <_{T_c} u <_{T_c} v$ and there is no blocking node x such that $z <_{T_c} x <_{T_c} v$, and $paths_G(z, u) \subseteq paths_G(z, v)$, $collapse_{CS}(u, v)$ (we will call such a transformation a transformation of type 2).

That the resulted completion structure is complete follows directly from Lemma 4.13 and Lemma 4.14. We still have to prove the following claim:

Claim 4.15 Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl\rangle$ be a complete relaxed completion structure to which no transformation of the form described above can be further applied. Then every branch of CS has at most $k = 2^p(2^{p^2} - 1) + 3$ nodes with $p = |upreds(P)|$.

We will analyze every branch of every tree T_c at a time. Consider the current branch is IB and that it contains the blocking nodes $x_1, x_2, \ldots x_n$. From Corollary 4.11 we know that $n \leq 2^p$, where $p = |upreds(P)|$. The last node of the branch will be denoted with end (Figure 8). We split the branch IB in $n + 1$ paths and count the maximum number of nodes with a certain content in each of these paths. In order to do this need an additional lemma which is defined next.

Lemma 4.16 *Let IB be a branch in a tree T_c as depicted in Figure 8. For a given $s \in 2^{upreds(()P)}$:*

- *for any $1 \leq i < n$, there can be at most 2^{p^2} nodes in $path_{T_c}(x_i, x_{i+1})$ with content equal to s, in case there is no node $x \in T_c$ such that $c <_{T_c} x \leq_{T_c} x_i$ and $\text{CT}(x) = s$*
- *for any $1 \leq i < n$, there can be at most $2^{p^2} - 1$ nodes in $path_{T_c}(x_i, x_{i+1})$ with content equal to s, except for x_i, in case there is a node $x \in T_c$ such that $c <_{T_c} x \leq_{T_c} x_i$ and $\text{CT}(x) = s$*
- *there can be at most 2^{p^2} nodes in $path_{T_c}(x_n, end)$ with content equal to s, except for x_n.*

Proof.

We will prove that for any $1 \leq i < n$, there can be at most 2^{p^2} nodes in $path_{T_c}(x_i, x_{i+1})$ with content equal to s in case there is no node $x \in T_c$ such that $c <_{T_c} x \leq_{T_c} x_i$ and $\text{CT}(x) = s$. Assume by contradiction that there are at least $2^{p^2} + 1$ nodes in $path_{T_c}(x_i, x_{i+1})$ with content equal to s. Let's call these node $y_1, y_2, \ldots, y_m$, where $m > 2^{p^2}$. It is necessary that $paths_G(y_1, y_i) \supset paths_G(y_1, y_{i+1})$ for every $1 < i < m$, otherwise a transformation of type 2 could be further applied to CS. As $paths_G(x, y) \subseteq upreds(()P) \times upreds(()P)$ and $|2^{upreds(()P)\times upreds(()P)}| = 2^{p^2}$, and there at least 2^{p^2} distinct values for $paths_G(y_1, y_i)$, when $1 < i < m$, there must be an $1 < i < m$ such that $paths_G(y_1, y_i) = \emptyset$. But in this case $(y_1, y_i) \in bl$ (as the two nodes also have equal content) which contradicts with the fact that $y_i \neq end$. The other cases are proved similarly. □

Now we will proceed to the actual counting. Let $s \in 2^{upreds(()P)}$ be a possible content value for any node in IB. We will count the maximum number of nodes with content s in IB - in order to do this we have to distinguish between three different cases as regards s:

Figure 8: A random branch IB in the resulted complete clash-free relaxed completion structure: $x_1, \ldots, x_n$ are blocking nodes

- there is no node $x \in T_c$ with $c <_{T_c} x <_{T_c} x_1$ such that $\text{CT}(x) = s$, and there is no $1 \leq i \leq n$ such that $\text{CT}(x_i) = s$. In this case there is maximum 1 node with content equal to s in $path_{T_c}(c, x_1)$ (the root), maximum 2^{p^2} nodes in each $path_{T_c}(x_i, x_i + 1)$ and maximum 2^{p^2} nodes in $path_{T_c}(x_n, end)$ (according to lemma 4.16); for the last path there cannot be $2^{p^2} + 1$ nodes as that would mean that end is a blocked node with content equal to s, so there would be a blocking node with content equal to s, which contradicts with the fact the hypothesis there is no blocking node with content equal to s). Also there are at most $2^p - 1$ blocking nodes (if there would be 2^p such nodes, the maximum indicated by corollary 4.11 there would remain no valid value for s). Summing all up, in this case there are at most $2^{p^2}(2^p - 1) + 1$ nodes with content equal to s.

- there is no node x such that $c <_{T_c} x <_{T_c} x_1$ such that $\text{CT}(x) = s$ but there is a node x_i, $1 \leq i \leq n$ such that $\text{CT}(x_i) = s$. In this case there is no node x such that $c <_{T_c} x <_{T_c} x_i$ which has content equal to s (lemma 4.10), and thus $path_{T_c}(c, x_1)$ maximum 1 node with content equal to s (the root). $path_{T_c}(x_i, x_{i+1})$ has maximum 2^{p^2} nodes, every path (x_j, x_{j+1}), where $i < j < n$ has maximum $2^{p^2} - 1$ nodes, and the path (x_n, end) has maximum 2^{p^2} nodes (according to lemma 4.16). Summing all up, in this case there are at most $(2^{p^2} - 1)(n - i + 1) + 3$ nodes with content equal to s, where n is the number of blocking nodes. There are at most 2^p blocking nodes (corollary 4.11), so the maximum of the expression is met when $i = 1$ and $n = 2^p$ and is $2^p(2^{p^2} - 1) + 3$.

- there is a node x such that $c <_{T_c} x <_{T_c} x_1$ and $\text{CT}(x) = s$. In this case $\text{CT}(x_i) \neq s$, for every $1 \leq i \leq n$ (lemma 4.10). The counting is as follows: $path_{T_c}(c, x_1)$ has maximum 1 node with content equal to s (x), otherwise a transformation of type 1 could be applied, $path_{T_c}(x_i, x_{i+1})$ has maximum $2^{p^2} - 1$ nodes, $1 \leq i < n$ and the path (x_n, end) has maximum 2^{p^2} nodes (according to lemma 4.16). Also there are at most $2^p - 1$ blocking nodes (if there would be 2^p such nodes, the maximum indicated by corollary 4.11 there would remain no valid value for s). Summing all up, in this case there are at most $(2^{p^2} - 1)(2^p - 1) + 1$ nodes with content equal to s.

From the three cases the maximum of number of nodes with content equal to a given s in any branch IB of a tree $T_c \in F$ is $2^p(2^{p^2} - 1) + 3$, which is exactly k.

Table 1: Syntax and Semantics of $\mathcal{SHOQ}$ Constructs

construct name	syntax	semantics
atomic concept $\mathbf{C}$	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
nominals $\mathbf{I}$	$\{o\}$	$\{o^{\mathcal{I}}\} \subseteq \Delta^{\mathcal{I}}$,
concept conj.	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
concept disj.	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
exists restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
atleast restriction	$\geq nS.C$	$(\geq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$
atmost restriction	$\leq nS.C$	$(\leq nS.C)^{\mathcal{I}} = \{x \mid \#\{y \mid (x,y) \in S^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$

At this point we have a complete relaxed clash-free completion structure with at most k nodes on any branch, thus a complete clash-free completion structure for p w.r.t. P. □

□

4.4 Complexity Results

Let $CS = \langle EF, G, \text{CT}, \text{ST}, bl \rangle$ be a complete completion structure. Every path of a tree in EF contains at most $k+1$ nodes with equal content (as suggested by the applicability rules (viii) and (ix)), where k is as defined in the redundancy rule; thus, there are at most $(k+1)2^n$ nodes on every such path, where $n = |upreds(P)|$. The branching of every tree is bound by a constant q which is a linear function of the number of variables in unary rules from P. Thus, there are at most $q^{(k+1)2^n}$ nodes in a tree, and at most $(c+1)q^{(k+1)2^n}$ nodes in EF, where c is the number of constants present in the program at hand. So the amount of nodes in CS is double exponential in the size of P, and the algorithm runs in 2-NEXPTIME.

Note that such a high complexity is expected when dealing with tableaux-like algorithms. For example in Description Logics, although satisfiability checking in $\mathcal{SHIQ}$ is EXPTIME-complete, practical algorithms run in 2-NEXPTIME [32].

5 F-hybrid Knowledge Bases

In this section, we introduce *f-hybrid* knowledge bases, a formalism that combines knowledge bases expressed in the Description Logic $\mathcal{SHOQ}$ with forest logic programs.

Description logics (DLs) are a family of logical formalisms based on frame-based systems [24] and useful for knowledge representation. Its basic language features include the notions of *concepts* and *roles* which are used to define the relevant concepts and relations in some (application) domain. Different DLs can then be identified, among others, by the set of constructors that are allowed to form complex concepts or roles; see, for example, the 2 left-most columns of Table 1, that define the constructs in $\mathcal{SHOQ}$ [18].

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 summarize the constructs of $\mathcal{SHOQ}$ with their interpretation in Table 1.

A $\mathcal{SHOQ}$ *knowledge base* is a set of *terminological axioms* $C \sqsubseteq D$ with C and D $\mathcal{SHOQ}$-concept expressions, *role axioms* $R \sqsubseteq S$ with R and S roles, and *transitivity axioms* $\mathsf{Trans}(R)$ for a role name R. If the knowledge base contains an axiom $\mathsf{Trans}(R)$, we call R *transitive*. For the role axioms in a knowledge base, we define $\sqsubseteq^*$ as the transitive closure of $\sqsubseteq$. A *simple role* R in a knowledge base is a role that is not transitive nor does it have any transitive subroles (w.r.t. to reflexive transitive closure $\sqsubseteq^*$ of $\sqsubseteq$). 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., [19]).

We will assume the *unique name assumption* by imposing that $o^{\mathcal{I}} = o$ for individuals $o \in \mathbf{I}$. Note that individuals are thus assumed to be part of any domain $\Delta^{\mathcal{I}}$. Note that OWL does not have the unique name assumption [31], and thus different individuals can point to the same resource. However, the open answer set semantics gives a Herbrand interpretation to constants, i.e., constants are interpreted as themselves, and for consistency we assume that also DL nominals are interpreted this way.

Example 5.1 *Consider the following $\mathcal{SHOQ}$ knowledge base Σ:*

$$\begin{array}{lcl} Father & \sqsubseteq & \exists child.Human \sqcap \neg Female \\ \{john\} & \sqsubseteq & (\leq 2child.Human) \end{array}$$

Intuitively, the first terminological axiom says that fathers have a human child and are not female. The second axiom says that $john$ has less than 2 human children. ∎

Definition 5.2 An *f-hybrid knowledge base* is a pair $\langle \Sigma, P \rangle$ where Σ is a $\mathcal{SHOQ}$ knowledge base and P is a FoLP.

Atoms and literals in P might have as the underlying predicate an atomic concept or role name from Σ, in which case they are called *DL atoms* and *DL literals* respectively. Additionally, there might be other predicate symbols available, but without loss of generality we assume they cannot coincide with complex concept or role descriptions. Note that we do not impose Datalog safeness or *(weakly) DL safeness* [28, 30, 29] for the rule component. Intuitively, the restricted shape of FoLPs suffices to guarantee decidability; FoLPs are in general neither Datalog safe nor weakly DL-safe; we will discuss the relation with weakly DL-safeness in detail in Section 6.

Example 5.3 *An f-hybrid knowledge base $\langle \Sigma, P \rangle$, with Σ as in Example 5.1 and P, the FoLP,*

$$unhappy(X) \leftarrow not\ Father(X)$$

indicates that persons that are not fathers are unhappy, where $Father(X)$ is a DL literal. ∎

Similarly as in [12], we define, given a DL interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and a ground program P, the *projection* $\Pi(P, \mathcal{I})$ of P with respect to $\mathcal{I}$, as follows: for every rule r in P,

- if there exists a DL literal in the head of the form

  - $A(t_1, \ldots, t_n)$ with $(t_1, \ldots, t_n) \in A^{\mathcal{I}}$, or
  - $not\ A(t_1, \ldots, t_n)$ with $(t_1, \ldots, t_n) \notin A^{\mathcal{I}}$,

  then delete r,

- if there exists a DL literal in the body of the form

  - $A(t_1, \ldots, t_n)$ with $(t_1, \ldots, t_n) \notin A^{\mathcal{I}}$, or
  - $not\ A(t_1, \ldots, t_n)$ with $(t_1, \ldots, t_n) \in A^{\mathcal{I}}$,

  then delete r,

- otherwise, delete all DL literals from r.

Intuitively, the projection "evaluates" the program with respect to $\mathcal{I}$ by removing (evaluating) rules and DL literals consistently with $\mathcal{I}$; conceptually this is similar to the GL-reduct, which removes rules and negative literals consistently with an interpretation of the program.

Definition 5.4 Let $\langle \Sigma, P \rangle$ be an f-hybrid knowledge base. An *interpretation* of $\langle \Sigma, P \rangle$ is a tuple $(U, \mathcal{I}, M)$ such that

- U is a universe for P,
- $\mathcal{I} = (U, \cdot^{\mathcal{I}})$ is an interpretation of Σ, and
- M is an interpretation of $\Pi(P_U, \mathcal{I})$.

Then, $(U, \mathcal{I}, M)$ is a *model* of an f-hybrid knowledge base $\langle \Sigma, P \rangle$ if $\mathcal{I}$ is a model of Σ and M is an answer set of $\Pi(P_U, \mathcal{I})$.

The semantics of a f-hybrid knowledge base $\langle \Sigma, P \rangle$ is such that if $\Sigma = \emptyset$, a model of $\langle \Sigma, P \rangle$ corresponds to an open answer set of P, and if $P = \emptyset$, a model of $\langle \Sigma, P \rangle$ corresponds to a DL model of Σ. In this way, the semantics of f-hybrid knowledge bases is nicely layered on top of both the DL semantics and the open answer set semantics.

Example 5.5 *For the f-hybrid knowledge base $\langle \Sigma, P \rangle$ in Example 5.3, take a universe $U = \{john, x\}$ and $\cdot^{\mathcal{I}}$ defined such that $Father^{\mathcal{I}} = \{x\}$, $child^{\mathcal{I}} = \{(x, john)\}$, $Female^{\mathcal{I}} = \emptyset$, $Human^{\mathcal{I}} = U$, and $john^{\mathcal{I}} = john$. It is easy to see that $\mathcal{I} = (U, \cdot^{\mathcal{I}})$ is indeed a model of Σ.*

We project the program P taking into account $\mathcal{I}$, such that P_U is the program

$$\begin{array}{rcl} unhappy(x) & \leftarrow & not\ Father(x) \\ unhappy(john) & \leftarrow & not\ Father(john) \end{array}$$

and since $x \in Father^{\mathcal{I}}$ and $john \notin Father^{\mathcal{I}}$, we have that $\Pi(P_U, \mathcal{I})$ is

$$unhappy(john) \quad \leftarrow$$

such that $M = \{unhappy(john)\}$ is an answer set of $\Pi(P_U, \mathcal{I})$, and $(U, \mathcal{I}, M)$ is a model of $\langle \Sigma, P \rangle$. ∎

For p a concept expression from Σ or a predicate from P, we say that p is *satisfiable* w.r.t. (Σ, P) if there is a model $(U, \mathcal{I}, M)$ such that $p^{\mathcal{I}} \neq \emptyset$ or $p(x_1, \ldots, x_n) \in M$ for some $x_1, \ldots, x_n$ in U, respectively. Note that Definition 5.4 is in general applicable to other DLs than $\mathcal{SHOQ}$ as well as to other programs than FoLPs. Indeed, in [12], a similar definition was used for $\mathcal{DLR}\mathcal{O}^{-\{\leq\}}$ and *guarded programs*.

We can reduce satisfiability checking w.r.t. f-hybrid knowledge bases to satisfiability checking of FoLPs only. Roughly, for each concept expression one introduces a new predicate together with rules that define the semantics of the corresponding DL construct. Constraints then encode the axioms, and the first-order interpretation of DL concept expressions is simulated using free rules.

Taking the knowledge base Σ of Example 5.3, we can translate $Father \sqsubseteq \exists child.Human \sqcap \neg Female$ to the constraint $\leftarrow Father(X), not\ (\exists child.Human \sqcap \neg Female)(X)$ where $(\exists child.Human \sqcap \neg Female)$ is a predicate defined by the rules

$$(\exists child.Human \sqcap \neg Female)(X) \leftarrow (\exists child.Human)(X), (\neg Female)(X)$$

i.e., a DL conjunction translates to a set of literals in the body. Further, we define an exists restriction and negation as follows:

$$\begin{array}{rcl} \exists child.Human(X) & \leftarrow & child(X, Y), Human(Y) \\ \neg Female(X) & \leftarrow & not\ Female(X) \end{array}$$

Finally, the first-order semantics of concepts and roles is obtained as follows:

$$\begin{array}{rcl} Father(X) \vee not\ Father(X) & \leftarrow & \\ Female(X) \vee not\ Female(X) & \leftarrow & \\ Human(X) \vee not\ Human(X) & \leftarrow & \\ child(X, Y) \vee not\ child(X, Y) & \leftarrow & \end{array}$$

Similarly, the axiom $\{john\} \sqsubseteq (\leq 2child.Human)$ is translated as the constraint

$$\leftarrow \{john\}(X), not\ (\leq 2child.Human)(X)$$

and rules

$$\begin{array}{rcl} \{john\}(john) & \leftarrow & \\ (\leq 2child.Human)(X) & \leftarrow & not\ (\geq 3child.Human)(X) \\ (\geq 3child.Human)(X) & \leftarrow & child(X, Y_1), child(X, Y_2), child(X, Y_3), \\ & & Human(Y_1), Human(Y_2), Human(Y_3), Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3 \end{array}$$

Before proceeding with the formal translation, we define the *closure* $clos(\Sigma)$ of a $\mathcal{SHOQ}$ knowledge base Σ as the smallest set satisfying the following conditions:

- for each $C \sqsubseteq D$ an axiom in Σ (role or terminological), $\{C, D\} \subseteq clos(\Sigma)$,
- for each $\mathsf{Trans}(R)$ in Σ, $\{R\} \subseteq clos(\Sigma)$,
- for every D in $clos(\Sigma)$, we have
  - if $D = \neg D_1$, then $\{D_1\} \subseteq clos(\Sigma)$,
  - if $D = D_1 \sqcup D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,

- if $D = D_1 \sqcap D_2$, then $\{D_1, D_2\} \subseteq clos(\Sigma)$,
- if $D = \exists R.D_1$, then $\{R, D_1\} \cup \{\exists S.D_1 \mid S \sqsubseteq^{*} R, S \neq R, \mathsf{Trans}(S) \in \Sigma\} \subseteq clos(\Sigma)$,
- if $D = \forall R.D_1$, then $\{\exists R.\neg D_1\} \subseteq clos(\Sigma)$,
- if $D = (\leq n\ Q.D_1)$, then $\{(\geq n+1\ Q.D_1)\} \subseteq clos(\Sigma)$,
- if $D = (\geq n\ Q.D_1)$, then $\{Q, D_1\} \subseteq clos(\Sigma)$.

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 $y \in D_1^{\mathcal{I}}$, and, in particular, $S \sqsubseteq^{*} R$ with S transitive such that $(x, u_0) \in S^{\mathcal{I}}, \ldots, (u_n, y) \in S^{\mathcal{I}}$ with $y \in D_1^{\mathcal{I}}$. The latter amounts to $x \in (\exists S.D_1)^{\mathcal{I}}$. Thus, in the open answer set setting, we have that $\exists R.D_1(x)$ is in the open answer set if $R(x, y)$ and $D_1(y)$ hold or $\exists S.D_1(x)$ holds for some transitive subrole S of R. The predicate $\exists S.D_1$ will be defined by adding recursive rules, hence the inclusion of such predicates in the closure.

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 $\Phi(\Sigma)$ to be the following FoLP, obtained from the $\mathcal{SHOQ}$ knowledge base Σ:

- For each terminological axiom $C \sqsubseteq D \in \Sigma$, add the constraint

$$\leftarrow C(X), not\ D(X) \tag{5}$$

- For each role axiom $R \sqsubseteq S \in \Sigma$, add the constraint

$$\leftarrow R(X,\, Y), not\ S(X,\, Y) \tag{6}$$

- Next, we distinguish between the types of concept expressions that appear in $clos(\Sigma)$. For each $D \in clos(\Sigma)$:
  - if D is a concept name, add
  $$D(X) \vee not\ D(X) \leftarrow \tag{7}$$
  - if D is a role name, add
  $$D(X,\, Y) \vee not\ D(X,\, Y) \leftarrow \tag{8}$$
  - if $D = \{o\}$, add
  $$D(o) \leftarrow \tag{9}$$
  - if $D = \neg E$, add
  $$D(X) \leftarrow not\ E(X) \tag{10}$$
  - if $D = E \sqcap F$, add
  $$D(X) \leftarrow E(X), F(X) \tag{11}$$
  - if $D = E \sqcup F$, add
  $$\begin{array}{rcl} D(X) & \leftarrow & E(X) \\ D(X) & \leftarrow & F(X) \end{array} \tag{12}$$

– if $D = \exists Q.E$, add

$$D(X) \leftarrow Q(X, Y), E(Y) \tag{13}$$

and for all $S \sqsubseteq^{*} Q$, $S \neq Q$, with $\mathsf{Trans}(S) \in \Sigma$, add rules

$$D(X) \leftarrow (\exists S.E)(X) \tag{14}$$

If $\mathsf{Trans}(Q) \in \Sigma$, we further add the rule

$$D(X) \leftarrow Q(X, Y), D(Y) \tag{15}$$

– if $D = \forall R.E$, add

$$D(X) \leftarrow not\ (\exists R.\neg E)(X) \tag{16}$$

– if $D = (\leq n\ Q.E)$, add

$$D(X) \leftarrow not\ (\geq n+1\ Q.E)(X) \tag{17}$$

– if $D = (\geq n\ Q.E)$, add

$$D(X) \leftarrow Q(X, Y_1), \ldots, Q(X, Y_n), E(Y_1), \ldots, E(Y_n), (Y_i \neq Y_j)_{1 \leq i \neq j \leq n} \tag{18}$$

Rule (13) 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 without violating the FoLP restrictions. We can solve this by adding to (13) the rule (15), such that such a chain $Q(x, y)$, $Q(y, z)$, with $E(z)$ in the open answer set correctly deduces $D(x)$.

It may still be that there are transitive subroles of Q that need the same recursive treatment as above. To this end, we introduce rule (14).

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.

Proposition 5.6 *Let $\langle \Sigma, P \rangle$ be a $\mathcal{SHOQ}$ knowledge base. Then, $\Phi(\Sigma) \cup P$ is a FoLP, and has a size that is polynomial in the size of Σ.*

Proof. Observing the rules in $\Phi(\Sigma)$, it is clear that this program is a FoLP.

The size of the elements in $clos(\Sigma)$ is linear and the size of $clos(\Sigma)$ itself is polynomial in Σ. The size of the FoLP $\Phi(\Sigma)$ is polynomial in the size of $clos(\Sigma)$. The only non-trivial case in showing the latter arises by the addition of rule (18) which introduces $\frac{n(n-1)}{2}$ inequalities for a number restriction $(\geq n\ Q.E)$. We assume, as is not uncommon in DLs (see, e.g., [32]), that the number n is represented in unary notation

$$\underbrace{11\ldots1}_{n}$$

such that the number of introduced inequalities is quadratic in the size of the number restriction. □

Proposition 5.7 *Let $\langle \Sigma, P \rangle$ be an f-hybrid knowledge base. Then, a predicate p is satisfiable w.r.t. (Σ, P) iff p is satisfiable w.r.t. $\Phi(\Sigma) \cup P$.*

Proof. The proof goes along the lines of the proof in [12, Theorem 1].
($\Rightarrow$). Assume p is satisfiable w.r.t. (Σ, P), i.e., there exists a model $(U, \mathcal{I}, M)$ of (Σ, P) in which p has a non-empty extension. Now, we construct the open interpretation (U, N) of $\Phi(\Sigma) \cup P$ as follows:

$$N = M \cup \{C(x) \mid x \in C^{\mathcal{I}}, C \in \mathit{clos}(\Sigma)\} \cup \{R(x_1, x_2) \mid (x_1, x_2) \in R^{\mathcal{I}}, R \in \mathit{clos}(\Sigma)\}$$

with C and R concept expressions and role names respectively.

It is easy to verify that (U, N) is an open answer set of $\Phi(\Sigma) \cup P$ and that (U, N) satisfies p.
($\Leftarrow$). Assume (U, N) is an open answer set of $\Phi(\Sigma) \cup P$ such that p is satisfied. We define the interpretation $(U, \mathcal{I}, M)$ of (Σ, P) as follows:

- $\mathcal{I} = (U, \cdot^{\mathcal{I}})$ is defined such that $A^{\mathcal{I}} = \{x \mid A(x) \in N\}$ for concept names A, $P^{\mathcal{I}} = \{(x_1, x_2) \mid P(x_1, x_2) \in N\}$ for role names P and $o^{\mathcal{I}} = o$, for o a constant symbol in Σ. $\mathcal{I}$ is then an interpretation of Σ.
- $M = N \setminus \{p(x_1, \ldots, x_n) \mid p \in \mathit{clos}(\Sigma)\}$, such that M is an interpretation of $\Pi(P_U, \mathcal{I})$.

As a consequence, $(U, \mathcal{I}, M)$ is an interpretation of $\langle \Sigma, P \rangle$ and it is easy to verify that $(U, \mathcal{I}, M)$ is a model of (Σ, P) which satisfies p. □

Note that Proposition 5.7 also holds for satisfiability checking of concept expressions C: introduce a rule $p(X) \leftarrow C(X)$ in P and check satisfiability of p.

Using the translation from f-hybrid knowledge bases to forest logic programs in Proposition 5.7 and the polynomiality of this translation (Proposition 5.6), together with the complexity of the terminating, sound, and complete algorithm for satisfiability checking w.r.t. FoLPs, we have the following result:

Proposition 5.8 *Satisfiability checking w.r.t. f-hybrid knowledge bases is in* 2-NEXPTIME.

As satisfiability checking of $\mathcal{ALC}$ concepts w.r.t. an $\mathcal{ALC}$ TBox (note that $\mathcal{ALC}$ is a fragment of $\mathcal{SHOQ}$) is EXPTIME-complete ([1, Chapter 3]), we have that satisfiability checking w.r.t. f-hybrid knowledge bases is EXPTIME-hard.

Proposition 5.9 *Satisfiability checking w.r.t. f-hybrid knowledge bases is* EXPTIME*-hard.*

Note that gap between the hardness result and the worst-case runtime complexity of the algorithm. This is a similar gap that arises for $\mathcal{SHIQ}$ and its tableaux algorithm (see Section 4.4).

6 Discussion and Related Work

We compare f-hybrid knowledge bases to the r-hybrid knowledge bases from [30], which extends $\mathcal{DL}$+*log* from [29] with inequalities and negated DL atoms.

In [30], a r-hybrid knowledge base consists of a DL knowledge base (the specific DL is a parameter) and a disjunctive Datalog program where each rule is *weakly DL-safe*:

- every variable in the rule appears in a positive atom in the body of the rule (*Datalog safeness*), and
- every variable either occurs in a positive non-DL atom in the body of the rule, or it only occurs in positive DL atoms in the body of the rule.

The semantics of r-hybrid and f-hybrid knowledge bases correspond to a large extent. The main difference is that f-hybrid knowledge bases do not make the *standard names assumption*, in which basically the domain of every interpretation is the same infinitely countable set of constants.

Some key differences to note are the following:

- We do not require Datalog safeness neither do we require weakly DL-safeness. Indeed, f-hybrid knowledge bases may have a rule component (i.e., the program part) that is not weakly DL-safe. Take the f-hybrid knowledge base $\langle \Sigma, P \rangle$ from Example 5.3 with P:

  $$unhappy(X) \quad \leftarrow \quad not\ Father(X)$$

  The atom $Father(X)$ is a DL-atom such that the rule is neither Datalog safe nor weakly DL-safe. Modifying the program to

  $$unhappy(X) \quad \leftarrow \quad Human(X), not\ Father(X)$$

  leads to a Datalog safe program (X appears in a positive atom $Human(X)$ in the body of the rule), however, it is still not weakly DL-safe as X is not appearing only in positive DL-atoms.

  On the other hand, both the above rules are FoLPs and thus constitute a valid component of an f-hybrid knowledge base.
- In the case of r-hybrid knowledge bases, due to the safeness conditions, it suffices for satisfiability checking to ground the rule component with the constants appearing explicitly in the knowledge base.[4] One does not have such a property for f-hybrid knowledge bases. Consider the f-hybrid knowledge base $\langle \Sigma, P \rangle$ with $\Sigma = \emptyset$ and the program P

  $$\begin{array}{rcl} a(X) & \leftarrow & not\ b(X) \\ b(0) & \leftarrow & \end{array}$$

  This program is a FoLP, but it is not Datalog safe nor is it weakly DL-safe. Grounding only with the constants in the program yields the projection

  $$\begin{array}{rcl} a(0) & \leftarrow & not\ b(0) \\ b(0) & \leftarrow & \end{array}$$

  such that a is not satisfiable. However, grounding with, e.g., $\{0, x\}$, one gets

  $$\begin{array}{rcl} a(0) & \leftarrow & not\ b(0) \\ a(x) & \leftarrow & not\ b(x) \\ b(0) & \leftarrow & \end{array}$$

  such that a is indeed satisfiable, in correspondence with one would expect.
- Decidability for satisfiability checking of r-hybrid knowledge bases is guaranteed if decidability of the conjunctive query containment/union of conjunctive queries containment problems is guaranteed for the DL at hand. In contrast, we relied on a translation of DLs to FoLPs for establishing decidability, and not all DLs can be translated this way; we illustrated the translation for $\mathcal{SHOQ}$.

[4] [30, 29] considers checking satisfiability of knowledge bases rather than satisfiability of predicates. However, the former can easily be reduced to the latter.

Hybrid MKNF knowledge bases [26, 25] consist of a DL component and a component of so-called *MKNF* rules. Such MKNF rules allow for modal operators $\mathbf{K}$ and $\mathbf{not}$ in front of atoms; for a detailed definition, we refer the reader to [26]. Hybrid MKNF knowledge bases generalize approaches to integrating ontologies and rules such as CARIN [23], $\mathcal{AL}$-log [4], DL-safe rules [27], and the Semantic Web Rule Language (SWRL) [17], as well as r-hybrid knowledge bases [30]. In particular, one can write the latter as equisatisfiable hybrid MKNF knowledge bases [26, Theorem 4.8].

In contrast with f-hybrid knowledge bases, hybrid MKNF knowledge bases, as do r-hybrid knowledge bases, make the standard names assumption. Additionally, decidability and reasoning for hybrid MKNF knowledge bases is guaranteed by *DL-safeness*, a restriction of weakly DL-safeness, where every variable in a rule has to appear in a non-DL atom of the body of the rule. As with r-hybrid knowledge bases, our f-hybrid knowledge bases do not have such a restriction of the interaction between the structural DL component and the rule component, but rely instead on the existence of an integrating framework (FoLPs under an open answer set semantics) that has reasoning support; reasoning support that we provided in this paper.

We give a brief overview of other approaches to integrating ontologies and rules, specifically highlighting reasoning support.

Description Logic Programs [11] represent the common subset of OWL-DL ontologies and Horn logic programs (programs without negation as failure or disjunction). As such, reasoning can be reduced to normal LP reasoning. In [27], a clever translation of $\mathcal{SHIQ}(\mathbf{D})$ ($\mathcal{SHIQ}$ with data types) combined with *DL-safe rules* to disjunctive Datalog is provided. The translation relies on a translation to clauses and subsequently applying techniques from basic superposition theory. Reasoning in $\mathcal{DL}{+}log$ [29] and r-hybrid knowledge bases (see above) does not use a translation to other approaches, but defines a specific algorithm based on a partial grounding of the program and a test for containment of conjunctive queries over the DL knowledge bases. *dl-programs* [6] have a more loosely coupled take on integrating DL knowledge bases and logic programs by allowing the program to query the DL knowledge base while as well having the possibility to send (controlled) input to the DL knowledge base. Reasoning is done via a stable model computation of the logic program, interwoven with queries that are oracles to the DL part.

Description Logic Rules [21] are defined as decidable fragments of SWRL. The rules have a tree-like structure similar to the structure of FoLPs. Depending on the underlying DL, one can distinguish between $\mathcal{SROIQ}$ rules (these do not actually extend $\mathcal{SROIQ}$, they are just syntactic sugar on top of the language), $\mathcal{EL}^{++}$ rules, Description Logic Program rules, and ELP rules [22]. The latter can be seen as an extension of both $\mathcal{EL}^{++}$ rules and Description Logic Program rules, hence their name. Although Description Logic Rules have tree-shaped bodies and from this perspective similar to FoLPs, their semantics is not a minimal model semantics. Like Description Logics, there semantics is first-order based.

The algorithm presented in Section 4 can be seen as a procedure that constructs a tableau (as is common in most DL reasoning procedures), representing the possibly infinite open answer set by a finite structure. There are several DL-based approaches which adopt a minimal-style semantics. Among this are autoepistemic [5], default [2] and circumscriptive extensions of DL [3, 10]. The first two extensions are restricted to reasoning with explicitly named individuals only, while [10] allows for defeats to be based on the existence of unknown individuals. A tableau-based method for reasoning with the DL $\mathcal{ALCO}$ in the circumscriptive case has been introduced in [9]. A special preference clash condition is introduced there to distinguish between minimal and non-minimal models which is based on constructing a new classical DL knowledge base and checking its satisfiability.

A formalism related to FoLPs is $\mathbb{FDNC}$ [34]. $\mathbb{FDNC}$ is an extension of ASP with function symbols where rules are syntactically restricted in order to maintain decidability. While the syntactical restriction is

similar to the one imposed on FoLP rules, predicates having arity maximum two, and the terms in a binary literal can be seen as arcs in a forest (imposing the Forest Model Property), the direction of deduction is different: while for FoLPs, all binary literals in a rule body have an identical first term which is also the term which appears in the head, for $\mathbb{FDNC}$ (with the exception of one rule type) the second term is the one which also appears in the head. $\mathbb{FDNC}$ rules are required to be safe unlike FoLP ones. The complexity for standard reasoning tasks for $\mathbb{FDNC}$ is EXPTIME-complete and worst-case optimal algorithms are provided.

7 Conclusions and Outlook

We introduced FoLPs, a logic programming paradigm suitable for integrating ontologies and rules, and provided a sound, complete, and terminating algorithm for satisfiability checking that runs in 2-NEXPTIME. We showed how to use FoLPs as the underlying integration vehicle for reasoning with f-hybrid knowledge bases, a nonmonotonic framework that integrates $\mathcal{SHOQ}$ with FoLPs, without having to resort to (weakly) DL-safeness.

For the future, we intend to look into an extension of f-hybrid knowledge bases and its reasoning algorithm, from $\mathcal{SHOQ}$ towards $\mathcal{SROIQ}(\mathbf{D})$ the DL underlying OWL-DL in OWL 2[5]. A prototype implementation of the algorithm is planned, and will feed the need for optimization strategies.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The DL Handbook: Theory, Implementation, and Applications*, 2003.

[2] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *J. of Automated Reasoning*, 14(2):149–180, 1995.

[3] P. Bonatti, C. Lutz, and F. Wolter. Expressive non-monotonic description logics based on circumscription. In *Proc. of 10th Int. Conf. on Principles of Knowledge Repr. and Reasoning (KR'06)*, pages 400–410, 2006.

[4] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intelligent and Cooperative Information Systems*, 10:227–252, 1998.

[5] F. M. Donini, D. Nardia, and R.Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. on Comput. Logic*, 3(2):177–225, 2002.

[6] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[7] C. Feier and S. Heymans. A sound and complete algorithm for simple conceptual logic programs. In *Proc. of ALPSWS 2008*, 2008.

[8] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080, 1988.

[5] `http://www.w3.org/2007/OWL`

[9] S. Grimm and P. Hitzler. Reasoning in circumscriptive $\mathcal{ALCO}$. Technical report, FZI at University of Karlsruhe, Germany, September 2007.

[10] S. Grimm and P. Hitzler. Defeasible inference with circumscriptive OWL ontologies. In *Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, 2008.

[11] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *Proc. of the World Wide Web Conference (WWW)*, pages 48–57, 2003.

[12] S. Heymans, J. de Bruijn, L. Predoiu, C. Feier, and D. Van Nieuwenborgh. Guarded hybrid knowledge bases. *TPLP*, 8(3):411–429, 2008.

[13] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Conceptual logic programs. *Annals of Mathematics and Artificial Intelligence (Special Issue on Answer Set Programming)*, 47(1–2):103–137, 2006.

[14] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming for the semantic web. *J. of Applied Logic*, 5(1):144–169, 2007.

[15] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming for the semantic web. *J. of Applied Logic*, 5(1):144–169, 2007.

[16] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open answer set programming with guarded programs. *ACM Trans. on Comp. Logic*, 9(4), October 2008.

[17] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the World Wide Web Conference (WWW)*, pages 723–731. ACM, 2004.

[18] I. Horrocks and U. Sattler. Ontology reasoning in the shoq(d) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence*, 2001.

[19] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in LNCS, pages 161–180. Springer, 1999.

[20] U. S. I. Horrocks and S. Tobies. Practical reasoning for expressive description logics. In *Proc. 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, volume LNAI 1705, pages 161–180. Springer Verlag, 1999.

[21] M. Krötzsch, S. Rudolph, and P. Hitzler. Description logic rules. In *Proc. 18th European Conf. on Artificial Intelligence(ECAI-08)*, pages 80–84, 2008.

[22] M. Krötzsch, S. Rudolph, and P. Hitzler. ELP: Tractable rules for OWL 2. In *Proc. 7th Int. Semantic Web Conf. (ISWC-08)*, 2008.

[23] A. Y. Levy and M. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *Proc. of ECAI'96*, pages 323–327, 1996.

[24] M. Minsky. A Framework for Representing Knowledge. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 245–262. Kaufmann, Los Altos, CA, 1985.

[25] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *Proc. of the Int. Semantic Web Conf. (ISWC)*, pages 501–514, 2006.

[26] B. Motik and R. Rosati. Closing Semantic Web Ontologies. Technical report, 2006.

[27] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[28] R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Web Semantics*, 3(1):41–60, 2005.

[29] R. Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 68–78, 2006.

[30] R. Rosati. On combining description logic ontologies and nonrecursive datalog rules. In *Proc. of the 2nd Int. Conf. on Web Reasoning and Rule Systems (RR 2008)*, 2008.

[31] M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2004.

[32] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH-Aachen, 2001.

[33] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th Int. Colloquium on Automata, Languages and Programming*, pages 628–641, 1998.

[34] M. Šimkus and T. Eiter. $\mathbb{FDNC}$: Decidable non-monotonic disjunctive logic programs with function symbols. In *Proc. 14th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2007)*, 2007.