Restricted Unification in the DL $\mathcal{FL}_0$

Franz Baader[1][0000−0002−4049−221X], Oliver Fernández Gil[1][0000−0002−9458−1701], and Maryam Rostamigiv[2][0000−0002−1130−7097]

[1] Theoretical Computer Science, TU Dresden, Dresden, Germany
franz.baader@tu-dresden.de, oliver.fernandez@tu-dresden.de
[2] Département d'Informatique, Paul Sabatier University, Toulouse, France
Maryam.Rostamigiv@irit.fr

Abstract. Unification in the Description Logic (DL) $\mathcal{FL}_0$ is known to be ExpTime-complete and of unification type zero. We investigate whether a lower complexity of the unification problem can be achieved by either syntactically restricting the role depth of concepts or semantically restricting the length of role paths in interpretations. We show that the answer to this question depends on whether the number formulating such a restriction is encoded in unary or binary: for unary coding, the complexity drops from ExpTime to PSpace. As an auxiliary result, we prove a PSpace-completeness result for a depth-restricted version of the intersection emptiness problem for deterministic root-to-frontier tree automata. Finally, we show that the unification type of $\mathcal{FL}_0$ improves from type zero to unitary (finitary) for unification without (with) constants in the restricted setting.

1 Introduction

Unification of concept patterns has been proposed as an inference service in Description Logics that can, for example, be used to detect redundancies in ontologies. For the DL $\mathcal{FL}_0$, which has the concept constructors conjunction ($\sqcap$), value restriction ($\forall r.C$), and top concept ($\top$), unification was investigated in detail in [6]. It was shown there that unification in $\mathcal{FL}_0$ corresponds to unification modulo the equational theory *ACUIh* since (modulo equivalence) conjunction is associative (A), commutative (C), idempotent (I) and has top as a unit (U), and value restrictions behave like homomorphisms for conjunction and top (h). For this equational theory, it had already been shown in [1] that it has unification type zero, which means that a solvable unification problem need not have a minimal complete set of unifiers, and thus in particular not a finite one. From the DL point of view, the decision problem is, however, more interesting than the unification type. Since *ACUIh* is a commutative/monoidal theory [1,14], solvability of *ACUIh* unification problems (and thus of unification problems in $\mathcal{FL}_0$) can be reduced to solvability of systems of linear equations in a certain semiring, which for the case of *ACUIh* consists of finite languages over a finite alphabet, with union as semiring addition and concatenation as semiring multiplication [6]. By a reduction to the emptiness problem for root-to-frontier tree automata (RFAs), it was then shown in [6] that solvability of the language equations corresponding to an $\mathcal{FL}_0$ unification problem can be decided in exponential time. In addition, ExpTime-hardness of this problem was proved in [6] by reduction from the intersection emptiness problem for deterministic RFAs (DRFAs) [16].

In the present paper, we investigate two kinds of restrictions on unification in $\mathcal{FL}_0$. On the one hand, we *syntactically restrict the role depth* (i.e., the maximal nesting of value restrictions) in the concepts obtained by applying a unifier to be bounded by a natural number $k \geq 1$. This restriction was motivated by a similar restriction used in research on least common subsumers (lcs) [15], where imposing a bound on the role depth guarantees existence of the lcs also in the presence of a (possibly cyclic) terminology. Also note that such a restriction was used in [11] for the theory ACh, for which unification is known to be undecidable [13]. It is shown in [11] that the problem becomes decidable if a bound on the maximal nesting of applications of homomorphisms is imposed. On the other hand, we consider a *semantic restriction* where only interpretations for which the length of role paths is bounded by a given number k are considered when defining the semantics of concepts. A similar restriction (for $k = 1$) was employed in [9] to improve the unification type for the modal logic $\mathbf{K}$ from type zero [10] to unitary or finitary for $\mathbf{K} + \Box\Box\bot$.

In the present paper we show that both the syntactic and the semantic restriction ensures that the unification type of $\mathcal{FL}_0$ (and equivalently, of the theory $ACUIh$) improves from type zero to unitary for unification without constants and finitary for unification with constants. Regarding the decision problem, we can show that the complexity depends on whether the bound k is assumed to be encoded in unary or binary.[3] For binary encoding of k, the complexity stays ExpTime, whereas for unary coding it drops from ExpTime to PSpace. This is again the case both for the syntactic and the semantic restriction. As an auxiliary result we prove that a depth-restricted variant of the intersection emptiness for DRFAs is PSpace-complete.

Showing these results requires combining methods and results from knowledge representation, unification theory, and automata theory. Due to space restrictions, we cannot give detailed proofs here. They can be found in [3].

2 The DL $\mathcal{FL}_0$ and Restrictions

Starting with mutually disjoint countably infinite sets N_C and N_R of concept and role names, respectively, the set of $\mathcal{FL}_0$ concepts is inductively defined as follows:

- $\top$ (top concept) and every concept name $A \in N_C$ is an $\mathcal{FL}_0$ concept,
- if C, D are $\mathcal{FL}_0$ concepts and $r \in N_R$ is a role name, then $C \sqcap D$ (conjunction) and $\forall r.C$ (value restriction) are $\mathcal{FL}_0$ concepts.

[3] For unary coding, the size of the input k is the number k, whereas for binary coding it is the size of its binary encoding, i.e., $\log k$.

The *semantics* of $\mathcal{FL}_0$ concepts is defined using first-order interpretations $\mathcal{I} = (\mathcal{D}^\mathcal{I}, \cdot^\mathcal{I})$ consisting of a non-empty domain $\mathcal{D}^\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns a set $A^\mathcal{I} \subseteq \mathcal{D}^\mathcal{I}$ to each concept name A, and a binary relation $r^\mathcal{I} \subseteq \mathcal{D}^\mathcal{I} \times \mathcal{D}^\mathcal{I}$ to each role name r. This function is extended to $\mathcal{FL}_0$ concepts as follows:

$$\top^\mathcal{I} = \mathcal{D}^\mathcal{I} \text{ and } (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I},$$
$$(\forall r.C)^\mathcal{I} = \{x \in \mathcal{D}^\mathcal{I} \mid \forall y \in \mathcal{D}^\mathcal{I}\colon (x, y) \in r^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\}.$$

Given two $\mathcal{FL}_0$ concepts C and D, we say that C is subsumed by D (written $C \sqsubseteq D$) if $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$, and that C is equivalent to D (written $C \equiv D$) if $C \sqsubseteq D$ and $D \sqsubseteq C$. It is well known that subsumption (and thus also equivalence) of $\mathcal{FL}_0$ concepts can be decided in polynomial time [12].

Note that, up to equivalence, conjunction is associative, commutative, and idempotent, and has the unit element $\top$. In addition, the following equivalences hold for value restrictions: $\forall r.\top \equiv \top$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$. Due to these equivalences, one can transform $\mathcal{FL}_0$ concepts into a normal form that uses formal languages over the alphabet of role names to represent value restrictions that end with the same concept name. In fact, using these equivalences as rewrite rules from left to right, every $\mathcal{FL}_0$ concept can be transformed into an equivalent one that is either $\top$ or a conjunction of concepts of the form $\forall r_1. \cdots \forall r_n.A$, where $r_1, \ldots, r_n$ are role names and A is a concept name. Such a concept can be abbreviated as $\forall w.A$, where $w = r_1 \ldots r_n$ is a word over the alphabet N_R. Note that $n = 0$ means that w is the empty word ε, and thus $\forall \varepsilon.A$ corresponds to A. Furthermore, a conjunction of the form $\forall w_1.A \sqcap \ldots \sqcap \forall w_m.A$ can be written as $\forall L.A$ where $L \subseteq N_R^*$ is the finite language $\{w_1, \ldots, w_m\}$. We use the convention that $\forall \emptyset.A$ corresponds to the top concept $\top$. Thus, any two $\mathcal{FL}_0$ concepts C, D containing only the concept names $A_1, \ldots, A_\ell$ can be represented as

$$C \equiv \forall K_1.A_1 \sqcap \ldots \sqcap \forall K_\ell.A_\ell \text{ and } D \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_\ell.A_\ell, \qquad (1)$$

where $K_1, L_1, \ldots, K_\ell, L_\ell$ are finite languages over the alphabet of role names N_R. We call this representation the *language normal form (LNF)* of C, D. If C, D have the LNF shown in (1), then $C \equiv D$ holds iff $L_1 = K_1, \ldots, L_\ell = K_\ell$ (see Lemma 4.2 of [6]).

2.1 Syntactically Restricting the Role Depth

The role depth of an $\mathcal{FL}_0$ concept is the maximal nesting of value restrictions in this concept. Since occurrences of $\top$ within value restrictions can increase the role depth artificially, we assume without loss of generality that $\mathcal{FL}_0$ concepts different from $\top$ do not contain any occurrences of $\top$. We will make this assumption in the rest of the paper without mentioning it explicitly.

The role depth $rd(C)$ of an $\mathcal{FL}_0$ concept C is defined by induction:

- $rd(\top) = rd(A) = 0$ for all $A \in N_C$,
- $rd(C \sqcap D) = \max(rd(C), rd(D))$ and $rd(\forall r.C) = 1 + rd(C)$.

It is easy to see that (under the above assumption) the role depth of $\mathcal{FL}_0$ concepts is preserved under equivalence.

We are now ready to define our first restricted version of subsumption and equivalence in $\mathcal{FL}_0$. For an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C and D, we define subsumption and equivalence restricted to concepts of role depth $\leq k$ as follows:

- $C \sqsubseteq^k_{syn} D$ if $C \sqsubseteq D$ and $\max(rd(C), rd(D)) \leq k$,
- $C \equiv^k_{syn} D$ if $C \sqsubseteq^k_{syn} D$ and $D \sqsubseteq^k_{syn} C$.

The effect of this definition is that subsumption and equivalence can only hold for concepts that satisfy the restriction of the role depth by k. For concepts satisfying this syntactic restriction, the relations $\sqsubseteq^k_{syn}$ and $\equiv^k_{syn}$ coincide with the classical subsumption and equivalence relations on $\mathcal{FL}_0$ concepts. Using the language normal form of $\mathcal{FL}_0$ concepts, the equivalence $\equiv^k_{syn}$ can be characterized as follows: if C, D have the LNF shown in (1), then $C \equiv^k_{syn} D$ iff $L_1 = K_1 \subseteq N_R^{\leq k}, \ldots, L_\ell = K_\ell \subseteq N_R^{\leq k}$, where $N_R^{\leq k}$ denotes the set of words over N_R of length at most k.

2.2 Semantically Restricting the Length of Role Paths

For an integer $n \geq 1$ and a given interpretation $\mathcal{I} = (\mathcal{D}^\mathcal{I}, \cdot^\mathcal{I})$, a role path of length n is a sequence $d_0, r_1, d_1, \ldots, d_{n-1}, r_n, d_n$, where $d_0, \ldots, d_n$ are elements of $\mathcal{D}^\mathcal{I}$, $r_1, \ldots, r_n$ are role names, and $(d_{i-1}, d_i) \in r_i^\mathcal{I}$ holds for all $i = 1, \ldots, n$. The interpretation $\mathcal{I}$ is called k-restricted if it does not contain any role paths of length $> k$.

For an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C and D, we define subsumption and equivalence restricted to interpretations with role paths of length $\leq k$ as follows:

- $C \sqsubseteq^k_{sem} D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all k-restricted interpretations $\mathcal{I}$,
- $C \equiv^k_{sem} D$ if $C \sqsubseteq^k_{sem} D$ and $D \sqsubseteq^k_{sem} C$.

The effect of this notion of equivalence is that all concepts occurring at a role depth $> k$ can be replaced by $\top$. To be more precise, we define the restriction of a concept C to role depth $n \geq 0$ by induction on n as follows:

- $A|_n = A$ for $A \in N_C \cup \{\top\}$ and $(C \sqcap D)|_n = C|_n \sqcap D|_n$ for all $n \geq 0$;
- $(\forall r.C)|_0 = \top$ and $(\forall r.C)|_n = \forall r.(C|_{n-1})$ for all $n \geq 1$.

For example, $(\forall r.\forall r.\forall r.A)|_4 = \forall r.\forall r.\forall r.A = (\forall r.\forall r.\forall r.A)|_3$ and $(\forall r.\forall r.\forall r.A)|_2 = \forall r.\forall r.\top \equiv \top$. In the language normal form, restricting to role depth n means that all words that are longer than n can simply be removed.

It is easy to see that $C \equiv^k_{sem} D$ iff $C|_k \equiv D|_k$, which yields the following characterization of the equivalence $\equiv^k_{sem}$: if C, D have the LNF shown in (1), then $C \equiv^k_{sem} D$ iff $L_1 \cap N_R^{\leq k} = K_1 \cap N_R^{\leq k}, \ldots, L_\ell \cap N_R^{\leq k} = K_\ell \cap N_R^{\leq k}$.

3 Unification in $\mathcal{FL}_0$

In unification, we consider concepts that may contain variables, which can be replaced by concepts. More formally, we introduce a countably infinite set N_V of concept variables, which is disjoint with N_C and N_R. An $\mathcal{FL}_0$ concept pattern is an $\mathcal{FL}_0$ concept that is constructed using $N_C \cup N_V$ as concept names. The semantics of concept patterns is defined as for concepts, i.e., concept variables are treated like concept names when defining the semantics. This way, the notions of subsumption and equivalence (both in the restricted and in the unrestricted setting) transfer from concepts to concept patterns in the obvious way.

A substitution σ is a mapping from N_V into the set of all $\mathcal{FL}_0$ concept patterns such that $dom(\sigma) := \{X \in N_V \mid \sigma(X) \neq X\}$ is finite. This mapping is extended to concept patterns in the obvious ways:

- $\sigma(A) := A$ for all $A \in N_C \cup \{\top\}$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$ and $\sigma(\forall r.C) := \forall r.\sigma(C)$.

An $\mathcal{FL}_0$ *unification problem* is an equation of the form $C \overset{?}{\equiv} D$ where C, D are $\mathcal{FL}_0$ concept patterns. A unifier of this equation is a substitution σ such that $\sigma(C) \equiv \sigma(D)$.

It was shown in [6] that the question of whether a given $\mathcal{FL}_0$ unification problem has a unifier or not can be reduced to solving linear language equations, i.e., equations of the form

$$S_0 \cup S_1{\cdot}X_1 \cup \cdots \cup S_n{\cdot}X_n = T_0 \cup T_1{\cdot}X_1 \cup \cdots \cup T_n{\cdot}X_n, \tag{2}$$

where $S_0, \ldots, S_n, T_0, \ldots, T_n$ are finite languages of words over an alphabet $\Delta = \{1, \ldots, \rho\}$[4] and $X_1, \ldots, X_n$ are variables that can be replaced by finite languages over Δ. A solution of the equation (2) is an assignment θ of finite languages $\theta(X_i)$ to the variables X_i (for $i = 1, \ldots, n$) such that

$$S_0 \cup S_1{\cdot}\theta(X_1) \cup \cdots \cup S_n{\cdot}\theta(X_n) = T_0 \cup T_1{\cdot}\theta(X_1) \cup \cdots \cup T_n{\cdot}\theta(X_n), \tag{3}$$

where $\cup$ is interpreted as union and $\cdot$ as concatenation of languages. Strictly speaking, a given $\mathcal{FL}_0$ unification problem yields one such language equation for every concept name occurring in the problem. But since these equations do not share variables, they can be solved separately. Also note that solvability of language equations of the form (2) can in turn be reduced in polynomial time to $\mathcal{FL}_0$ unification.

A word $w = i_1 \ldots i_\ell$ occurring in a solution of the form (3) corresponds to a conjunct $\forall r_{i_1}.\cdots\forall r_{i_\ell}.A$ in the unified concept $\sigma(C) \equiv \sigma(D)$. Thus, the length of the word w is equal to the role depth of the corresponding sequence of value restrictions.

[4] Intuitively, ρ is the number of different role names occurring in the unification problem and each letter $i, 1 \leq i \leq \rho$, stands for a role name r_i.

Example 1. Consider the $\mathcal{FL}_0$ unification problem $\forall r_1.\forall r_1.A \sqcap \forall r_1.\forall r_1.X \overset{?}{\equiv} X \sqcap \forall r_1.\forall r_1.\forall r_1.Y$. The substitution σ with $\sigma(X) = \forall r_1.\forall r_1.A$ and $\sigma(Y) = \forall r_1.A$ is one of the unifiers of this problem. The language equation induced by this unification problem is $\{11\} \cup \{11\}\cdot X = \{\varepsilon\}\cdot X \cup \{111\}\cdot Y$. The unifier σ corresponds to the following solution θ of this problem: $\theta(X) = \{11\}$ and $\theta(Y) = \{1\}$.

3.1 Syntactically Restricted Unification in $\mathcal{FL}_0$

For an integer $k \geq 1$, a syntactically k-restricted unification problem is an equation of the form $C \overset{?}{\equiv}{}^{k}_{syn} D$, where C, D are $\mathcal{FL}_0$ concept patterns. A unifier of this equation is a substitution σ such that $\sigma(C) \equiv^{k}_{syn} \sigma(D)$.

Due to the LNF characterization of $\equiv^{k}_{syn}$ and the correspondence between role depth and word length mentioned above, solvability of a given syntactically k-restricted unification problem can be reduced to checking whether language equations of the form (2) have solutions θ such that

$$S_0 \cup S_1\cdot\theta(X_1) \cup \cdots \cup S_n\cdot\theta(X_n) = T_0 \cup T_1\cdot\theta(X_1) \cup \cdots \cup T_n\cdot\theta(X_n) \subseteq \Delta^{\leq k}, \quad (4)$$

where $\Delta^{\leq k}$ denotes the set of words over Δ of length at most k.

The unifier σ of the $\mathcal{FL}_0$ unification problem in Example 1 is not a syntactically 3-restricted unifier of this problem since the unified concept $\sigma(\forall r_1.\forall r_1.A \sqcap \forall r_1.\forall r_1.X) = \forall r_1.\forall r_1.A \sqcap \forall r_1.\forall r_1.\forall r_1.\forall r_1.A = \sigma(X \sqcap \forall r_1.\forall r_1.\forall r_1.Y)$ has role depth 4. This is reflected on the language equation side by the fact that $\{11\} \cup \{11\}\cdot\{11\} = \{11, 1111\} = \{\varepsilon\}\cdot\{11\} \cup \{111\}\cdot\{1\} \not\subseteq \Delta^{\leq 3}$. In fact, it is easy to see that this problem does not have a syntactically 3-restricted unifier.

3.2 Semantically Restricted Unification in $\mathcal{FL}_0$

For an integer $k \geq 1$, a semantically k-restricted unification problem is an equation of the form $C \overset{?}{\equiv}{}^{k}_{sem} D$, where C, D are $\mathcal{FL}_0$ concept patterns. A unifier of this equation is a substitution σ such that $\sigma(C) \equiv^{k}_{sem} \sigma(D)$.

Whereas in the syntactically restricted case a sequence of value restrictions of depth $> k$ (a word of length $> k$) destroys the property of being a unifier (solution), in the semantically restricted case one can simply ignore such sequences (words). Thus, one can reduce the question of whether a given semantically k-restricted unification problem has a unifier or not to checking whether, for language equations of the form (2), there is an assignment θ such that

$$\begin{aligned}&(S_0 \cup S_1\cdot\theta(X_1) \cup \cdots \cup S_n\cdot\theta(X_n)) \cap \Delta^{\leq k} = \\ &(T_0 \cup T_1\cdot\theta(X_1) \cup \cdots \cup T_n\cdot\theta(X_n)) \cap \Delta^{\leq k}.\end{aligned} \quad (5)$$

Note that, in general, such an assignment need not be a solution of (2), but clearly any solution θ of (2) satisfying (3) also satisfies (5).

Example 2. The $\mathcal{FL}_0$ unification problem $\forall r_1.A \sqcap \forall r_1.\forall r_1.X \overset{?}{\equiv} X$ induces the language equation $\{1\} \cup \{11\}\cdot X = \{\varepsilon\}\cdot X$. This language equation does not have a solution in the classical sense, but it has a semantically 3-restricted solution. In fact, for the assignment θ with $\theta(X) = \{1, 111\}$ we have $\{1\} \cup \{11\}\cdot\theta(X) = \{1, 111, 11111\}$ and $\{\varepsilon\}\cdot\theta(X) = \{1, 111\}$. Intersecting these two sets with $\Delta^{\leq 3}$ yields the same set $\{1, 111\}$. Thus, the above unification problem does not have a unifier, but it has a semantically 3-restricted unifier.

4 Root-to-Frontier Tree Automata

It was shown in [6] that checking solvability of linear language equations can be reduced to testing emptiness of tree automata. More precisely, the tree automata employed in [6] work on finite node-labelled trees, going from the root to the leaves. Such automata are called root-to-frontier tree automata (RFAs) in [6]. Basically, given a linear language equation, one can construct an RFA whose size is exponential in the size of the language equation, and which accepts some tree iff the language equation has a solution. Since the emptiness problem for RFAs is polynomial, this yields an ExpTime upper bound for solvability of linear language equations. The matching ExpTime lower bound was proved in [6] by reduction from the intersection emptiness problem for deterministic RFAs (DRFAs). In this section, we formally introduce (D)RFAs and the trees they accept, and recall the ExpTime-completeness result for the intersection emptiness problem for DRFAs from [16]. We then state our new result that a depth-restricted version of this problem is PSpace-complete.

We consider trees with labels in a ranked alphabet Σ, where the number of successors of a node is determined by the rank of its label.

Definition 1. *Let Σ be a finite alphabet, where each $f \in \Sigma$ is associated with a rank, denoted as $rank(f)$, such that $rank(f) \geq 0$, and let ρ be the maximal rank of the elements of Σ. A (finite) Σ-tree is a mapping $t : dom(t) \to \Sigma$ such that $dom(t)$ is a finite subset of $\{1, \ldots, \rho\}^*$ such that*

- *the empty word ε belongs to $dom(t)$;*
- *for all $u \in \{1, \ldots, \rho\}^*$ and $i \in \{1, \ldots, \rho\}$, we have $ui \in dom(t)$ iff $u \in dom(t)$ and $i \leq rank(t(u))$.*

The elements of $dom(t)$ are the nodes of the tree t, and $t(u)$ is called the label of node u. The empty word ε is the root of t, and the nodes u such that $ui \notin dom(t)$ for all $i = 1, \ldots, \rho$ are the leaves of t. By the above definition, the leaves are the nodes labeled with a symbol of rank zero, i.e., $rank(t(u)) = 0$ iff u is a leaf of t. We denote the set of symbols of rank 0 by $\Sigma_0 := \{f \in \Sigma \mid rank(f) = 0\}$. We always assume $\Sigma_0 \neq \emptyset$ since otherwise there is no finite Σ-tree. Nodes of t that are not leaves are called inner nodes. The *depth of a node $u \in dom(t)$* is just the length of the word u. The *depth of the tree t*, denoted as $depth(t)$, is the maximal depth of a node in $dom(t)$.

Definition 2. *A (non-deterministic)* root-to-frontier tree automaton (RFA) *that works on Σ-trees is a 5-tuple $\mathcal{A} = (\Sigma, Q, I, T, F)$, where*

- *Σ is a finite, ranked alphabet,*
- *Q is a finite set of states,*
- *$I \subseteq Q$ is the set of initial states,*
- *T assigns to each $f \in \Sigma \setminus \Sigma_0$ of rank n a transition relation $T(f) \subseteq Q \times Q^n$,*
- *$F : \Sigma_0 \to 2^Q$ assigns to each $c \in \Sigma_0$ a set of final states $F(c) \subseteq Q$.*

A run *of $\mathcal{A}$ on the tree t is a mapping $r : dom(t) \to Q$ such that*

- *$(r(u), r(u1), \ldots, r(un)) \in T(t(u))$ for all inner nodes u of rank n.*

The run r is called successful *if*

- *$r(\varepsilon) \in I$ (root condition),*
- *$r(u) \in F(t(u))$ for all leaves u (leaf condition).*

The tree language accepted *by $\mathcal{A}$ is defined as*

$$\mathcal{L}(\mathcal{A}) := \{t \mid \textit{there exists a successful run of } \mathcal{A} \textit{ on } t\}.$$

The emptiness problem *for $\mathcal{A}$ is the question whether $\mathcal{L}(\mathcal{A}) = \emptyset$.*

These automata are called root-to-frontier automata since they start at the root with an initial state, and then label successor nodes with states according to the transition relation, until they reach the leaves (also called the frontier), which must be labeled by final states to yield a successful run. Frontier-to-root automata (FRAs) work in the other direction. It is well-known that both types of automata accept the same class of tree languages, but only FRAs can be determinized, i.e., deterministic RFAs are weaker than general ones, whereas deterministic FRAs accept the same class of tree languages as general FRAs.

It is well-known that the emptiness problem for RFAs is decidable in polynomial time (see, e.g., [17]). It is also known that, if an RFA $\mathcal{A}$ accepts a tree, then it also accepts one of depth at most q, where q is the number of states of $\mathcal{A}$. In contrast to the emptiness problem, the intersection emptiness problem is ExpTime-complete even for deterministic RFAs [16].

Definition 3. *The RFA $\mathcal{A} = (\Sigma, Q, I, T, F)$ is a* deterministic root-to-frontier automaton (DRFA) *if*

- *the set I of initial states consists of a single initial state q_0,*
- *for all states $q \in Q$ and all symbols f of rank $n > 0$ there exists exactly one n-tuple $(q_1, \ldots, q_n)$ such that $(q, q_1, \ldots, q_n) \in T(f)$.*

For deterministic automata it is often more convenient to use a transition function δ in place of the (functional) transition relations. This function is defined as $\delta(q, f) := (q_1, \ldots, q_n)$, where $(q_1, \ldots, q_n)$ is the unique tuple satisfying $(q, q_1, \ldots, q_n) \in T(f)$.

Given a collection $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DRFAs, the intersection emptiness problem *asks whether $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$. For a natural number k, the* k-restricted intersection emptiness problem *asks, for given DFRAs $\mathcal{A}_1, \ldots, \mathcal{A}_n$, whether there is a tree t with $depth(t) \leq k$ such that $t \in L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n)$.*

The complexity of the k-restricted intersection emptiness problem depends on the encoding of the number k. A proof of the following theorem can be found in [3]. The most challenging task is proving PSpace-hardness for the unary case.

Theorem 1. *The k-restricted intersection emptiness problem for DRFAs is ExpTime-complete if the number k is encoded in binary, and PSpace-complete if the number k is encoded in unary.*

5 Solving Linear Language Equations Using RFAs

As mentioned in Section 3, checking solvability of linear language equations was reduced in [6] to testing emptiness of RFAs. However, this approach cannot directly treat equations of the form (2). It needs equations where the variables X_i are in front of the coefficients S_i. Fortunately, such equations can easily be obtained from the ones of the form (2) by considering the mirror images of the involved languages. For a word $w = i_1 \dots i_\ell \in \Delta^*$, its mirror image is defined as $w^{mi} := i_\ell \dots i_1$, and for a finite set of words $L = \{w_1, \dots, w_m\}$, its mirror image is $L^{mi} := \{w_1^{mi}, \dots, w_m^{mi}\}$. Obviously, the assignment θ with $\theta(X_1) = L_1, \dots, \theta(X_n) = L_n$ is a solution of (2) iff θ^{mi} with $\theta^{mi}(X_1) = L_1^{mi}, \dots, \theta^{mi}(X_n) = L_n^{mi}$ is a solution of the corresponding mirrored equation

$$S_0^{mi} \cup X_1 \cdot S_1^{mi} \cup \dots \cup X_n \cdot S_n^{mi} = T_0^{mi} \cup X_1 \cdot T_1^{mi} \cup \dots \cup X_n \cdot T_n^{mi}. \tag{6}$$

Finite languages over the alphabet $\Delta = \{1, \dots, \rho\}$ can be represented by Σ-trees for the ranked alphabet $\Sigma = \{f_0, f_1, c_0, c_1\}$, where f_0, f_1 are ρ-ary and c_0, c_1 nullary symbols. A given Σ-tree t represents the finite language

$$L_t = \{u \in dom(t) \mid t(u) \in \{c_1, f_1\}\}.$$

Given an equation of the form (6), it is shown in [6] how to construct an RFA $\mathcal{A} = (\Sigma, Q, I, T, F)$ of size exponential in the size of the equation that satisfies the following property.

Lemma 1 (Lemma 6.3 in [6]). *For a Σ-tree t, the following are equivalent:*

1. *The tree t is accepted by $\mathcal{A}$.*
2. *There are finite sets of words $\theta(X_1), \dots, \theta(X_n)$ such that*

$$S_0^{mi} \cup \theta(X_1) \cdot S_1^{mi} \cup \dots \cup \theta(X_n) \cdot S_n^{mi} = L_t =$$
$$T_0^{mi} \cup \theta(X_1) \cdot T_1^{mi} \cup \dots \cup \theta(X_n) \cdot T_n^{mi}.$$

Consequently, equation (2) has a solution iff equation (6) has a solution iff the RFA $\mathcal{A}$ constructed from (6) accepts some tree. Since the size of $\mathcal{A} = (\Sigma, Q, I, T, F)$ is exponential in the size of (2), and the emptiness problem for RFAs is decidable in polynomial time, this yields an ExpTime decision procedure for solvability of equations of the form (2), and thus for unifiability in $\mathcal{FL}_0$. As mentioned in Section 4, it is also shown in [6] by reduction from the intersection emptiness problem for DRFAs, that these problems are actually ExpTime-hard.

Theorem 2 ([6]). *Unifiability in $\mathcal{FL}_0$ as well as solvability of language equations of the forms* (2) *and* (6) *are ExpTime-complete problems.*

In the restricted setting, we consider equations of the form (2) and are looking for solutions θ satisfying (4) for the syntactically restricted setting or satisfying (5) for the semantically restricted setting. Since clearly $(\Delta^{\leq k})^{mi} = \Delta^{\leq k}$, the respective restrictions apply unchanged to the mirrored equation (6).

5.1 The Syntactically Restricted Case

In this case we are thus looking for solutions θ of (6) satisfying

$$\begin{aligned} &S_0^{mi} \cup \theta(X_1)\cdot S_1^{mi} \cup \cdots \cup \theta(X_n)\cdot S_n^{mi} = \\ &T_0^{mi} \cup \theta(X_1)\cdot T_1^{mi} \cup \cdots \cup \theta(X_n)\cdot T_n^{mi} \subseteq \Delta^{\leq k}. \end{aligned} \tag{7}$$

Intuitively, for the trees accepted by the automaton $\mathcal{A}$ this means that we want to check whether $\mathcal{A}$ accepts a tree of depth $\leq k$. This can be achieved by adding a counter that is decremented whenever we go from a node in the tree to a successor node. As soon as the counter reaches 0, no more transitions are possible.

To be more precise, let $\mathcal{A} = (\Sigma, Q, I, T, F)$ be the RFA constructed from (6), as described in [6]. For an integer $k \geq 1$, we define the automaton $\mathcal{A}^k_{syn} = (\Sigma, Q^k_{syn}, I^k_{syn}, T^k_{syn}, F^k_{syn})$ as follows:

- $Q^k_{syn} = Q \times \{0, 1, \ldots, k\}$,
- $I^k_{syn} = I \times \{k\}$,
- $T^k_{syn}(f) = \{((q, i), (q_1, i-1), \ldots, (q_\rho, i-1)) \mid (q, q_1, \ldots, q_\rho) \in T(f) \text{ and } i \geq 1\}$ for $f \in \{f_0, f_1\}$,
- $F^k_{syn}(c) = F(c) \times \{0, 1, \ldots, k\}$ for $c \in \{c_0, c_1\}$.

Basically, $\mathcal{A}^k_{syn}$ works like $\mathcal{A}$, but once it has reached a node at depth k in the tree, it cannot make any transition. Thus, it accepts exactly the trees that have depth at most k and are accepted by $\mathcal{A}$. Since nodes at a depth i correspond to words of this length i, we obtain the following lemma.

Lemma 2. *The automaton $\mathcal{A}^k_{syn}$ accepts a tree t iff* (6) *has a solution θ that satisfies* (7).

Proof. If (6) has a solution θ that satisfies (7), then there is a tree t of depth at most k that represents this solution in the sense that it satisfies 2. of Lemma 1. The tree t then also satisfies 1. of Lemma 1, i.e., it is accepted by $\mathcal{A}$. Since t has depth at most k, it is then also accepted by $\mathcal{A}^k_{syn}$.

Conversely, if the tree t is accepted by $\mathcal{A}^k_{syn}$, then it is also accepted by $\mathcal{A}$ and has depth at most k. The former implies, by Lemma 1, that 2. of Lemma 1 holds, and the latter yields that $L(t) \subseteq \Delta^{\leq k}$. Thus, the sets $\theta(X_1), \ldots, \theta(X_n)$ provided by 2. of Lemma 1 satisfy (7). □

As an easy consequence of this lemma and the connection between syntactically k-restricted unification and the problem of finding solutions of (6) that satisfy (7), we obtain the following complexity results (see [3] for detailed proofs).

Theorem 3. *Given an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C, D as input, the problem of deciding whether the syntactically k-restricted unification problem $C \overset{?}{\equiv}{}^k_{syn} D$ has a unifier or not is ExpTime-complete if the number k is assumed to be encoded in binary, and PSpace-complete if k is assumed to be encoded in unary.*

The ExpTime upper bound is an immediate consequence of the fact that the size of $\mathcal{A}^k_{syn}$ is exponentially bounded by the size of the input equation and the binary representation of k. The ExpTime lower bound can be shown using the fact that the automaton $\mathcal{A}$ accepts a tree iff it accepts one of depth linear in the size of $\mathcal{A}$ (which is exponential in the size of the input equation). Regarding the PSpace upper bound, one cannot construct the exponentially large modified automaton $\mathcal{A}^k_{syn}$ before testing it for emptiness, but rather constructs the relevant parts of $\mathcal{A}^k_{syn}$ while doing the emptiness test on-the-fly. This needs only polynomial space since the depth of the run to be constructed is linear in the size of the unary representation of k. The PSpace lower bound can be shown by reduction of the k-restricted intersection emptiness problem for DRFAs, based on the reduction for the unrestricted case given in [6].

5.2 The Semantically Restricted Case

In this case, to solve the mirrored equation (6), we are looking for assignments θ satisfying

$$\begin{aligned}(S_0^{mi} \cup \theta(X_1)\cdot S_1^{mi} \cup \cdots \cup \theta(X_n)\cdot S_n^{mi}) \cap \Delta^{\leq k} = \\ (T_0^{mi} \cup \theta(X_1)\cdot T_1^{mi} \cup \cdots \cup \theta(X_n)\cdot T_n^{mi}) \cap \Delta^{\leq k}.\end{aligned} \tag{8}$$

The existence of such a solution can again be tested by building an RFA that extends the automaton $\mathcal{A} = (\Sigma, Q, I, T, F)$ constructed from (6), as described in [6], by a counter. But now, we allow the automaton to make transitions where the value of the counter becomes -1. States that have counter value -1 are final states since they indicate that the word represented by this node of the tree is longer than k, and thus it is not relevant for deciding whether the tree represents a solution or not. To be more precise, for an integer $k \geq 1$, we define the automaton $\mathcal{A}^k_{sem} = (\Sigma, Q^k_{sem}, I^k_{sem}, T^k_{sem}, F^k_{sem})$ as follows:

- $Q^k_{sem} = Q \times \{-1, 0, 1, \ldots, k\}$,
- $I^k_{sem} = I \times \{k\}$,
- $T^k_{sem}(f) = \{((q, i), (q_1, i-1), \ldots, (q_\rho, i-1)) \mid (q, q_1, \ldots, q_\rho) \in T(f) \text{ and } i \geq 0\}$ for $f \in \{f_0, f_1\}$,
- $F^k_{sem}(c) = (F(c) \times \{0, 1, \ldots, k\}) \cup \{(q, -1) \mid q \in Q\}$ for $c \in \{c_0, c_1\}$.

The following lemma, whose proof can be found in [3], states correctness of this construction.

Lemma 3. *The automaton $\mathcal{A}^k_{sem}$ accepts a tree t iff there is an assignment θ that satisfies (8).*

Based on this lemma, the complexity upper bounds stated in the following theorem can be shown analogously to the proof of Theorem 3. The PSpace lower bound can be shown by reduction from syntactically k-restricted unification (see [3] for a proof).

Theorem 4. *Given an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C, D as input, the problem of deciding whether the semantically k-restricted unification problem $C \stackrel{?}{\equiv}^{k}_{sem} D$ has a unifier or not is in ExpTime if the number k is assumed to be encoded in binary, and PSpace-complete if k is assumed to be encoded in unary.*

If k is encoded in binary, then the reduction used for the unary case is no longer polynomial. It is an open problem whether, for the case of binary coding, the ExpTime upper bound in Theorem 4 is tight.

6 The Unification Type

Until now, we were mainly interested in the complexity of deciding solvability of unification problems. For this, it is sufficient to consider ground unifiers. Now, we want to investigate the question of whether all unifiers of a given unification problem can be represented as instances of a finite set of (non-ground) unifiers.

In the unrestricted setting, the instance relation between $\mathcal{FL}_0$ unifiers is defined as follows. Let $C \stackrel{?}{\equiv} D$ be an $\mathcal{FL}_0$ unification problem, V the set of concept variables occurring in C and D, and σ, θ two unifiers of this problem. We define

$$\sigma \leq^{\bullet} \theta \text{ if there is a substitution } \lambda \text{ such that } \theta(X) \equiv \lambda(\sigma(X)) \text{ for all } X \in V.$$

If $\sigma \leq^{\bullet} \theta$, then we say that θ is an *instance* of σ.

Definition 4. *Let $C \stackrel{?}{\equiv} D$ be an $\mathcal{FL}_0$ unification problem. The set of substitutions M is called a* complete set of unifiers *for $C \stackrel{?}{\equiv} D$ if it satisfies*

1. *every element of M is a unifier of $C \stackrel{?}{\equiv} D$;*
2. *if θ is a unifier of $C \stackrel{?}{\equiv} D$, then there exists a unifier $\sigma \in M$ such that $\sigma \leq^{\bullet} \theta$.*

The set M is a minimal complete set of unifiers *for $C \stackrel{?}{\equiv} D$ if it additionally satisfies*

3. *if $\sigma, \theta \in M$, then $\sigma \leq^{\bullet} \theta$ implies $\sigma = \theta$.*

The unification type of a given unification problem is determined by the existence and cardinality of such a minimal complete set.

Definition 5. *Let $C \stackrel{?}{\equiv} D$ be an $\mathcal{FL}_0$ unification problem. This problem has type* unitary *(*finitary, infinitary*) if it has a minimal complete set of unifiers of cardinality 1 (finite cardinality, infinite cardinality). If $C \stackrel{?}{\equiv} D$ does not have a minimal complete set of unifiers, then it is of type* zero.

The unification types can be ordered as follows:

$$\text{unitary} < \text{finitary} < \text{infinitary} < \text{type zero}.$$

Basically, the unification type of $\mathcal{FL}_0$ is the maximal type of an $\mathcal{FL}_0$ unification problem. However, in unification theory, one usually distinguishes between unification with and without constants [8]. In an $\mathcal{FL}_0$ unification problem with constants, no restrictions are put on the concepts C and D to be unified. In an $\mathcal{FL}_0$ unification problem without constants, C and D must not contain concept names from N_C. The unification type of $\mathcal{FL}_0$ for unification with (without) constants is the maximal type of an $\mathcal{FL}_0$ unification problem with (without) constants.

It was shown in [6] that equivalence of $\mathcal{FL}_0$ concepts can be axiomatized by the equational theory

$$\begin{aligned} \mathit{ACUIh} := \{\ & (x \wedge y) \wedge z = x \wedge (y \wedge z),\ x \wedge y = y \wedge x,\ x \wedge x = x,\ x \wedge 1 = x\ \} \\ \cup \{\ & h_r(x \wedge y) = h_r(x) \wedge h_r(y),\ h_r(1) = 1 \mid r \in N_R\ \}, \end{aligned}$$

where $\wedge$, h_r, and 1 in the terms respectively correspond to $\sqcap$, $\forall r.$, and $\top$ in the concepts. These identities say that $\wedge$ is associative (A), commutative (C), and idempotent (I) with unit 1 (U), and that the unary function symbols behave like homomorphisms (h) for $\wedge$ and 1.

The unification type of an equational theory is defined analogously to the definitions given above for $\mathcal{FL}_0$ (see [8] for details). It was shown in [1] that the unification type of the theory ACUIh (called AIMH in [1]) is zero, even if one has only one homomorphism h and considers unification without constants. Thus, unification in $\mathcal{FL}_0$ is also of type zero for unification without constants, and thus also for unification with constants. We will show in this section that this is no longer the case if we consider restricted unification. For the semantically restricted case, this is an easy consequence of general results about commutative/monoidal theories [1,14].

6.1 The Semantically Restricted Case

The equivalence $\equiv^k_{sem}$ can be axiomatized by adding identities to ACUIh that say that nesting of homomorphisms of depth $> k$ produces the unit. Given a word $u = r_1 r_2 \ldots r_n \in N_R^*$, we denote a term of the form $h_{r_1}(h_{r_2}(\cdots h_{r_n}(t)\cdots))$ as $h_u(t)$. It is now easy to see that $\equiv^k_{sem}$ is axiomatized by

$$\mathit{ACUIh}^k := \mathit{ACUIh} \cup \{h_u(x) = 1 \mid u \in N_R^* \text{ with } |u| = k+1\}.$$

Both ACUIh and ACUIh^k are so-called commutative/monoidal theory [1,14,7], for which unification can be reduced to solving linear equations over a corresponding semiring. For ACUIh this semiring consists of finite languages over the alphabet Δ with union as addition and concatenation as multiplication [6]. As shown in [3], the semiring corresponding to ACUIh^k consists of the subsets of $\Delta^{\leq k}$, with union as addition and the following multiplication: $L_1 \cdot_k L_2 = (L_1 \cdot L_2) \cap \Delta^{\leq k}$.

According to [14], unification without constants in a monoidal theory E is unitary if the semiring S_E corresponding to E is finite. In [1], the same result is shown for commutative theories E under the assumption that the finitely generated E-free algebras are finite. It is easy to see that these two conditions actually coincide for commutative theories [7]. In addition to unification without constants, also unification with constants is considered in [1], and it is shown that, if the finitely generated E-free algebras are finite, then unification with constants in the commutative theory E is at most finitary (i.e., unitary or finitary). The following theorem is an easy consequence of these results.

Theorem 5. *Unification in $ACUIh^k$, and thus also semantically k-restricted unification in $\mathcal{FL}_0$, is unitary for unification without constants and finitary for unification with constants.*

Proof. It is easy to see that the semiring corresponding to $ACUIh^k$ is finite since its elements are all the subsets of the finite set $\Delta^{\leq k}$. Thus, the results in [1,14] yield that $ACUIh^k$ is unitary for unification without constants and at most finitary for unification with constants. The following example shows that the theory is not unitary for unification with constants: if x is a variable and a a constant, then the terms $x \wedge a$ and a have (restricted to x) exactly two $ACUIh^k$ unifiers $\{x \mapsto 1\}$ and $\{x \mapsto a\}$, which are not in any instance relationship. □

6.2 The Syntactically Restricted Case

To deal with the syntactically restricted case, the results on the unification type for commutative/monoidal theories cannot be applied directly, but we can show the same results as for the semantically restricted case, using the ideas underlying the proofs in [1,14]. We will formulate our proof using the syntax of $\mathcal{FL}_0$ rather than the equational theory variant.

Let C, D be $\mathcal{FL}_0$ concepts and σ a syntactically k-restricted unifier of C and D. Let $X_1, \ldots, X_n$ be the concept variables occurring in C, D and $A_1, \ldots, A_\ell$ the concept constants. First, note that we can assume that σ does not introduce new concept constants since otherwise one could get a more general unifier by replacing such a constant by a new variable. Let $Y_1, \ldots, Y_m$ be the concept variables in the range of σ, where we assume without loss of generality that they are different from the variables $X_1, \ldots, X_n$. For $i = 1, \ldots, n$, the LNF of the concept $\sigma(X_i)$ is of the form

$$\sigma(X_i) = K_i \sqcap \forall L_{i,1}.Y_1 \sqcap \ldots \sqcap \forall L_{i,m}.Y_m, \tag{9}$$

where K_i is a concept of role depth $\leq k$ not containing concept variables and only concept constants in $\{A_1, \ldots, A_\ell\}$ and the $L_{i,j}$ are subsets of $\Delta^{\leq k}$. Recall that $\forall L_{i,j}.Y_j$ abbreviates the conjunction of the value restrictions $\forall w.Y_j$ for $w \in L_{i,j}$, which in turn is an abbreviation for $\forall r_1. \cdots \forall r_\nu.Y_j$ if $w = r_1 \ldots r_\nu$.

Now, consider for every variable $Y_j, 1 \leq j \leq m$, the tuple of languages $L(Y_j) = (L_{1,j}, \ldots, L_{n,j})$, and assume that there are indices $j \neq j'$ such that $L(Y_j) = L(Y_{j'})$. Let $\theta_{j'}$ be the substitution that replaces $Y_{j'}$ with $\top$ and leaves all other variable Y_μ unchanged. Then $\sigma_{j'} = \sigma\theta_{j'}$ is an instance of σ, which is still a syntactically k-restricted unifier of C and D, but introduces one variable less. Conversely, using the substitution $\lambda_{j,j'} = \{Y_j \mapsto Y_j \sqcap Y_{j'}\}$, we obtain σ as an instance of $\sigma_{j'}$ since $\sigma = \sigma_{j'}\lambda_{j,j'}$.

Let c denote the (finite) cardinality of $\Delta^{\leq k}$. Then there are at most $2^{c \cdot n}$ different n-tuples of subsets of $\Delta^{\leq k}$. Thus, if a syntactically k-restricted unifier of C and D introduces more than $2^{c \cdot n}$ variables, it is an instance of a syntactically k-restricted unifier of C and D that introduces at least one variable less. This observation can be used to show the following lemma.

Lemma 4. *There is a complete set of syntactically k-restricted unifiers of C and D that consists of unifiers whose range contains at most the variables $Y_1, \ldots, Y_m$ for $m = 2^{c \cdot n}$.*

Once we have restricted the unifiers in the complete set to ones using only finitely many variables, we know that there can be only finitely many unifiers in this set. In fact, if we consider (9), then we see that the K_i and $L_{i,j}$ range over finite sets. This proves the following theorem.

Theorem 6. *Syntactically k-restricted unification with constants in $\mathcal{FL}_0$ is finitary.*

To show that unification with constants is not unitary, we can use the same example as in the semantically restricted case. Unification without constants is again unitary.

Corollary 1. *Syntactically k-restricted unification without constants in $\mathcal{FL}_0$ is unitary.*

Proof. By the previous theorem, there is a finite complete set $\{\sigma_1, \ldots, \sigma_\kappa\}$ of syntactically k-restricted unifiers of C, D. Without loss of generality, we can assume that the variables occurring in the ranges of these unifiers are disjoint and that no concept constant occurs in the range. The latter assumption can be made since the unification problem itself does not contain such constants. Under these assumptions, the substitution σ defined as

$$\sigma(X_i) = \sigma_1(X_i) \sqcap \ldots \sqcap \sigma_\kappa(X_i) \quad \text{for } i = 1, \ldots, n$$

is also a syntactically k-restricted unifier of C, D, and it has the substitutions $\sigma_1, \ldots, \sigma_\kappa$ as instances (see [3] for a proof of these two claims). This shows that $\{\sigma\}$ is a complete set of unifiers. □

7 Conclusion

We have investigated both a semantically and a syntactically restricted variant of unification in $\mathcal{FL}_0$, where either the role depth of concepts or the length of role paths in interpretations is restricted by a natural number $k \geq 1$. These restrictions lead to a considerable improvement of the unification type from the worst possible type to unitary/finitary for unification without/with constants. For the complexity of the decision problem, we only obtain an improvement if k is assumed to be encoded in unary.

While these results are mainly of (complexity) theoretic interest, they could also have a practical impact. In fact, in our experiments with the system UEL, which implements several unification algorithms for the DL $\mathcal{EL}$ [4], we have observed that the algorithms usually yield many different unifiers, and it is hard to choose one that is appropriate for the application at hand (e.g., when generating new concepts using unification [2]). For this reason, we added additional constraints to the unification problem to ensure that the generated concepts are of a similar shape as the concepts already present in the ontology [2]. It makes sense also to use a restriction on the role depth as such an additional constraint since the role depth of the (unfolded) concepts occurring in real-world ontologies is usually rather small. This claim is supported by our experiments with the medical ontology SNOMED CT,[5] which has a maximal role depth of 10, and the acyclic ontologies in Bioportal 2017,[6] where a large majority also has a role depth of at most 10.

As future work, we will investigate whether the ExpTime upper bound in Theorem 4 for the case of binary coding of k is tight. In addition, we will consider similar restrictions for other DLs. For example, the unification type of the DL $\mathcal{EL}$ is also known to be zero, and the decision problem is NP-complete [5]. We conjecture that, for $\mathcal{EL}$, the restricted variants will not lead to an improvement of unification type or complexity.

In [11], a syntactically restricted version of unification in the theory ACh was shown to be decidable, but neither the unification type nor the complexity of the decision problem was determined. It would be interesting to investigate these problems and also consider a semantically restricted variant. Note that, with the exception of the missing unit, ACh is commutative/monoidal, but the main difference to $ACUIh$ is that already the semiring corresponding to the subtheory without homomorphisms is infinite, whereas the semiring corresponding to $ACUI$ is finite.

Acknowledgements

Franz Baader was partially supported by DFG TRR 248 (cpec, grant 389792660), Oliver Fernández Gil by DFG in project number 335448072, and Maryam Rostamigiv by a DAAD Short-Term Grant, 2021 (57552336). The authors should like to thank Patrick Koopmann for determining the maximal role depth of concepts in ontologies from Bioportal 2017 and in SNOMED CT.

[5] `https://www.snomed.org/`
[6] `https://zenodo.org/record/439510`

References

1. Franz Baader. Unification in commutative theories. *J. Symbolic Computation*, 8(5):479–497, 1989.
2. Franz Baader, Stefan Borgwardt, and Barbara Morawska. Constructing SNOMED CT concepts via disunification. LTCS-Report 17-07, Chair for Automata Theory, Institute for Theoretical Computer Science, Technische Universität Dresden, Dresden, Germany, 2017. `https://lat.inf.tu-dresden.de/research/reports/2017/BaBM-LTCS-17-07.pdf`.
3. Franz Baader, Oliver Fernández Gil, and Maryam Rostamigiv. Restricted unification in the DL $\mathcal{FL}_0$ (extended version). LTCS-Report 21-02, Chair of Automata Theory, Institute of Theoretical Computer Science, Technische Universität Dresden, Dresden, Germany, 2021. `https://lat.inf.tu-dresden.de/research/reports/2021/BaGiRo21.pdf`.
4. Franz Baader, Julian Mendez, and Barbara Morawska. UEL: Unification solver for the description logic $\mathcal{EL}$ – system description. In *Proc. of the 6th International Joint Conference on Automated Reasoning (IJCAR 2012)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 45–51. Springer, 2012.
5. Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. *Logical Methods in Computer Science*, 6(3), 2010.
6. Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *J. Symbolic Computation*, 31(3):277–305, 2001.
7. Franz Baader and Werner Nutt. Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning. *J. Applicable Algebra in Engineering, Communication and Computing*, 7(4):309–337, 1996.
8. Franz Baader and Wayne Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers, 2001.
9. Philippe Balbiani, Cigdem Gencer, Maryam Rostamigiv, and Tinko Tinchev. About the unification type of $\mathbf{K} + \Box\Box\bot$. In *Proc. of the 34th International Workshop on Unification (UNIF 2020)*, pages 4:1–4:6. RISC-Linz, 2020.
10. Emil Jerabek. Blending margins: The modal logic K has nullary unification type. *J. Logic and Computation*, 25(5):1231–1240, 2015.
11. Ajay Kumar Eeralla and Christopher Lynch. Bounded ACh unification. *Math. Struct. Comput. Sci.*, 30(6):664–682, 2020.
12. Hector J. Levesque and Ron J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.
13. Paliath Narendran. Solving linear equations over polynomial semirings. In *Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 466–472. IEEE Computer Society, 1996.
14. Werner Nutt. Unification in monoidal theories. In Mark E. Stickel, editor, *Proc. of the 10th Int. Conf. on Automated Deduction (CADE 1990)*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 618–632, Kaiserslautern, Germany, 1990. Springer.
15. Rafael Peñaloza and Anni-Yasmin Turhan. A practical approach for computing generalization inferences in $\mathcal{EL}$. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *Proc. of the 8th Extended Semantic Web Conference (ESWC 2011)*, volume 6643 of *Lecture Notes in Computer Science*, pages 410–423. Springer, 2011.

16. Helmut Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52:57–60, 1994.
17. Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 134–189. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.