TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Approximate Unification in the Description Logic $\mathcal{FL}_0$

Franz Baader Pavlos Marantidis Alexander Okhotin

LTCS-Report 16-04

Submitted to *Jelia 16*

Postal Address:
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nöthnitzer Str. 46
Dresden

Approximate Unification in the Description Logic $\mathcal{FL}_0$

Franz Baader[1⋆], Pavlos Marantidis[1⋆⋆], and Alexander Okhotin[2]
firstname.lastname@tu-dresden.de, alexander.okhotin@utu.fi

[1] Theoretical Computer Science, TU Dresden, Germany
[2] Chebyshev Laboratory, St. Petersburg State University, Russia

Abstract. Unification in description logics (DLs) has been introduced as a novel inference service that can be used to detect redundancies in ontologies, by finding different concepts that may potentially stand for the same intuitive notion. It was first investigated in detail for the DL $\mathcal{FL}_0$, where unification can be reduced to solving certain language equations. In order to increase the recall of this method for finding redundancies, we introduce and investigate the notion of approximate unification, which basically finds pairs of concepts that "almost" unify. The meaning of "almost" is formalized using distance measures between concepts. We show that approximate unification in $\mathcal{FL}_0$ can be reduced to approximately solving language equations, and devise algorithms for solving the latter problem for two particular distance measures.

1 Introduction

Description logics [1] are a well-investigated family of logic-based knowledge representation formalisms. They can be used to represent the relevant concepts of an application domain using concept descriptions, which are built from concept names and role names using certain concept constructors. In this paper, we concentrate on the DL $\mathcal{FL}_0$, which offers the constructors conjunction ($\sqcap$), value restriction ($\forall r.C$), and the top concept ($\top$).

Unification in DLs has been introduced as a novel inference service that can be used to detect redundancies in ontologies, and was first investigated in detail for $\mathcal{FL}_0$ [4]. For example, assume that one developer of a medical ontology defines the concept of a *patient with severe head injury* as

$$\mathsf{Patient} \sqcap \forall \mathsf{finding}.(\mathsf{Head\_injury} \sqcap \forall \mathsf{severity}.\mathsf{Severe}), \tag{1}$$

whereas another one represents it as

$$\mathsf{Patient} \sqcap \forall \mathsf{finding}.(\mathsf{Severe\_finding} \sqcap \mathsf{Injury} \sqcap \forall \mathsf{finding\_site}.\mathsf{Head}). \tag{2}$$

Formally, these two concept descriptions are not equivalent, but they are nevertheless meant to represent the same concept. They can obviously be made equivalent by treating the concept names Head_injury and Severe_finding as variables, and substituting the first one by Injury $\sqcap$ ∀finding_site.Head and the second one by ∀severity.Severe. In this case, we say that the descriptions are unifiable, and call the substitution that makes them equivalent a *unifier*. Intuitively, such a unifier proposes definitions for the concept names that are used as variables: in our example, we know that, if we define Head_injury as Injury $\sqcap$ ∀finding_site.Head and Severe_finding as ∀severity.Severe, then the two concept descriptions (1) and (2) are equivalent w.r.t. these definitions.

⋆ Supported by the Cluster of Excellence 'Center for Advancing Electronics Dresden'.
⋆⋆ Supported by DFG Graduiertenkolleg 1763 (QuantLA).

Of course, this example was constructed such that a unifier providing sensible definitions for the concept names used as variables actually exists. It is based on the assumption that both knowledge engineers had the same definition of the concept *patient with severe head injury* in mind, but have modelled certain subconcepts on different levels of granularity. Whereas the first knowledge engineer used Head_injury as a primitive (i.e., not further defined) concept, the other one provided a more detailed definition for *head injury*; and the other way round for *severe finding*. But what if there are more differences between the two concepts, maybe due to small modelling errors? For example, assume that a third knowledge engineer has left out the concept name Severe_finding from (2), based on the assumption that all injuries with finding site head are severe:

$$\mathsf{Patient} \sqcap \forall \mathsf{finding}.(\mathsf{Injury} \sqcap \forall \mathsf{finding\_site}.\mathsf{Head}). \tag{3}$$

The concept descriptions (1) and (3) cannot be unified if only Head_injury is used as a variable. Nevertheless, the substitution that replaces Head_injury by $\mathsf{Injury} \sqcap \forall \mathsf{finding\_site}.\mathsf{Head}$ makes these two descriptions quite similar, though not equivalent. We call such a substitution an *approximate unifier*.

The purpose of this paper is to introduce and investigate the notion of approximate unification for the DL $\mathcal{FL}_0$. Basically, to formalize approximate unification, we first need to fix the notion of a distance between $\mathcal{FL}_0$ concept descriptions. An approximate unifier is then supposed to make this distance as small as possible. Of course, there are different ways of defining the distance between concept descriptions, which then also lead to different instances of approximate unification. In this paper, we consider two such distance functions, which are based on the idea that differences at larger role depth (i.e., further down in the nesting of value restrictions) are less important than ones at smaller role depth. The first distance considers only the smallest role depth ℓ where the difference occurs (and then uses $2^{-\ell}$ as distance), whereas the second one "counts" all differences, but the ones at larger role depth with a smaller weight. This idea is in line with work on nonstandard inferences in DLs that approximate least common subsumers and most specific concepts by fixing a bound on the role depth [8].

Exact unification in $\mathcal{FL}_0$ was reduced in [4] to solving certain language equations, which in turn was reduced to testing certain tree automata for emptiness. We show that this approach can be extended to approximate unification. In fact, by linking distance functions on concept descriptions with distance functions on languages, we can reduce approximate unification in $\mathcal{FL}_0$ to approximately solving language equations. In order to reduce this problem to a problem for tree automata, we do not employ the original construction of [4], but the more sophisticated one of [5]. Using this approach, both the decision variant (is there a substitution that makes the distance smaller than a threshold) and the computation variant (compute the infimum of the achievable distances) of approximate unification can be solved in exponential time, and are thus of the same complexity as exact unification in $\mathcal{FL}_0$.

2 Unification in $\mathcal{FL}_0$

We will first recall syntax and semantics of $\mathcal{FL}_0$ and describe the normal form of $\mathcal{FL}_0$ concept descriptions that is based on representing value restrictions as finite languages over the alphabet of role names. Then, we introduce unification in $\mathcal{FL}_0$ and recall how it can be reduced to solving language equations.

Syntax and semantics

The *concept descriptions* C of the DL $\mathcal{FL}_0$ are built recursively over a finite set of concept names N_c and a finite set of role names N_r using the following syntax rules:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C, \tag{4}$$

where $A \in N_c$ and $r \in N_r$. In the following, we assume that $N_c = \{A_1, \ldots, A_k\}$ and $N_r = \{r_1, \ldots, r_n\}$.

The semantics of $\mathcal{FL}_0$ is defined in the usual way, using the notion of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of a nonempty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns binary relations on $\Delta^{\mathcal{I}}$ to role names and subsets of $\Delta^{\mathcal{I}}$ to concept names. The interpretation function $\cdot^{\mathcal{I}}$ is extended to $\mathcal{FL}_0$ concept descriptions as follows: $\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\forall r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}} : \text{ if } (d, e) \in r^{\mathcal{I}}, \text{then } e \in C^{\mathcal{I}}\}$.

Equivalence and normal form

Two $\mathcal{FL}_0$ concept descriptions C, D are *equivalent* (written $C \equiv D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$.

As an easy consequence of the semantics of $\mathcal{FL}_0$, we obtain that value restrictions $(\forall s.\cdot)$ distribute over conjunction $(\sqcap)$, i.e., $\forall s.(C \sqcap D) \equiv \forall s.C \sqcap \forall s.D$ holds for all $\mathcal{FL}_0$ concept descriptions C, D. Using this equivalence from left to right, we can rewrite every $\mathcal{FL}_0$ concept description into a finite conjunction of descriptions $\forall s_1. \cdots \forall s_m.A$, where $m \geq 0, s_1, \ldots, s_m \in N_r$, and $A \in N_c$. We further abbreviate $\forall s_1. \cdots \forall s_m.A$ as $\forall (s_1 \ldots s_m).A$, where $s_1 \ldots s_m$ is viewed to be a word over the alphabet of all role names N_r, i.e., an element of N_r^*. For $m = 0$, this is the empty word ε. Finally, grouping together value restrictions that end with the same concept name, we abbreviate conjunctions $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ as $\forall \{w_1, \ldots, w_\ell\}.A$, where $\{w_1, \ldots, w_\ell\} \subseteq N_r^*$ is viewed to be a (finite) language over N_r. Additionally we use the convention that $\forall \emptyset.A$ is equivalent to $\top$. Then, any $\mathcal{FL}_0$ concept description C (over $N_c = \{A_1, \ldots, A_k\}$ and $N_r = \{r_1, \ldots, r_n\}$) can be rewritten into the *normal form* $\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k$, where $L_1, \ldots L_k$ are finite languages over the alphabet N_r. For example, if $k = 3$, then the concept description $A_1 \sqcap \forall r_1.(A_1 \sqcap \forall r_1.A_2 \sqcap \forall r_2.A_1)$ has the normal form $\forall \{\varepsilon, r_1, r_1r_2\}.A_1 \sqcap \forall \{r_1r_1\}.A_2 \sqcap \forall \emptyset.A_3$. Using this normal form, equivalence of $\mathcal{FL}_0$ concept descriptions can be characterized as follows (see [4] for a proof).

Lemma 1. *Let $C = \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k$ and $D = \forall M_1.A_1 \sqcap \ldots \sqcap \forall M_k.A_k$ be $\mathcal{FL}_0$ concept descriptions in normal form. Then*

$$C \equiv D \text{ iff } L_1 = M_1, \ldots, L_k = M_k.$$

Consider the head injury example from the introduction, where for brevity we replace the concept and role names by single letters: (1) thus becomes $A \sqcap \forall r.(X \sqcap \forall s.B)$ and (2) becomes $A \sqcap \forall r.(Y \sqcap D \sqcap \forall t.E)$. The normal forms of these two concept descriptions are

$$\begin{aligned} &\forall \{\varepsilon\}.A \sqcap \forall \{rs\}.B \sqcap \forall \emptyset.D \sqcap \forall \emptyset.E \sqcap \forall \{r\}.X \sqcap \forall \emptyset.Y, \\ &\forall \{\varepsilon\}.A \sqcap \forall \emptyset.B \sqcap \forall \{r\}.D \sqcap \forall \{rt\}.E \sqcap \forall \emptyset.X \sqcap \forall \{r\}.Y. \end{aligned} \tag{5}$$

Unification

In order to define unification in $\mathcal{FL}_0$, we need to introduce an additional set of concept names N_v, whose elements we call *concept variables*. Intuitively, N_v contains the concept names that

have possibly been given another name or been specified in more detail in another concept description describing the same notion. From a syntactic point of view, concept variables are treated like concept names when building concepts. We call expressions built using the syntax rules (4), but with $A \in N_c \cup N_v$, *concept patterns*, to distinguish them from concept descriptions, where only $A \in N_c$ is allowed. The difference between elements of N_c and N_v is that concept variables can be replaced by substitutions.

A *substitution* σ is a function that maps every variable $X \in N_v$ to a concept description $\sigma(X)$. This function can be extended to concept patterns, by setting $\sigma(A) := A$ for $A \in N_c \cup \{\top\}$, $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$, and $\sigma(\forall r.C) := \forall r.\sigma(C)$. We denote the set of all substitutions as Sub.

Definition 1 (Unification). *The substitution σ is a* unifier *of the two $\mathcal{FL}_0$ concept patterns C, D if $\sigma(C) \equiv \sigma(D)$. If C, D have a unifier, then we call them* unifiable. *The $\mathcal{FL}_0$* unification problem *asks whether two given $\mathcal{FL}_0$ concept patterns are unifiable or not.*

In [4] it is shown that the $\mathcal{FL}_0$ unification problem is ExpTime-complete. The ExpTime upper bound is proved by a reduction to language equations, which in turn are solved using tree automata. Here we sketch the reduction to language equations. The reduction to tree automata will be explained in Section 4. Without loss of generality, we can assume that the input patterns are in normal form (where variables are treated like concept names), i.e.,

$$\begin{aligned} C &= \forall S_{0,1}.A_1 \sqcap \ldots \sqcap \forall S_{0,k}.A_k \sqcap \forall S_1.X_1 \sqcap \ldots \sqcap \forall S_m.X_m, \\ D &= \forall T_{0,1}.A_1 \sqcap \ldots \sqcap \forall T_{0,k}.A_k \sqcap \forall T_1.X_1 \sqcap \ldots \sqcap \forall T_m.X_m, \end{aligned} \tag{6}$$

where $S_{0,i}, T_{0,i}, S_j, T_j$ are finite languages over N_r. The unification problem for C, D can be reduced to (independently) solving the language equations

$$S_{0,i} \cup S_1 \cdot X_{1,i} \cup \ldots \cup S_m \cdot X_{m,i} = T_{0,i} \cup T_1 \cdot X_{1,i} \cup \ldots \cup T_m \cdot X_{m,i} \tag{7}$$

for $i = 1, \ldots, k$, where "$\cdot$" stands for concatenation of languages. A *solution* σ_i of such an equation is an *assignment* of languages (over N_r) to the variables $X_{j,i}$ such that $S_{0,i} \cup S_1 \cdot \sigma_i(X_{1,i}) \cup \ldots \cup S_m \cdot \sigma_i(X_{m,i}) = T_{0,i} \cup T_1 \cdot \sigma_i(X_{1,i}) \cup \ldots \cup T_m \cdot \sigma_i(X_{m,i})$. This assignment is called *finite* if all the languages $\sigma_i(X_{j,i})$ are finite. We denote the set of all assignments as Ass and the set of all finite assignments as $finAss$.

As shown in [4], C, D are unifiable iff the language equations of the form (7) have finite solutions for all $i = 1, \ldots, k$. In fact, given finite solutions $\sigma_1, \ldots, \sigma_k$ of these equations, a unifier of C, D can be obtained by setting

$$\sigma(X_i) := \forall \sigma_i(X_{i,1}).A_1 \sqcap \ldots \sqcap \forall \sigma_i(X_{i,k}).A_k, \tag{8}$$

and every unifier of C, D can be obtained in this way. Of course, this construction of a substitution from a k-tuple of finite assignments can be applied to arbitrary finite assignments (and not just to finite solutions of the equations (7)), and it yields a bijection ρ between k-tuples of finite assignments and substitutions.

Coming back to our example (5), where we now view X, Y as variables, the language equations for the concept names A and B are

$$\begin{aligned} \{\varepsilon\} \cup \{r\} \cdot X_A \cup \emptyset \cdot Y_A &= \{\varepsilon\} \cup \emptyset \cdot X_A \cup \{r\} \cdot Y_A, \\ \{rs\} \cup \{r\} \cdot X_B \cup \emptyset \cdot Y_B &= \emptyset \cup \emptyset \cdot X_B \cup \{r\} \cdot Y_B. \end{aligned}$$

Among others, the first equation has $X_A = Y_A = \emptyset$ as a solution, and the second $X_B = \emptyset$ and $Y_B = \{s\}$. The equations for D, E are built in a similar way, and $X_D = \{\varepsilon\}, Y_D = \emptyset$ and $X_E = \{t\}, Y_E = \emptyset$ are solutions of these equations. Using (8), but leaving out the value restrictions for $\emptyset$, these solutions yield the unifier σ with $\sigma(X) = \forall\{\varepsilon\}.D \sqcap \forall\{t\}.E \equiv D \sqcap \forall t.E$ and $\sigma(Y) = \forall\{s\}.B \equiv \forall s.B$.

3 Approximate unifiers and solutions

As motivated in the introduction, it makes sense to look for substitutions σ that are actually not unifiers, but come close to being unifiers, in the sense that the distance between $\sigma(C)$ and $\sigma(D)$ is small. We call such substitutions *approximate unifiers*. In the following, we will first recall some definitions regarding distances from metric topology [14,11]. Subsequently, we will first introduce approximate unification based on distances between concept descriptions, and then approximately solving language equations based on distances between languages. Next, we will show how distances between languages can be used to define distances between concept descriptions, and that approximate unification for distances obtained this way can be reduced to approximately solving language equations.

Metric topology

Given a set X, a *metric* (or *distance*) on X is a mapping $d\colon X \times X \to [0, \infty)$ that satisfies the properties:

$(M1)$ $d(a, b) = 0 \iff a = b$
$(M2)$ $d(a, b) = d(b, a)$
$(M3)$ $d(a, c) \leqslant d(a, b) + d(b, c)$

In this case, (X, d) is called a *metric space*. A useful metric on $\mathbb{R}^k$ (that will be used later) is the Chebyshev distance, d_∞, which for $\boldsymbol{x} = (x_1, \ldots, x_k)$, $\boldsymbol{y} = (y_1, \ldots, y_k) \in \mathbb{R}^k$ is defined as

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1,\ldots,k} |x_i - y_i|.$$

Given a metric space (X, d), a sequence (a_n) of elements of X is said to *converge* to $a \in X$ (written $a_n \xrightarrow{d} a$) if for every $\epsilon > 0$ there is an $n_0 \in \mathbb{N}$ s.t. $d(a_n, a) < \epsilon$ for every $n \geqslant n_0$. For a sequence $((a_1^n, \ldots, a_k^n))$ of elements of $\mathbb{R}^k$, we have that $(a_1^n, \ldots, a_k^n) \xrightarrow{d_\infty} (a_1, \ldots, a_k)$ iff $a_i^n \xrightarrow{d} a_i$ for every $i = 1, \ldots, k$. A sequence (a_n) is called a *Cauchy* sequence, if for every $\epsilon > 0$, there exists an $n_0 \in \mathbb{N}$, s.t. for every $m, n \geqslant n_0$ it holds that $d(a_n, a_m) < \epsilon$. A metric space (X, d) is called *complete*, if every Cauchy sequence converges to a point in X. It is well known that the metric space $(\mathbb{R}^k, d_\infty)$ is complete [11].

Definition 2. *Given a metric space (X, d) a function $f\colon X \to X$ is called a* contraction*, if there is a $\lambda \in (0, 1)$ such that $d(f(x), f(y)) \leqslant \lambda d(x, y)$ for any $x, y \in X$.*

Theorem 1 (Banach's Fixed Point Theorem [7,11]). *Let (X, d) be a complete metric space and a function $f\colon X \to X$ be a contraction. Then there exists a unique $p \in X$ such that $f(p) = p$.*

Furthermore, let $(X, d), (Y, d')$ be metric spaces. A function $f\colon X \to Y$ is called *continuous*, if for every sequence (a_n) of X, it holds that

$$a_n \xrightarrow{d} a \implies f(a_n) \xrightarrow{d'} f(a).$$

Finally, we provide the formal definition of the *infimum* of a set of real numbers, which will be needed in the proofs.

Definition 3. *Given a set of real numbers S, we say that p is the* infimum *of S, and denote this by $p = \inf S$ if the following two conditions hold:*

(a) $p \leqslant s$ *for all* $s \in S$*, i.e.* p *is a lower bound of* S*,*
(b) *for all* $\epsilon > 0$*, there is an* $s \in S$ *such that* $s < p + \epsilon$*.*

Note that this means that if $p = \inf S$, there exists a sequence (s_n) of elements of S, s.t. $s_n \xrightarrow{d_\infty} p$. Furthermore, if $f\colon \mathbb{R}^n \to \mathbb{R}^m$

Approximate unification

In order to define how close $\sigma(C)$ and $\sigma(D)$ are, we need to use a function that measures the distance between these two concept descriptions. We say that a function that takes as input a pair of $\mathcal{FL}_0$ concept descriptions and yields as output an element of $[0, \infty)$ is a *concept distance* for $\mathcal{FL}_0$ if it satisfies the following three properties:

- equivalence closedness: $m(C, D) = 0 \iff C \equiv D$,
- symmetry: $m(C, D) = m(D, C)$,
- equivalence invariance: $C \equiv D \implies m(C, E) = m(D, E)$.

Note that equivalence closedness corresponds to $(M1)$ and symmetry to $(M2)$ in the definition of a metric. Equivalence invariance ensures that m can be viewed as operating on equivalence classes of concept descriptions.

Definition 4 (Approximate unification). *Given a concept distance* m*,* $\mathcal{FL}_0$ *concept patterns* C, D*, and a substitution* σ*, the* degree of violation *of* σ *is defined as* $v_m(\sigma, C, D) := m(\sigma(C), \sigma(D))$*. For* $p \in \mathbb{Q}$*, we say that* σ *is a* p-approximate unifier of C, D *if* $2^{-p} > v_m(\sigma, C, D)$*.*

Equivalence closedness of m yields that $v_m(\sigma, C, D) = 0$ iff σ is a unifier of C, D.

The *decision problem* for approximate unification asks, for a given threshold $p \in \mathbb{Q}$, whether C, D have a p-approximate unifier or not. In addition, we consider the following *computation problem*: compute $\inf_{\sigma \in Sub} v_m(\sigma, C, D)$. The following lemma, which is immediate from the definitions, shows that a solution of the computation problem also yields a solution of the decision problem.

Lemma 2. *Let* m *be a concept distance and* C, D $\mathcal{FL}_0$ *concept patterns. Then* C, D *have a* p*-approximate unifier iff* $2^{-p} > \inf_{\sigma \in Sub} v_m(\sigma, C, D)$*.*

Proof. By definition, if C, D have a p-approximate unifier, then there exists a substitution $\sigma \in Sub$, s.t. $p > v_m(\sigma, C, D)$, and thus $p > \inf_{\sigma \in Sub} v_m(\sigma, C, D)$.

Suppose now that $p > \inf_{\sigma \in Sub} v_m(\sigma, C, D)$. By the definition of infimum, for every $\epsilon > 0$, there exists a $\sigma_\epsilon \in Sub$ s.t. $v_m(\sigma_\epsilon, C, D) \leqslant \inf_{\sigma \in Sub} v_m(\sigma, C, D) + \epsilon$. Thus, for $\epsilon < p - \inf_{\sigma \in Sub} v_m(\sigma, C, D)$ we have the required result. □

The reduction of the decision problem to the computation problem obtained from this lemma is actually polynomial. In fact, though the size of a representation of the number 2^{-p} may be exponential in the size of a representation of p, the number 2^{-p} need not be computed. Instead, we can compare p with $\log_2 \inf_{\sigma \in Sub} v_m(\sigma, C, D)$, where for the comparison we only need to compute as many digits of the logarithm as p has.

Approximately solving language equations

Following [5], we consider a more general form of language equations than the one given in (7). Here, all Boolean operators (and not just union) are available. *Language expressions* are built recursively over a finite alphabet Σ using union, intersection, complement, and concatenation of regular languages from the left, as formalized by the following syntax rules:

$$\phi ::= L \mid X \mid \phi \cup \phi \mid \phi \cap \phi \mid \sim\phi \mid L \cdot \phi, \tag{9}$$

where L can be instantiated with any regular language over Σ and X with any variable. We assume that all the regular languages occurring in an expression are given by finite automata. Obviously, the left- and the right-hand sides of (7) are such language expressions. As before, an *assignment* $\sigma \in Ass$ maps variables to languages over Σ. It is extended to expressions in the obvious way (where $\sim$ is interpreted as set complement). The assignment σ *solves* the *language equation* $\phi = \psi$ if $\sigma(\phi) = \sigma(\psi)$. For *finite solvability* we require the languages $\sigma(X)$ to be finite, i.e., σ should be an element of $finAss$.

In order to define approximate solutions, we need the notion of distances between languages. A function $d : 2^{\Sigma^*} \times 2^{\Sigma^*} \to [0, \infty)$ satisfying (M1), (M2), and (M3) is called a *language distance*.

Definition 5 (Approximate solutions). *Given a language distance d, language expressions ϕ, ψ, and an assignment σ, the* degree of violation *of σ is defined as $v_d(\sigma, \phi, \psi) := d(\sigma(\phi), \sigma(\psi))$. For $p \in \mathbb{Q}$, we say that σ is a p-*approximate solution of $\phi \approx \psi$ *if $2^{-p} > v_d(\sigma, \phi, \psi)$.*

The *decision* and the *computation problem* for approximately solving language equations are defined analogously to the case of unification. In addition, the analog of Lemma 2 also holds in this case, and thus the decision problem can be reduced to the computation problem.

Recall that unification in $\mathcal{FL}_0$ is reduced to *finite* solvability of language equations. The above definition of approximate solutions and of the decision and the computation problem can also be restricted to finite assignments, in which case we talk about *finite approximate solvability*. However, we will show that finite approximate solvability can actually be reduced to approximate solvability. For this to be the case, we need the language distance to satisfy an additional property (M4). Given a natural number ℓ, we call two languages $K, L \subseteq \Sigma^*$ *equal up to length* ℓ (and write $K \equiv_\ell L$) if K and L coincide on all words of length at most ℓ.

($M4$) Let L be a language and (L_n) a sequence of languages over Σ.
Then, $L_n \equiv_n L$ for all $n \geq 0$ implies $L_n \xrightarrow{d} L$.

If ($M4$) is satisfied for d, then the computation problem for finite assignments has the same solution as for arbitrary assignments.

Lemma 3. *Let d be a language distance satisfying ($M4$) and ϕ, ψ language expressions. Then,*

$$\inf_{\sigma \in finAss} v_d(\sigma, \phi, \psi) = \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi).$$

Proof. Obviously,

$$\inf_{\sigma \in finAss} v_d(\sigma, \phi, \psi) \geqslant \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi).$$

Set $p = \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi)$. This means that there exists a sequence of assignments $\sigma_1, \sigma_2, \ldots$, s.t. $v_d(\sigma_n, \phi, \psi) \xrightarrow{d_\infty} p$. By ($M4$), for each σ_i, there exists a sequence of finite assignments

$\sigma_i^{(1)}, \sigma_i^{(2)}, \ldots$, s.t. $v_d(\sigma_i^{(n)}, \phi, \psi) \xrightarrow{d_\infty} v_d(\sigma_i, \phi, \psi)$ We will construct a sequence of finite assignments $\tau_1, \tau_2, \ldots$, s.t. $v_d(\sigma_i^{(n)}, \phi, \psi) \xrightarrow{d_\infty} p$. This implies that $\inf_{\sigma \in finAss} v_d(\sigma, \phi, \psi) \leqslant p$, and the proof is complete.

By definition of convergence, we have that for every $n \in \mathbb{N}$:

- there exists σ_{i_n} s.t. $d_\infty(v_d(\sigma_{i_n}, \phi, \psi), p) < \frac{1}{2n}$,
- there exists $\sigma_{i_n}^{j_n}$ s.t. $d_\infty(v_d(\sigma_{i_n}^{j_n}, \phi, \psi), v_d(\sigma_{i_n}, \phi, \psi)) < \frac{1}{2n}$.

Thus, by the triangle inequality, we get that $d_\infty(v_d(\sigma_{i_n}^{j_n}, \phi, \psi), p) < \frac{1}{n}$. Set $\tau_n = \sigma_{i_n}^{j_n}$ and we have the required sequence. □

Before showing that language distances can be used to construct concept distances, we give two concrete examples of language distances satisfying $(M4)$.

Two language distances satisfying $(M4)$

The following two mappings from $2^{\Sigma^*} \times 2^{\Sigma^*}$ to $[0, \infty)$ are defined by looking at the words in the symmetric difference $K \triangle L := (K \setminus L) \cup (L \setminus K)$ of the languages K and L:

$$d_1(K, L) := 2^{-\ell} \qquad \text{where } \ell = \min\{|w| \mid w \in K \triangle L\},^1$$
$$d_2(K, L) := \mu(K \triangle L) \quad \text{where } \mu(M) = \tfrac{1}{2} \textstyle\sum_{w \in M} (2|\Sigma|)^{-|w|}.$$

The intuition underlying both functions is that differences between the two languages are less important if they occur for longer words. The first function considers only the length ℓ of the shortest word for which such a difference occurs and yields $2^{-\ell}$ as distance, which becomes smaller if ℓ gets larger. The second function also takes into account how many such differences there are, but differences for longer words count less than differences for shorter ones. More precisely, a difference for the word u counts as much as the sum of all differences for words uv properly extending u.

Now we show that these functions satisfy the required properties.

Lemma 4. *The functions d_1, d_2 are language distances satisfying $(M4)$.*

Proof. In order to show that they are language distances, we have to show that they satisfy $(M1)$, $(M2)$ and $(M3)$. $(M1)$ and $(M2)$ are obvious. Regarding the triangle inequality $(M3)$ $d(K, L) \leqslant d(K, M) + d(M, L)$:
Initially note that for languages K, L, M it holds that $K \triangle L \subseteq K \triangle M \cup M \triangle L$. Indeed, if $x \in K \setminus L$ then either $x \notin M$, implying $x \in K \triangle M$, or $x \in M$, implying $x \in M \triangle L$.

For d_1, if $d_1(K, L) = 0$ then $(M3)$ holds trivially. Suppose that $d_1(K, L) = 2^{-n}$, where $n = |w|$ for some $w \in K \triangle L$. Without loss of generality, suppose that $w \in K \triangle M$. Thus $n \geqslant \min\{\, |w| : w \in K \triangle M \,\}$ and consequently $d_1(K, M) \geqslant 2^{-n}$. Consequently, $d_1(K, L) \leqslant d_1(K, M) + d_1(M, L)$.

For d_2, since $K \triangle L \subseteq K \triangle M \cup M \triangle L$, summing over these sets, we get that

$$\sum_{w \in K \triangle L} (2|\Sigma|)^{-|w|} \leqslant \sum_{w \in K \triangle M \cup M \triangle L} (2|\Sigma|)^{-|w|} \leqslant \sum_{w \in K \triangle M} (2|\Sigma|)^{-|w|} + \sum_{w \in M \triangle L} (2|\Sigma|)^{-|w|}$$

[1] As usual, we assume that $\min \emptyset = \infty$ and $2^{-\infty} = 0$.

and thus $d_2(K,L) \leqslant d_2(K,M) + d_2(M,L)$.

For requirement $(M4)$:

The assumption that $L_n \equiv L \mod \Sigma^{\leqslant n}$ implies that $d_1(L_n,L) \leqslant 2^{-(n+1)}$ and $d_2(L_n,L) = \frac{1}{2}\sum_{w\in L_n \triangle L}(2|\Sigma|)^{-|w|} \leqslant \frac{1}{2}\sum_{w\in\Sigma^*\setminus\Sigma^{\leqslant n}}(2|\Sigma|)^{-|w|} = \frac{1}{2^{n+1}}$. Thus $L_n \xrightarrow{d_1} L$ and also $L_n \xrightarrow{d_2} L$. □

From language distances to concept distances

Based on the normal form of $\mathcal{FL}_0$ concept descriptions introduced in Section 3, we can use a language distance d to define a concept distance. Basically, given $\mathcal{FL}_0$ concept descriptions $C = \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k$ and $D = \forall M_1.A_1 \sqcap \ldots \sqcap \forall M_k.A_k$ in normal form, we can use the distances $e_i = d(L_i,M_i)$ to define a distance between C and D. For this, we need an appropriate function that combines the k values $e_1,\ldots,e_k$ into a single value. We say that the function $f : [0,\infty)^k \to [0,\infty)$ is a *combining function* if it is

- commutative: $f(a_1,\ldots,a_k) = f(a_{\pi(1)},\ldots,a_{\pi(k)})$ for all permutations π of the indices $1,\ldots,k$,
- monotone: $a_1 \leqslant b_1,\ldots,a_k \leqslant b_k \implies f(a_1,\ldots,a_k) \leqslant f(b_1,\ldots,b_k)$,
- zero closed: $f(a_1,\ldots,a_k) = 0 \iff a_1 = \cdots = a_k = 0$,
- and continuous.

The following are simple examples of combining functions:

- $\max(a_1,\ldots,a_k)$,
- $\mathrm{sum}(a_1,\ldots,a_k) = a_1 + \cdots + a_k$,
- $\mathrm{avg}(a_1,\ldots,a_k) = \sum_{i=1}^k a_i/k$.

Given a language distance d and a combining function f, the *concept distance $m_{d,f}$ induced by d,f* is defined as follows. If C,D are $\mathcal{FL}_0$ concept descriptions with normal forms $C \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k$ and $D \equiv \forall M_1.A_1 \sqcap \ldots \sqcap \forall M_k.A_k$, then we set

$$m_{d,f}(C,D) := f\big(d(L_1,M_1),\ldots,d(L_k,M_k)\big).$$

Using one of the language distances d_1,d_2 introduced above in this setting means that differences between the concepts C,D at larger role depth count less than differences at smaller role depth.

Lemma 5. *Let d be a language distance and f be a combining function. Then the concept distance induced by f,d is indeed a concept distance, i.e., it is equivalence closed, symmetric, and equivalence invariant.*

Proof. Let $C = \forall L_1.A_1 \sqcap \cdots \sqcap \forall L_k.A_k$, $D = \forall M_1.A_1 \sqcap \cdots \sqcap \forall M_k.A_k$ and $E = \forall N_1.A_1 \sqcap \cdots \sqcap \forall N_k.A_k$ be concept descriptions. All properties can be proved with simple computations.
Equivalence closedness:

$$\begin{aligned} m_{d,f}(C,D) = 0 &\iff f(d(L_1,M_1),\ldots,d(L_k,M_k)) = 0 \\ &\iff d(L_1,M_1) = \cdots = d(L_k,M_k)) = 0 \\ &\iff L_1 = M_1,\ldots,L_k = M_k \iff C \equiv D \end{aligned}$$

Symmetry:

$$\begin{aligned} m_{d,f}(C,D) &= f(d(L_1,M_1),\ldots,d(L_k,M_k)) \\ &= f(d(M_1,L_1),\ldots,d(M_k,L_k)) = m_{d,f}(D,C) \end{aligned}$$

Equivalence invariance:
$C \equiv D \overset{Lemma\ 1}{\Longrightarrow} L_i = M_i$ for $i = 1, \ldots, k$

$$\begin{aligned} m_{d,f}(C, E) &= f(d(L_1, W_1), \ldots, d(L_k, W_k)) \\ &= f(d(M_1, W_1), \ldots, d(M_k, W_k)) = m_{d,f}(D, E) \end{aligned}$$

□

Reducing approximate unification to approximately solving language equations

In the following, we assume that d is a language distance, f a combining function, and $m_{d,f}$ the concept distance induced by f, d. Let C, D be $\mathcal{FL}_0$ concept patterns in normal form, as shown in (6), and (7) the corresponding language equations, for $i = 1, \ldots, k$. We denote the left- and right-hand sides of the equations (7) with ϕ_i and ψ_i, respectively. The following lemma shows that the degree of violation transfers from finite assignments $\sigma_1, \ldots, \sigma_k$ to the induced substitution $\rho(\sigma_1, \ldots, \sigma_k)$ as defined in (8).

Lemma 6. *Let $\sigma_1, \ldots, \sigma_k$ be finite assignments. Then $f(v_d(\sigma_1, \phi_1, \psi_1), \ldots, v_d(\sigma_k, \phi_k, \psi_k)) = v_{m_{d,f}}(\rho(\sigma_1, \ldots, \sigma_k), C, D)$.*

Proof. Consider the concept patterns

$$\begin{aligned} C &= \forall S_{0,1}.A_1 \sqcap \ldots \sqcap \forall S_{0,k}.A_k \sqcap \forall S_1.X_1 \sqcap \ldots \sqcap \forall S_n.X_n \\ D &= \forall T_{0,1}.A_1 \sqcap \ldots \sqcap \forall T_{0,k}.A_k \sqcap \forall T_1.X_1 \sqcap \ldots \sqcap \forall T_n.X_n \end{aligned}$$

Set $L_{i,j} := \sigma_j(X_i)$ for $i = 1, \ldots, n,\ j = 1, \ldots, k$. Recall that ρ is the bijection between tuples of assignments and substitutions described in 8, and in the following abbreviate $\rho(\sigma_1, \ldots, \sigma_k)$ by σ.

Then we have that

$$\begin{aligned} \sigma(C) &= \prod_{i=1}^{k} \forall(S_{0,i} \cup S_1 L_{1,i} \cup \cdots \cup S_n L_{n,i}) A_i \\ \sigma(D) &= \prod_{i=1}^{k} \forall(T_{0,i} \cup T_1 L_{1,i} \cup \cdots \cup T_n L_{n,i}) A_i \end{aligned}$$

and

$$\begin{aligned} \sigma_i(\varphi_i) =& S_{0,i} \cup S_1 L_{1,i} \cup \cdots \cup S_n L_{n,i} \\ \sigma_i(\psi_i) =& T_{0,i} \cup T_1 L_{1,i} \cup \cdots \cup T_n L_{n,i}. \end{aligned}$$

Thus

$$\begin{aligned} v_{m_{d,f}}(\sigma, C, D) =& m_{d,f}(\sigma(C), \sigma(D)) \\ =& f(d(S_{0,1} \cup S_1 L_{1,1} \cup \cdots \cup S_n L_{n,1}, T_{0,1} \cup T_1 L_{1,1} \cup \cdots \cup T_n L_{n,1}), \\ & \ldots, d(S_{0,k} \cup S_1 L_{1,k} \cup \cdots \cup S_n L_{n,k}, T_{0,k} \cup T_1 L_{1,k} \cup \cdots \cup T_n L_{n,k})) \\ =& f(d(\sigma_1(\varphi_1), \sigma_1(\psi_1)), \ldots, d(\sigma_k(\varphi_k), \sigma_k(\psi_k))) \\ =& f(v_d(\sigma_1, \varphi_1, \psi_1), \ldots, v_d(\sigma_k, \varphi_k, \psi_k)) \end{aligned}$$

□

Since the combining function is continuous, the equality stated in this lemma is preserved under building the infimum. In addition, Lemma 3 shows that the restriction to finite assignments can be dispensed with if d satisfies $(M4)$.

Lemma 7. *Assume that d satisfies $(M4)$. Then,*

$$\begin{aligned}\inf_{\sigma \in Sub} v_{m_{d,f}}(\sigma, C, D) = \\ = f\left(\inf_{\sigma_1 \in finAss} v_d(\sigma_1, \phi_1, \psi_1), \ldots, \inf_{\sigma_k \in finAss} v_d(\sigma_k, \phi_k, \psi_k)\right) \\ = f\left(\inf_{\sigma_1 \in Ass} v_d(\sigma_1, \phi_1, \psi_1), \ldots, \inf_{\sigma_k \in Ass} v_d(\sigma_k, \phi_k, \psi_k)\right).\end{aligned}$$

In case f is computable (in polynomial time), this lemma yields a (polynomial time) reduction of the computation problem for approximate $\mathcal{FL}_0$ unification to the computation problem for approximately solving language equations. In addition, we know that the decision problem can be reduced to the computation problem. Thus, it is sufficient to devise a procedure for the computation problem for approximately solving language equations.

In our example, the normal forms of the abbreviated concept descriptions (1) and (3) are

$$\forall\{\varepsilon\}.A \sqcap \forall\{rs\}.B \sqcap \forall\emptyset.D \sqcap \forall\emptyset.E \sqcap \forall\{r\}.X,$$
$$\forall\{\varepsilon\}.A \sqcap \forall\emptyset.B \sqcap \forall\{r\}.D \sqcap \forall\{rt\}.E \sqcap \forall\emptyset.X.$$

It is easy to see that the language equations for the concept names A, D, E are solvable, and thus these solutions contribute distance 0 to the overall concept distance. The language equation for the concept name B is $\{rs\} \cup \{r\} \cdot X_B = \emptyset \cup \emptyset \cdot X_B$, and the assignment $X_B = \emptyset$ leads to the smallest possible symmetric difference $\{rs\}$, which w.r.t. d_1 yields the value $2^{-2} = 1/4$. It is easy to see that this is actually the infimum for this equation. If we use the combining function avg, then this gives us the infimum $1/16$ for our approximate unification problem.

4 Approximately solving language equations

In the following, we show how to solve the computation problem for the language distances d_1 and d_2 introduced above. Our solution uses the automata-based approach for solving language equations introduced in [5].

The first step in this approach is to transform the given system of language equations into a single equation of the form $\phi = \emptyset$ such that the language expression ϕ is *normalized* in the sense that all constant languages L occurring in ϕ are singleton languages $\{a\}$ for $a \in \Sigma \cup \{\varepsilon\}$. This normalization step can easily be adapted to approximate equations, but in addition to a normalized approximate equation $\phi_a \approx \emptyset$ it also generates a normalized strict equation $\phi_s = \emptyset$.

Lemma 8. *Let ϕ, ψ be language expressions. Then we can compute in polynomial time normalized language expressions ϕ_a and ϕ_s such that the following holds for $d \in \{d_1, d_2\}$:*

$$\{v_d(\sigma, \phi, \psi) \mid \sigma \in Ass\} = \{v_d(\sigma, \phi_a, \emptyset) \mid \sigma \in Ass \wedge \sigma(\phi_s) = \emptyset\}.$$

Proof. In [5] (Lemma 1) it is shown how a given system of language equations can be transformed into a single normalized language equation such that there is a one-to-one correspondence between the solutions of the original system and the solutions of the normal form. Given an approximate equation $\phi \approx \psi$, we first abstract the left- and right-hand side of this equation

with new variables X, Y, and add strict equations that say that X must be equal to ϕ and Y must be equal to ψ, i.e., we consider the approximate equation $X \approx Y$ together with the strict equations $X = \phi$ and $Y = \psi$. We then apply the normalization approach of [5] to the two strict equations. Basically, this approach introduces new variables and equations that express the regular languages occurring in ϕ and ψ, using the fact that regular languages can be expressed as unique solutions of language equations in solved form (see [5] for details). The resulting system of strict equations can then be expressed by a single strict equation $\phi_s = \emptyset$ using the facts that

- $M = N$ iff $M \triangle N = \emptyset$;
- $M = \emptyset$ and $N = \emptyset$ iff $M \cup N = \emptyset$.

Though it is not explicitely stated in [5], it is easy to see that this transformation is such that, for any assignment σ, there is a solution θ of $\phi_s = \emptyset$ such that $\theta(X) = \sigma(\psi)$ and $\theta(Y) = \sigma(\psi)$. Conversely, any solution θ of $\phi_s = \emptyset$ satisfies $\theta(X) = \sigma(\psi)$ and $\theta(Y) = \theta(\psi)$. Consequently, we have

$$\{(\sigma(\phi), \sigma(\psi)) \mid \sigma \in Ass\} = \{(\theta(X), \theta(Y)) \mid \theta \in Ass \wedge \theta(\phi_s) = \emptyset\}.$$

If we now define $\phi_a := X \triangle Y$, then the lemma is an easy consequence of the above identity and the fact that both d_1 and d_2 consider the symmetric difference of the input languages. □

This lemma shows that, to solve the computation problem for $\phi \approx \psi$, we must solve the computation problem for $\phi_a \approx \emptyset$, but restrict the infimum to assignments that solve the strict equation $\phi_s = \emptyset$.

In a second step, [5] shows how a normalized language equation can be translated into a tree automaton working on the infinite, unlabeled n-ary tree (where $n = |\Sigma|$). The nodes of this tree can obviously be identified with Σ^*. The automata considered in [5] are such that the state in each successor of a node is determined independently of the choice of the states in its siblings. These automata are called *looping tree automata with independent transitions (ILTA)*.

Definition 6. *An ILTA is of the form $A = (\Sigma, Q, Q_0, \delta)$, where Σ is a finite alphabet, Q is a finite set of states, with initial states $Q_0 \subseteq Q$, and $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function that defines possible successors of a state for each $a \in \Sigma$. A* run *of this ILTA is any function $r\colon \Sigma^* \to Q$ with $r(\varepsilon) \in Q_0$ and $r(wa) \in \delta(r(w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$.*

According to this definition, ILTAs do not have a fixed set of final states. However, by choosing any set of states $F \subseteq Q$, we can use runs r of A to define languages over Σ as follows: $L_r(A, F) := \{w \in \Sigma^* \mid r(w) \in F\}$.

Given a normalized language equation $\phi = \emptyset$ with variables $\{X_1, \ldots, X_m\}$, it is shown in [5] how to construct an ILTA $A^\phi = (\Sigma, Q^\phi, Q_0^\phi, \delta^\phi)$ and subsets $F, F_1, \ldots, F_m \subseteq Q^\phi$ such that the following holds:

Proposition 1. *If r is a run of A^ϕ, then the* induced assignment *σ_r with $\sigma_r(X_i) := L_r(A^\phi, F_i)$, for $i = 1, \ldots, m$, satisfies $\sigma_r(\phi) = L_r(A^\phi, F)$. In addition, every assignment is induced by some run of A_ϕ.*

The size of this ILTA is exponential in the size of ϕ. In order to decide whether the language equation $\phi = \emptyset$ has a solution, one thus needs to decide whether A^ϕ has a run in which no state of F occurs. This can easily be done by removing all states of F from A^ϕ, and then checking the resulting automaton A^ϕ_{-F} for emptiness. In fact, as an easy consequence of the above proposition we obtain that there is a 1–1-correspondence between the runs of A^ϕ_{-F} and the solutions of $\phi = \emptyset$ (Proposition 2 in [5]).

This approach can easily be adapted to the situation where we have an approximate equation $\phi_a \approx \emptyset$ and a strict equation $\phi_s = \emptyset$. Basically, we apply the construction of [5] to $\phi_a \cup \phi_s$, but instead of one set of states F we construct two sets F_a and F_s such that $\sigma_r(\phi_a) = L_r(A^{\phi_a \cup \phi_s}, F_a)$ and $\sigma_r(\phi_s) = L_r(A^{\phi_a \cup \phi_s}, F_s)$ holds for all runs r of $A^{\phi_a \cup \phi_s}$. By removing all states of F_s from $A^{\phi_a \cup \phi_s}$, we obtain an automaton whose runs are in 1–1-correspondence with the assignments that solve $\phi_s = \emptyset$. In addition, we can make this automaton *trim*[2] using the polytime construction in the proof of Lemma 2 in [5].

Theorem 2. *Given an approximate equation $\phi_a \approx \emptyset$ and a strict equation $\phi_s = \emptyset$, we can construct in exponential time a trim ILTA $A = (\Sigma, Q, Q_0, \delta)$ and sets of states $F_a, F_1, \ldots, F_m \subseteq Q$ such that every run r of A satisfies $\sigma_r(\phi_a) = L_r(A, F_a)$ and $\sigma_r(\phi_s) = \emptyset$. In addition, every assignment σ with $\sigma(\phi_s) = \emptyset$ is induced by some run of A.*

The measure d_1

Using Lemma 8, Theorem 2, and the definition of d_1, it is easy to see that the computation problem for an approximate language equation $\phi \approx \psi$ can be reduced to solving the following problem for the trim ILTA $A = (\Sigma, Q, Q_0, \delta)$ of Theorem 2: compute $\sup_{r \text{ run of } A} \min\{|w| \mid r(w) \in F_a\}$. More formally, we have the following lemma.

Lemma 9. *If $\ell = \sup_{r \text{ run of } A} \min\{|w| \mid r(w) \in F_a\}$ then $\inf_{\sigma \in Ass} v_{d_1}(\sigma, \phi, \psi) = 2^{-\ell}$.*

In order to compute this supremum, it is sufficient to compute, for every state $q \in Q$, the *length $lpr(q)$ of the longest partial run of A starting with q that does not have states of F_a at non-leaf nodes*. More formally, we define:

Definition 7. *Let $\Sigma^{\leqslant \ell}$ denote the set of all words over Σ of length at most ℓ. Given a trim ILTA $A = (\Sigma, Q, Q_0, \delta)$, a* partial run of A of length ℓ from a state $q \in Q$ *is a mapping $p : \Sigma^{\leqslant \ell} \to Q$ such that $p(\varepsilon) = q$ and $p(wa) \in \delta(p(w), a)$ for all $w \in \Sigma^{\leqslant \ell - 1}$ and $a \in \Sigma$. The* leaves *of p are the words of length ℓ. Finally, for every $q \in Q$ we have that*

$$lpr(q) := \sup\{\ell \mid \exists p : \Sigma^{\leqslant \ell} \to Q \text{ s.t. } p(\epsilon) = q \wedge \forall w \in \Sigma^{\leqslant \ell - 1}(p(wa) \in \delta(p(w), a) \wedge p(w) \notin F_a)\}.$$

Lemma 10. *The function $lpr : Q \to \mathbb{N} \cup \{\infty\}$ can be computed in time polynomial in the size of A.*

Proof. In order to compute *lpr*, we use an iteration similar to the emptiness test for looping tree automata [6].

If $q \in F_a$, then clearly $lpr(q) = 0$ and otherwise q has an appropriate partial run of length > 0 (recall that A is trim). For this reason, we start the iteration with

$$Q^{(0)} := F_a.$$

Next, for $i \geq 0$, we define

$$Q^{(i+1)} := Q^{(i)} \cup \{q \in Q \mid \exists a \in \Sigma : \delta(q, a) \subseteq Q^{(i)}\}.$$

We have $Q^{(0)} \subseteq Q^{(1)} \subseteq Q^{(2)} \subseteq \ldots \subseteq Q$. Since Q is finite, there is an index $j \leq |Q|$ such that $Q^{(j)} = Q^{(j+1)}$, and thus the iteration becomes stable.

[2] An ILTA (Σ, Q, Q_0, δ) is *trim* if every state is reachable from an initial state and $\delta(q, a) \neq \emptyset$ for all $q \in Q, a \in \Sigma$.

It is easy to show that

$$lpr(q) = \begin{cases} \min\{i \mid q \in Q^{(i)}\} & \text{if } q \in Q^{(j)} \\ \infty & \text{if } q \notin Q^{(j)} \end{cases}$$

To prove the above, the following claim is enough.

Claim. It holds that $q \notin Q^{(i)}$ iff there is a partial run of length $i+1$ of A starting with q that does not have states of F_a at non-leaf nodes.

Proof (Claim). By induction on i:
For $i = 0$, $q \notin Q^{(i)}$ iff $q \notin F_a$ iff there is a partial run of length 1 of A starting with q that does not have states of F_a at non-leaf nodes (i.e. at the root).
For $i \geqslant 1$, if $q \notin Q^{(i)}$ then for every $a \in \Sigma$, it holds that $\delta(q,a) \not\subseteq Q^{(i-1)}$, i.e., for every $a \in \Sigma$ there exists a $q_a \in \delta(q,a) \setminus Q^{(i-1)}$. By the induction hypothesis, for every such q_a there is a partial run of length i of A starting with q_a that does not have states of F_a at non-leaf nodes, thus we can construct such a run of length $i+1$ for q. If $q \in Q^{(i)}$, then there exists an $a \in \Sigma$, s.t. $\delta(q,a) \subseteq Q^{(i-1)}$. By the induction hypothesis, there is no partial run of length i of A starting with p for any $p \in \delta(q,a)$ that does not have states of F_a at non-leaf nodes. Thus, there is no such run of lenth $i+1$ starting with q, and this completes the proof of the claim.

If $q \notin Q^{(j)}$, note that the claim implies that there are such runs for every $n \in \mathbb{N}$, and thus $lpr(q) = \infty$.

Since the number of iterations is linear in $|Q|$ and every iteration step can obviously be performed in polynomial time, this completes the proof. □

The function *lpr* can now be used to solve the computation problem as follows:

$$\sup_{r \text{ run of } A} \min\{|w| \mid r(w) \in F_a\} = \max\{lpr(q) \mid q \in Q_0\}.$$

If this maximum is ∞, then the measure d_1 yields value 0 and the approximate equation was actually solvable as a strict one.

Theorem 3. *For the distance d_1 and a polytime computable combining function, the computation problem (for approximate $\mathcal{FL}_0$ unification and for approximately solving language equations) can be solved in exponential time, and the decision problem is ExpTime-complete.*

Proof. The ExpTime-upper bounds follow from our reductions and the fact that the automaton A can be computed in exponential time and is thus of at most exponential size. Hardness can be shown by a reduction of the strict problems, which are known to be ExpTime-complete [4,5]. In fact, the proof of Lemma 10 shows that d_1 either yields the value $0 = 2^{-\infty}$ (in which case the strict equation is solvable) or a value larger than $2^{-(|Q|+1)}$ (in which case the strict equation is not solvable). In other words, for a threshold smaller than $2^{-(|Q|+1)}$ the decision problem is equivalent to the classical solvability problem. □

The measure d_2

Recall that the value of d_2 is obtained by applying the function μ to the symmetric difference of the input languages. In case one of the two languages is empty, its value is thus obtained by applying μ to the other language. It is easy to show that the following lemma holds.

Lemma 11. *The value $\mu(L)$ for $L \subseteq \Sigma^*$ satisfies the recursive equation:*

$$\mu(L) = \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(a^{-1}L), \tag{10}$$

where $a^{-1}L := \{w \in \Sigma^* \mid aw \in L\}$ and χ_L is the characteristic function of the language L.

Proof.

$$\begin{aligned}
\mu(L) &= \frac{1}{2}\sum_{w\in L}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}(2|\Sigma|)^{-|\epsilon|}\chi_L(\epsilon) + \frac{1}{2}\sum_{w\in L\setminus\{\epsilon\}}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}\chi_L(\epsilon) + \frac{1}{2}\sum_{a\in\Sigma}\sum_{aw\in L}(2|\Sigma|)^{-(1+|w|)} \\
&= \frac{1}{2}\chi_L(\epsilon) + \frac{1}{2}\sum_{a\in\Sigma}\sum_{w\in a^{-1}L}(2|\Sigma|)^{-1}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}\chi_L(\epsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\frac{1}{2}\sum_{w\in a^{-1}L}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}\chi_L(\epsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(a^{-1}L)
\end{aligned}$$

□

Using Lemma 8, Theorem 2, and the definition of d_2, it is easy to see that the computation problem for an approximate language equation $\phi \approx \psi$ w.r.t. d_2 can be reduced to solving the following problem for the trim ILTA $A = (\Sigma, Q, Q_0, \delta)$ of Theorem 2: compute $\inf_{r\ run\ of\ A}\mu(L_r(A, F_a))$. In fact, this is the exact value that answers the computation problem, as can be seen from the following lemma.

Lemma 12. *It holds that*

$$\inf_{r\ run\ of\ A}\mu(L_r(A, F_a)) = \inf_{\sigma\in Ass} v_{d_2}(\sigma, \phi, \psi).$$

In the following, we are only interested in languages defined by runs of A with set of final states F_a, thus for ease of notation we will write L_r instead of $L_r(A, F_a)$.

Using (10), we now show that this infimum can be computed by solving a system of recursive equations that is induced by the transitions of A. Given an arbitrary (not necessarily initial) state $q \in Q$, we say that $r\colon \Sigma^* \to Q$ is a *q-run of A* if $r(\varepsilon) = q$ and $r(wa) \in \delta(r(w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. We denote the set of all q-runs of A with $R_A(q)$. Since each run of A is a q_0-run for some $q_0 \in Q_0$, we have

$$\inf_{r\ run\ of\ A}\mu(L_r) = \min_{q_0\in Q_0}\inf_{r\in R_A(q_0)}\mu(L_r).$$

For all $q \in Q$, we define $\mu(q) := \inf_{r\in R_A(q)}\mu(L_r)$. The identity above shows that we can solve the computation problem for approximate language equations w.r.t. d_2 if we can devise a procedure for computing the values $\mu(q) \in \mathbb{R}$ for all $q \in Q$. The identity (10) can now be used to show the following lemma.

Lemma 13. *For all states $q \in Q$ we have*

$$\mu(q) = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{p\in\delta(q,a)}\mu(p),$$

where χ_{F_a} denotes the characteristic function of the set F_a.

Proof. Given a run $r \in R_A(q)$, for every $a \in \Sigma$ a unique run $r_a \in R_A(r(a))$ is defined as $r_a(w) = r(aw)$. Obviously, $r(a) \in \delta(q, a)$. Conversely, given a run $r_a \in R_A(q_a)$, for every $a \in \Sigma$, such that $q_a \in \delta(q, a)$, a unique run $r_0 \in R_A(q)$ can be derived, as

$$r_0(w) = \begin{cases} q & \text{if } w = \epsilon \\ r_a(u) & \text{if } w = au \end{cases}$$

Hence, there is a bijection between the set of q-runs $R_A(q)$ and the set of "successor" runs $SR(q) := \bigcup_{p_1\in\delta(q,a_1)} R_A(p_1) \times \cdots \times \bigcup_{p_n\in\delta(q,a_n)} R_A(p_n)$.

Given a run $r \in R(q)$, it holds that

$$L_r = \epsilon(q) \cup \bigcup_{a\in\Sigma} aL_{r_a}$$

where

$$\epsilon(q) = \begin{cases} \{\epsilon\} & \text{if } q \in F_a \\ \emptyset & \text{otherwise} \end{cases}$$

For the measure μ and disjoint sets of words A and B it holds that $\mu(A\dot\cup B) = \mu(A) + \mu(B)$. Additionally, for $a \in \Sigma$ and $A \subseteq \Sigma^*$, it holds $\mu(aA) = \frac{1}{2|\Sigma|}\mu(A)$.

Thus, given a run $r \in R_A(q)$ it holds

$$\begin{aligned} \mu(L_r) &= \mu(\epsilon(q) \cup \bigcup_{a\in\Sigma} aL_{r_a}) \\ &= \mu(\epsilon(q)) + \sum_{a\in\Sigma}\mu(aL_{r_a}) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(L_{r_a}). \end{aligned}$$

Thus it can be inferred that

$$\begin{aligned}
\mu(q) &= \inf_{r \in R_A(q)} \mu(L_{r_q}) \\
&= \inf_{r \in R_A(q)} \left(\mu(\epsilon \cdot \chi_{F_a}(q)) + \frac{1}{2|\Sigma|} \sum_{a \in \Sigma} \mu(L_{r_a}) \right) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \inf_{r \in R_A(q)} \sum_{i=1}^{n} \mu(L_{r_{a_i}}) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \inf_{(r_{a_1},\ldots,r_{a_n}) \in SR(q)} \sum_{i=1}^{n} \mu(L_{r_{a_i}}) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \min_{\substack{(p^1,\ldots,p^n) \in \\ \delta(q,a_1)\times\cdots\times\delta(q,a_n)}} \inf_{\substack{(r_1,\ldots,r_n) \in \\ R(p^1)\times\cdots\times R(p^n)}} \sum_{i=1}^{n} \mu(L_{r_i}) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \min_{\substack{(p^1,\ldots,p^n) \in \\ \delta(q,a_1)\times\cdots\times\delta(q,a_n)}} \sum_{i=1}^{n} \inf_{r \in R(p^i)} \mu(L_r) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \sum_{i=1}^{n} \min_{p \in \delta(q,a_i)} \inf_{r \in R(p)} \mu(L_r) \\
&= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \sum_{a \in \Sigma} \min_{p \in \delta(q,a)} \mu(p).
\end{aligned}$$

□

By introducing variables x_q (for $q \in Q$) that range over $\mathbb{R}$, we can rephrase this lemma by saying that the values $\mu(q)$ yield a solution to the system of equations

$$x_q = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \sum_{a \in \Sigma} \min_{p \in \delta(q,a)} x_p \quad (q \in Q). \tag{11}$$

Thus, to compute the values $\mu(q)$ for $q \in Q$ it is sufficient to compute a solution of (11).

Next, we use Banach's fixed point theorem to show that the system has a *unique* solution in $\mathbb{R}$. In particular, we will transform the system of equations to a contraction in $\mathbb{R}^k$ that has as a fixed point the solution of (11).

For every equation (11) corresponding to a state $q_i \in Q$, we represent the right-hand side as a function $f_i \colon \mathbb{R}^k \to \mathbb{R}$, defined by

$$f_i(x_1, \ldots, x_k) = \frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|} \sum_{a \in \Sigma} \min_{q_j \in \delta(q_i,a)} x_j$$

Next, the right-hand sides of the entire system are denoted by a vector function $f \colon \mathbb{R}^k \to \mathbb{R}^k$, with

$$f(x_1, \ldots, x_k) = (f_1(x_1, \ldots, x_k), \ldots, f_k(x_1, \ldots, x_k))$$

Proposition 2. *A vector* $\mathbf{x} = (x_1, \ldots, x_k)$ *is a solution of the system of equations of the form* (11) *iff it is a fixed point of* f.

Before proving that f is a contraction, we provide a technical lemma that will be useful in the proof of the next one.

Lemma 14. *Given a finite set of indices I and a set $\{J(i) \subseteq I \mid i \in I\}$, it holds that*

$$\max_{i\in I}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| \leqslant \max_{i\in I}|x_i - y_i|\,.$$

Proof. For every $i \in I$ we have

$$\min_{j\in J(i)} x_j = x_{k_i} \text{ for some } k_i \in J(i)$$
$$\min_{j\in J(i)} y_j = y_{\ell_i} \text{ for some } \ell_i \in J(i).$$

Then, for every $i \in I$ we get

$$\begin{aligned}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| &= |x_{k_i} - y_{\ell_i}|\\ &\overset{(*)}{=} x_{k_i} - y_{\ell_i}\\ &\leqslant x_{\ell_i} - y_{\ell_i}\\ &\leqslant \max_{i\in I}|x_i - y_i|.\end{aligned}$$

For equality $(*)$ suppose without loss of generality, $x_{k_i} \geqslant y_{\ell_i}$. If $x_{k_i} \leqslant y_{\ell_i}$, the exact symmetric argument can be used.

Finally, we have that

$$\begin{aligned}\max_{i\in I}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| &\leqslant \max_{i\in I}\max_{i\in I}|x_i - y_i|\\ &= \max_{i\in I}|x_i - y_i|.\end{aligned}$$

□

The following lemma provides the last condition for Theorem 1.

Lemma 15. *The function f defined above is a contraction in $(\mathbb{R}^k, d_\infty)$.*

Proof. Let $\mathbf{x} = (x_1, \ldots, x_k),\ \mathbf{y} = (y_1, \ldots, y_k) \in \mathbb{R}^k$. Then

$$\begin{aligned}d_\infty(f(\mathbf{x}),f(\mathbf{y})) &= \max_{i=1,\ldots,k}|f_i(\mathbf{x}) - f_i(\mathbf{y})|\\ &= \max_{i=1,\ldots,k}\left|\frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{q_j\in\delta(q_i,a)} x_j - \left(\frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &= \max_{i=1,\ldots,k}\left|\frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\left(\min_{q_j\in\delta(q_i,a)} x_j - \min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &\leqslant \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\max_{i=1,\ldots,k}\left|\left(\min_{q_j\in\delta(q_i,a)} x_j - \min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &\overset{(*)}{\leqslant} \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\max_{i=1,\ldots,k}|x_i - y_i|\\ &= \frac{1}{2|\Sigma|}|\Sigma|\max_{i=1,\ldots,k}|x_i - y_i|\\ &= \frac{1}{2}\max_{i=1,\ldots,k}|x_i - y_i|\end{aligned}$$

where inequality $(*)$ holds because of Lemma 14. □

Finally, since $(\mathbb{R}^k, d_\infty)$ is complete, from Theorem 1 and Proposition 2 we get the following.

Lemma 16. *The system of equations* (11) *has a unique solution.*

In order to actually compute this, we can use Linear Programming [15].

A *Linear Programming problem* or *LP problem* is a set of restrictions along with an objective function. In its most general form, an LP problem looks like this:

$$\begin{aligned} \text{objective:}\quad & \min/\max\ z = c_1x_1 + \ldots + c_nx_n \\ \text{restrictions:}\quad & a_{1,1}x_1 + \ldots + a_{1,n}x_n \gtreqless b_1 \\ & \vdots \\ & a_{m,1}x_1 + \ldots + a_{m,n}x_n \gtreqless b_m \end{aligned}$$

where $a_{i,j}, b_i, c_j$ are rational numbers.

The *feasible region* of the LP problem consists of all the tuples $(x_1, \ldots, x_n)$ that satisfy the restrictions. The answer to an LP problem is a tuple in the feasible region that maximizes the objective function and "no" if the feasible region is empty.

It is well known that LP problems are solvable in polynomial time in the size of the problem.[15]

From the above system of equations 11 we can derive an LP problem. The only non-trivial step in this translation is to express the minimum operator. For this, we introduce additional variables $y_{q,a}$, which intuitively stand for $\min_{p\in\delta(q,a)} x_p$. Then (11) is transformed into

$$x_q = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma} y_{q,a} \quad (q \in Q). \tag{12}$$

To express the intuitive meaning of the variables $y_{q,a}$, we add the inequalities

$$y_{q,a} \leq x_p \quad \text{for all } q \in Q \text{ and } p \in \delta(q,a) \tag{13}$$

as well as the objective to maximize the values of these variables:

$$z = \max \sum_{q\in Q}\sum_{a\in\Sigma} y_{q,a}. \tag{14}$$

Lemma 17. *The LP problem consisting of the equations* (12)*, the inequations* (13)*, and the objective* (14) *has the unique solution*

$$\{x_q \mapsto \mu(q) \mid q \in Q\} \cup \{y_{q,a} \mapsto \min_{p\in\delta(q,a)} \mu(p) \mid p \in Q, a \in \Sigma\}.$$

Proof. Initially, observe that the above vector is in the feasible region, since it satisfies the restrictions (12) and (13). Next, we procede to show that it is indeed the only point that maximizes the objective function. First, we need the following claim.

Claim. If $\mathbf{x}$ is a solution that maximizes the objective function then, for every $q \in Q$ and every $a \in \Sigma$, at least one of the inequalities (13) holds as an equality.

Proof (Claim). Suppose on the contrary that $\mathbf{x}$ is a solution that maximizes z, but for some $q \in Q$ and $a \in \Sigma$, inequalities $x_{q,a} \leqslant \mu_p$ are strict for all $p \in \delta(q,a)$. This would mean that the value of $x_{q,a}$ can be increased, until it actually becomes equal to $\min_{p\in\delta(q,a)} \mu_p$, and all inequalities would still hold. The only restriction that would be hurt, is the one of the form (12) for the state q. This can be easily mended by setting μ_q to be equal to the right-hand side. This change will not affect any of the other restrictions. Thus, a new point $\mathbf{x}'$ has been produced, that satisfies all the restrictions of the LPP and additionally gives a larger value for the objective function. This is a contradiction to our initial assertion about $\mathbf{x}$. This completes the proof of the claim.

As a result, any points that are solutions to the LP problem, satisfy the condition $y_{q,a} = \min_{p\in\delta(q,a)} x_p$ for all q, a. Given that they also satisfy the equality constraints (12) (since they are in the feasible region), they correspond to solutions of the system of equations (11).

Finally, since there is a unique such solution, the solution of the LP problem is this unique solution. □

Since LP problems can be solved in polynomial time and the size of the LP problem in the above lemma is polynomial in the size of A, we obtain an ExpTime-upper bound for the computation problem and the decision problem. ExpTime-hardness can again be shown by a reduction of the strict problem.

Even though the main idea is the same, the formal proof is quite more technical in this case. Initially note that solving (11) induces a "best" q-run of the automaton for every q; in every step, pick the state with the minimum value among all possible. Given such a best run of the automaton, we say that p is a *descendant* of q at depth d, if there are states $q_0 := q, q_1, \ldots, q_{d-1}, q_d := p$, and a word $a_1 \ldots, a_d$ s.t. $q_i = \arg\min_{q\in\delta(q_{i-1},a_i)} \mu(q)$ for $i = 1, \ldots, d$. A *bad descendant* of a state q is a state $p \in F_a$ that is a descendant of q. Note that, if $q \in F_a$, then $\mu(q) \geqslant \frac{1}{2}$ and if $q \notin F_a$, then $\mu(q) \leqslant \frac{1}{2}$.

Lemma 18. *For a state q it holds that $\mu(q) > 0$ if and only if q has a bad descendant.*

Proof. If q has a bad descendant p, say at depth d, then there is a branch with nodes labeled $q, q_1, \ldots, p$ and thus $\mu(q) \geqslant \frac{1}{2|\Sigma|}\mu(q_1) \geqslant \cdots \geqslant (\frac{1}{2|\Sigma|})^d \mu(p) \geqslant (\frac{1}{2|\Sigma|})^d \frac{1}{2} > 0$.

Conversely, suppose that q has no bad descendant. Thus $q \notin F_a$ and the same holds for all its descendants. Then it holds that

$$\begin{aligned}\mu(q) &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma} \min_{p\in\delta(q,a)} \mu(p) \\ &\leqslant \frac{1}{2|\Sigma|}\sum_{a\in\Sigma} \max_{a\in\Sigma} \min_{p\in\delta(q,a)} \mu(p) \\ &= \frac{1}{2}\mu(p)\end{aligned}$$

for some child p of q. Iterating this for d steps, we get that $\mu(q) \leqslant (\frac{1}{2})^d \mu(p')$ for some descendant p' of q. But since $p' \notin F_a$, $\mu(p') \leqslant \frac{1}{2}$ and thus $\mu(q) \leqslant (\frac{1}{2})^{d+1}$. Since this holds for every d, it can be concluded that $\mu(q) = 0$.

Lemma 19. *If q has a bad descendant, then q has a bad descendant at depth at most $|Q|$.*

Proof. Set $k = |Q|$. Suppose that there exists a $q_0 \in Q$ with no bad descendants up to depth k. It will be proved that there is a q_0-run with no states from F_a, i.e. q_0 has no bad descendants.

No bad descendants up to depth k implies that there is a partial run of length k of A starting with q_0 that does not have state of F_a. For a branch of length k, the nodes are labelled with states $q_0, q_1, \ldots, q_k$. Since there are only k states, there are indices $i < j \leqslant k$ such that $q_i = q_j$. The tree having as root the node labeled with q_i has bigger length than the one labeled with q_j. Replace the latter tree with the former one. Then, all branches passing through the node labeled with q_j have length at least $k+1$. Iterating this procedure for all branches, a partial run of length at least $k+1$ is derived. Every time the above is repeated, a longer partial run is derived. We conclude that a partial run of infinite length, i.e. a q_0-run of A can be derived that has no states from F_a. Thus q_0 has no bad descendants. □

Lemma 20. *For the distance d_2, the decision problem for approximately solving language equations is ExpTime-hard.*

Proof. If $\mu(q) > 0$, then q has a bad descendant at depth at most $|Q|$. Thus $\mu(q) \geqslant (\frac{1}{2|\Sigma|})^{|Q|} \cdot \frac{1}{2} =: t$. We conclude that the decision problem with threshold t has a positive answer for the equation $\phi \approx \emptyset$ iff the equation $\phi = \emptyset$ has a solution. Since the latter problem is ExpTime-complete, we get an ExpTime-hardness result for our problem as well.

Theorem 4. *For the distance d_2 and a polytime computable combining function, the computation problem (for approximate $\mathcal{FL}_0$ unification and for approximately solving language equations) can be solved in exponential time, and the decision problem is ExpTime-complete.*

For this theorem to hold, the exact definition of the distance d_2 is actually not important. Our approach works as long as the distance induces a system of equations similar to (11) such that Banach's fixed point theorem ensures the existence of a unique solution, which can be found using linear programming.

As an example, we can consider a weighted version of d_2, where different letters have different weights (degrees of importance). Given a weight function $wt\colon \Sigma \to [0,1]$, such that $\sum_{a\in\Sigma} wt(a) = 1$, extend this to a function over Σ^*, by setting $wt(u) = \prod_{i=1}^{k} wt(a_i)$ for $u = a_1 \ldots a_k \in \Sigma^*$. Then, for $\nu, \lambda \in [0,1]$, define

$$\mu'(L) = \nu \sum_{u\in L} \lambda^{|u|} wt(u).$$

The condition that a difference for the word u counts as much as the sum of all differences for words uv properly extending u holds if we set $\lambda = \frac{1}{2}$. Furthermore, note that for $\nu = \lambda = \frac{1}{2}$, $wt(a) = \frac{1}{|\Sigma|}$ for every $a \in \Sigma$, we get the function μ defined for d_2.

5 Conclusion

We have extended unification in DLs to approximate unification in order to enhance the recall of this method of finding redundancies in DL-based ontologies. For the DL $\mathcal{FL}_0$, unification can be reduced to solving certain language equations [4]. We have shown that, w.r.t. two particular distance measures, this reduction can be extended to the approximate case. Interesting topics for future research are considering approximate unification for other DLs such as $\mathcal{EL}$ [3]; different distance measures for $\mathcal{FL}_0$ and other DLs, possibly based on similarity measures between concepts [13,16]; and approximately solving other kinds of language equations [12].

Approximate unification has been considered in the context of similarity-based Logic Programming [9], based on a formal definition of proximity between terms. The definition of proximity

used in [9] is quite different from our distances, but the major difference to our work is that [9] extends syntactic unification to the approximate case, whereas unification in $\mathcal{FL}_0$ corresponds to unification w.r.t. the equational theory $ACUIh$ (see [4]). Another topic for future research is to consider unification w.r.t. other equational theories. First, rather simple, results for the theory $ACUI$, which extend the results for strict $ACUI$-unification [10], can be found in [2].

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Baader, F., Marantidis, P., Okhotin, A.: Approximately solving set equations. In: Ghilardi, S., Schmidt-Schauß, M. (eds.) Proceedings of the 30th International Workshop on Unification (UNIF'16). Porto, Portugal (2016)
3. Baader, F., Morawska, B.: Unification in the description logic $\mathcal{EL}$. Logical Methods in Computer Science 6(3) (2010)
4. Baader, F., Narendran, P.: Unification of concept terms in description logics. J. of Symbolic Computation 31(3), 277–305 (2001)
5. Baader, F., Okhotin, A.: On language equations with one-sided concatenation. Fundamenta Informaticae 126(1), 1–35 (2013)
6. Baader, F., Tobies, S.: The inverse method implements the automata approach for modal satisfiability. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001). Lecture Notes in Artificial Intelligence, vol. 2083, pp. 92–106. Springer-Verlag (2001)
7. Banach, S.: Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. Fundamenta Mathematicae 3(1), 133–181 (1922)
8. Ecke, A., Peñaloza, R., Turhan, A.Y.: Computing role-depth bounded generalizations in the description logic $\mathcal{ELOR}$. In: Timm, I.J., Thimm, M. (eds.) Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013). Lecture Notes in Artificial Intelligence, vol. 8077, pp. 49–60. Springer-Verlag, Koblenz, Germany (2013)
9. Iranzo, P.J., Rubio-Manzano, C.: Proximity-based unification theory. Fuzzy Sets and Systems 262, 21–43 (2015)
10. Kapur, D., Narendran, P.: Complexity of unification problems with associative-commutative operators. J. Automated Reasoning 9, 261–288 (1992)
11. Kreyszig, E.: Introductory Functional Analysis With Applications. Wiley Classics Library, John Wiley & Sons (1978)
12. Kunc, M.: What do we know about language equations? In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) Proc. of the 11th International Conference on Developments in Language Theory (DLT 2007). Lecture Notes in Computer Science, vol. 4588, pp. 23–27. Springer-Verlag (2007)
13. Lehmann, K., Turhan, A.: A framework for semantic-based similarity measures for $\mathcal{ELH}$-concepts. In: del Cerro, L.F., Herzig, A., Mengin, J. (eds.) Proc. of the 13th European Conference on Logics in Artificial Intelligence (JELIA 2012). Lecture Notes in Computer Science, vol. 7519, pp. 307–319. Springer-Verlag (2012)
14. Munkres, J.: Topology. Featured Titles for Topology Series, Prentice Hall (2000)
15. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons (1999)
16. Tongphu, S., Suntisrivaraporn, B.: On desirable properties of the structural subsumption-based similarity measure. In: Supnithi, T., Yamaguchi, T., Pan, J.Z., Wuwongse, V., Buranarach, M. (eds.) Proc. of the 4th Joint International Conference on Semantic Technology (JIST 2014). Lecture Notes in Computer Science, vol. 8943, pp. 19–32. Springer-Verlag (2015)