TECHNISCHE UNIVERSITÄT DRESDEN

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

# LTCS–Report

## Approximation in Description Logics: How Weighted Tree Automata Can Help to Define the Required Concept Comparison Measures in $\mathcal{FL}_0$

Franz Baader  Oliver Fernández Gil  Pavlos Marantidis

LTCS-Report 16-08

Accepted to *LATA 2017*

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

# Approximation in Description Logics: How Weighted Tree Automata Can Help to Define the Required Concept Comparison Measures in $\mathcal{FL}_0$

Franz Baader, Oliver Fernández Gil, and Pavlos Marantidis*
firstname.lastname@tu-dresden.de

Theoretical Computer Science, TU Dresden, Germany

**Abstract.** Recently introduced approaches for relaxed query answering, approximately defining concepts, and approximately solving unification problems in Description Logics have in common that they are based on the use of concept comparison measures together with a threshold construction. In this paper, we will briefly review these approaches, and then show how weighted automata working on infinite trees can be used to construct computable concept comparison measures for $\mathcal{FL}_0$ that are equivalence invariant w.r.t. general TBoxes. This is a first step towards employing such measures in the mentioned approximation approaches.

## 1 Introduction

Description Logics (DLs) [5] are a well-investigated family of logic-based knowledge representation languages, which are frequently used to formalize ontologies for application domains such as biology and medicine [21]. To define the important notions of such an application domain as formal concepts, DLs state necessary and sufficient conditions for an individual to belong to a concept. These conditions can be atomic properties required for the individual (expressed by concept names) or properties that refer to relationships with other individuals and their properties (expressed as role restrictions). The expressivity of a particular DL $\mathfrak{L}$ is determined on the one hand by what sort of properties can be required and how they can be combined. On the other hand, DLs provide their users with ways of stating terminological axioms in a so-called TBox. The simplest kind of TBoxes are called acyclic TBoxes, which consist of concept definitions without cyclic dependencies among the defined concepts. Basically, such a TBox introduces abbreviations for complex concept descriptions. General TBoxes use so-called general concept inclusions (GCIs) to state subconcept-superconcept constraints between concepts. Once the relevant concepts of an application domain are formalized in a TBox, they can be employed to state information about specific entities (individuals, objects) and their relationships in a so-called ABox. Given a TBox and an ABox, queries can then be used to retrieve new information from the data formalized this way. We will introduce the basic notions of DLs in Section 2, and define three DLs of different expressive power, namely the DLs $\mathcal{ALC}$, $\mathcal{EL}$, and $\mathcal{FL}_0$.

Since the semantics of traditional DLs is based on classical first-order logic, the interpretation of the properties required for a concept is strict in the sense that all these properties need to be satisfied for an individual to belong to a concept, and the same is true for answers to queries. In applications where exact definitions are hard to come by, it would be useful to relax this strict requirement and allow for approximate definitions of concepts, where most, but not all, of the stated properties are required to hold. Similarly, if a query has no exact answer, approximate answers that satisfy most of the features the query is looking for could be useful. For example,

* Supported by DFG Graduiertenkolleg 1763 (QuantLA).

in clinical diagnosis, diseases are often linked to a long list of medical signs and symptoms, but patients that have a certain disease rarely show all these signs and symptoms. Instead, one looks for the occurrence of sufficiently many of them. Similarly, people looking for a flat to rent or a bicycle to buy may have a long list of desired properties, but will also be satisfied if many, but not all, of them are met.

In order to allow for approximate definitions of concepts, we have introduced the notion of a graded membership function in [4]. Instead of a Boolean membership value 0 or 1 such a graded function yields a membership degree from the interval [0,1]. Threshold concepts can then, for example, require that an individual belongs to a concept $C$ with degree at least 0.8. A different approach, which is based on the use of similarity measures on concepts [25], was used by Ecke et al. [18,19] to relax instance queries (i.e., queries that consist of a single concept). Given a query concept $C$, they are looking for answers to queries $D$ whose similarity to $C$ is higher than a certain threshold. While these two approaches were originally developed independently of each other, it has turned out that there are close connections. Similarity measures can be used to define graded membership functions, and threshold concepts w.r.t. these functions provide a more natural semantics for relaxed instance queries [4,6]. Thus, in both approximation approaches mentioned until now, the availability of appropriate measures for comparing concepts is crucial. The same is true for the approximate unification of concepts introduced in [7]. Basically, unification in DLs tries to make two concepts equivalent by replacing some of the concept names occurring in their descriptions by complex concepts [9,8]. In the approximate case, one requires that concepts are made "almost" equivalent, where the meaning of "almost" is formalized using distance measures between concepts. Strictly speaking, these distance measures are not similarity measures in the sense of [25] since they need not map into [0,1]. In the following, we will call functions that compare pairs of concepts by mapping them into a (usually numerical) domain equipped with a partial order *concept comparison measures*.

An indispensable requirement for the concept comparison measures used in the three approximation approaches mentioned above is that they respect the semantics of concepts in the sense that they are invariant under equivalence of concepts w.r.t. their definitions in the TBox. For the DL $\mathcal{EL}$, a framework for defining concept similarity measures that are equivalence invariant w.r.t. acyclic TBoxes has been introduced in [25]. This was extended in [19] to general TBoxes. For $\mathcal{FL}_0$, concept similarity measures that are equivalence invariant for acyclic TBoxes were introduced in [30]. The main technical contribution of this paper is to introduce a framework for defining *computable* concept comparison measures for $\mathcal{FL}_0$ that are *equivalence invariant w.r.t. general TBoxes*. Basically, this is achieved by leveraging a new formal language-based characterization of equivalence in $\mathcal{FL}_0$ w.r.t. general TBoxes [28], where the semantics of a concept is characterized using a tuple of (possibly infinite) formal languages. Following the ideas in [9,10,28], such tuples can be represented by (infinite) trees. These trees (or more precisely, appropriate finite representations of them) can in turn be used as inputs for weighted tree automata [31], which then yield the output of the measure. We will show that, under certain conditions on the weighted tree automata, this approach indeed yields computable concept comparison measures.

## 2 Description Logics, concept comparison measures, and approximation

We start by recalling basic notions of Description Logics, and in particular the DLs $\mathcal{ALC}$, $\mathcal{EL}$, and $\mathcal{FL}_0$. Then, we introduce concept comparison measures, which generalize concept similarity measures, and finally we show how such measures can be used to relax query answering, approximately define concepts, and approximately solve unification problems.

### 2.1 Description Logics

In Description Logics, *concept constructors* are used to build complex *concept descriptions* out of *concept names* (unary predicates) and *role names* (binary predicates). A particular DL $\mathfrak{L}$ is determined by the available constructors. Given finite, disjoint sets $\mathsf{N_C}$ and $\mathsf{N_R}$ of concept names and role names, respectively, we denote the set of all concept descriptions that can be built from $\mathsf{N_C}$ and $\mathsf{N_R}$ using the constructors of $\mathfrak{L}$ with $\mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R})$.

As an example, consider the constructors top concept ($\top$), bottom concept ($\bot$), conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation ($\neg C$), value restriction ($\forall r.C$), and existential restriction ($\exists r.C$), which determine the DL $\mathcal{ALC}$. Then, $\mathcal{C}_{\mathcal{ALC}}(\mathsf{N_C}, \mathsf{N_R})$ is inductively defined as follows:

- $\{\top, \bot\} \cup \mathsf{N_C} \subseteq \mathcal{C}_{\mathcal{ALC}}$,
- if $C, D \in \mathcal{C}_{\mathcal{ALC}}$ and $r \in \mathsf{N_R}$, then $\{C \sqcap D, C \sqcup D, \neg C, \forall r.C, \exists r.C\} \subseteq \mathcal{C}_{\mathcal{ALC}}$.

We will also consider the following two sub-logics $\mathcal{EL}$ and $\mathcal{FL}_0$ of $\mathcal{ALC}$:

- $\mathcal{EL}$ has the constructors top concept, conjunction, *existential* restriction;
- $\mathcal{FL}_0$ has the constructors top concept, conjunction, *value* restriction.

The *semantics* of a DL $\mathfrak{L}$ is defined using first-order interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each concept name $A \in \mathsf{N_C}$ and a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role name $r \in \mathsf{N_R}$. This function is extended to complex concept descriptions by assigning a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each $C \in \mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R})$ according to the semantics of the constructors of $\mathfrak{L}$. The semantics of the constructors is defined by equations that enable the inductive definition of $C^{\mathcal{I}}$ for any interpretation $\mathcal{I}$.

For the above constructors, the equations fixing their semantics are as follows:

$$\begin{aligned}
&\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \text{ and } \bot^{\mathcal{I}} = \emptyset,\\
&(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \text{ and } (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\\
&(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\},\\
&(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}.
\end{aligned}$$

An $\mathfrak{L}$ *terminology (TBox)* $\mathcal{T}$ is a finite set of *general concept inclusions (GCIs)*, which are expressions of the form $C \sqsubseteq D$ for $C, D \in \mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R})$. The interpretation $\mathcal{I}$ is a *model* of $\mathcal{T}$ if it satisfies all its GCIs, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all GCIs $C \sqsubseteq D$ in $\mathcal{T}$. An $\mathfrak{L}$ *ABox* $\mathcal{A}$ is a finite set of *assertions*, which are expressions of the form $C(a)$ or $r(a, b)$, where $C \in \mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R}), r \in \mathsf{N_R}$ and $a, b$ are elements of an additional set $\mathsf{N_I}$ of individual names, which is disjoint with $\mathsf{N_C}$ and $\mathsf{N_R}$. An interpretation then additionally assigns elements $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to individual names $a \in \mathsf{N_I}$. The interpretation $\mathcal{I}$ is a *model* of $\mathcal{A}$ if it satisfies all its assertions, i.e., $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$) holds for all assertions $C(a)$ (resp. $r(a, b)$) in $\mathcal{A}$.

Given an $\mathfrak{L}$ TBox $\mathcal{T}$ and two $\mathfrak{L}$ concept descriptions $C, D$, we say that $C$ *is subsumed by* $D$ *w.r.t.* $\mathcal{T}$ (denoted as $C \sqsubseteq_{\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$. These two concept descriptions are *equivalent* (denoted as $C \equiv_{\mathcal{T}} D$) if $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$. Equivalent concept descriptions have the same meaning w.r.t. $\mathcal{T}$ in the sense that they always (i.e., in every model of $\mathcal{T}$) yield the same set. In the presence of an $\mathfrak{L}$ ABox $\mathcal{A}$, we can also consider the *instance problem*: given an individual name $a$ and an $\mathfrak{L}$ concept description $C$ we say that $a$ *is an instance of* $C$ *in* $\mathcal{A}$ *w.r.t.* $\mathcal{T}$ (written $\mathcal{A} \models_{\mathcal{T}} C(a)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$ that are also models of $\mathcal{A}$. For the DL $\mathcal{EL}$, the subsumption, equivalence, and instance problem are polynomial [14] whereas they are ExpTime-complete for $\mathcal{FL}_0$ [3] and for $\mathcal{ALC}$ [32].

### 2.2 Concept comparison measures

Subsumption and equivalence can be seen as operations that compare concept descriptions, and yield the comparison value 1 if the relation holds and 0 otherwise, i.e.,

$$\sqsubseteq_{\mathcal{T}}(C,D) = 1 \text{ if } C \sqsubseteq_{\mathcal{T}} D \text{ and } \sqsubseteq_{\mathcal{T}}(C,D) = 0 \text{ if } C \not\sqsubseteq_{\mathcal{T}} D,$$

and accordingly for equivalence. Intuitively, concept comparison measures generalize such operations by yielding a degree to which the comparison relation is satisfied. More formally, they return a value in a partially ordered set.

**Definition 1.** *Let $\mathfrak{L}$ be a DL. A* concept comparison measure (CCM) *for $\mathfrak{L}$ is a family of functions $\mathfrak{c}$ that contains, for every $\mathfrak{L}$ TBox $\mathcal{T}$, an equivalence invariant function $\mathfrak{c}_{\mathcal{T}} : \mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R}) \times \mathcal{C}_{\mathfrak{L}}(\mathsf{N_C}, \mathsf{N_R}) \to S$, where*

- *$S$ is a non-empty set equipped with a partial order $\leq_S$,*
- *and* equivalence invariant *means that $\mathfrak{c}_{\mathcal{T}}(C,D) = \mathfrak{c}_{\mathcal{T}}(C',D')$ whenever $C \equiv_{\mathcal{T}} C'$ and $D \equiv_{\mathcal{T}} D'$.*

The reason we require equivalence invariance is that we do not view concept descriptions as syntactic objects, but rather as semantic ones that, for every interpretation, yield a subset of the interpretation domain. Since equivalent concept descriptions always yield the same sets, they are the same objects from a semantic point of view, and thus should also be treated the same way by the comparison function. The partial order on $S$ allows us to compare different comparison degrees. We will later use the natural numbers and the non-negative real numbers, possibly extended with infinity $+\infty$, as well as the closed real interval $[0,1]$ with the obvious orders as sets $S$.

Well-investigated examples of CCMs are *concept similarity measures (CSMs)*, for which $S = [0,1]$ (see e.g., [25]). Intuitively, a CSM $\bowtie_{\mathcal{T}}$ is a graded variant of equivalence, where two concept descriptions $C, D$ are equivalent iff $\bowtie_{\mathcal{T}}(C,D) = 1$, and they become less and less similar with decreasing value $\bowtie_{\mathcal{T}}(C,D)$. Usually, one also requires CSMs to be symmetric in the sense that $\bowtie_{\mathcal{T}}(C,D) = \bowtie_{\mathcal{T}}(D,C)$. For the DL $\mathcal{EL}$, a framework for defining CSMs satisfying certain additional properties has been introduced in [25], but equivalence invariance was only achieved for so-called acyclic TBoxes. This was extended in [19] to general TBoxes. For $\mathcal{FL}_0$, CSMs that are equivalence invariant for acyclic TBoxes were introduced in [30]. We will show later how CCMs for $\mathcal{FL}_0$ that are equivalence invariant for general TBoxes can be obtained by using weighted tree automata. CSMs for $\mathcal{ALC}$ are, for instance, investigated in [15].

Our definition of CCMs encompasses CSMs, but also covers other measures such as concept distance measures, which are mappings into $[0, +\infty)$ for which a larger value indicates that the concept descriptions are less similar (see e.g., [7]). In addition, it covers graded variants of subsumption, which map into $[0,1]$, but in contrast to CSMs are not supposed to be symmetric. For example, the CSMs for $\mathcal{EL}$ and $\mathcal{FL}_0$ in [34] and [30], respectively, are based on asymmetric concept subsumption measures, which are then turned into symmetric CSMs by combining the results of the comparisons in both directions by computing the average.

### 2.3 Approximation

In contrast to approaches that try to speed up reasoning by employing approximate inference techniques [27], we use approximation as a way to extend the range of admissible answers to queries or admissible elements of concept descriptions. In this context, CCMs can be used together with a threshold construction to define which answers or individuals are admissible.

**Relaxing instance queries** Ecke et al. [18,19] use CSMs to *relax instance queries* in $\mathcal{EL}$, i.e., instead of requiring that an individual is an instance of the query concept, they only require that it is an instance of a concept description that is "similar enough" to the query concept.

**Definition 2.** *Let $\bowtie$ be a CSM for $\mathcal{EL}$, $\mathcal{T}$ an $\mathcal{EL}$ TBox, $\mathcal{A}$ an $\mathcal{EL}$ ABox, and $t \in [0,1)$. The individual $a \in \mathsf{N_I}$ is a* relaxed instance *of the $\mathcal{EL}$ concept description $Q$ w.r.t. $\mathcal{T}$, $\mathcal{A}$, $\bowtie$, and the threshold $t$ if there exists an $\mathcal{EL}$ concept description $X$ such that $\bowtie_{\mathcal{T}}(Q, X) > t$ and $\mathcal{A} \models_{\mathcal{T}} X(a)$.*

Ecke et al. [18,19] show that, under certain conditions on the CSMs used, the relaxed instance problem for $\mathcal{EL}$ is decidable. They also introduce a class of polynomially computable CSMs on $\mathcal{EL}$ concept descriptions for which the relaxed instance problem is in NP.

**Adding threshold concepts to $\mathcal{EL}$** In [4], a similar construction is used to *relax membership in $\mathcal{EL}$ concept descriptions*. To be more precise, the authors introduce the notion of a graded membership function $m$ to generalize elementhood in concept descriptions, and then use a threshold construction to obtain new concept constructors.

**Definition 3.** *A* graded membership function *$m$ is a family of functions that contains for every interpretation $\mathcal{I}$ a function $m^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \mathcal{C}_{\mathcal{EL}}(\mathsf{N_C}, \mathsf{N_R}) \to [0,1]$ satisfying the following conditions (for $C, D \in \mathcal{C}_{\mathcal{EL}}(\mathsf{N_C}, \mathsf{N_R})$):*

$$\textit{M1: } \forall \mathcal{I}\ \forall d \in \Delta^{\mathcal{I}} : d \in C^{\mathcal{I}} \Leftrightarrow m^{\mathcal{I}}(d, C) = 1,$$
$$\textit{M2: } C \equiv D \Leftrightarrow \forall \mathcal{I}\ \forall d \in \Delta^{\mathcal{I}} : m^{\mathcal{I}}(d, C) = m^{\mathcal{I}}(d, D).$$

Intuitively, given an interpretation $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$, $m^{\mathcal{I}}(d, C) \in [0,1]$ represents the degree to which $d$ belongs to $C$ in $\mathcal{I}$. The threshold concept $C_{\sim t}$ for $\sim \in \{<, \leq, >, \geq\}$ then collects all the elements of $\Delta^{\mathcal{I}}$ that belong to $C$ with degree $\sim t$, as measured by $m$. To be more precise, the formal semantics of threshold concepts is then defined as follows: $(C_{\sim t})^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid m^{\mathcal{I}}(d, C) \sim t\}$. The DL $\tau\mathcal{EL}(m)$ extends $\mathcal{EL}$ with such threshold concepts.

In [4] a specific such graded membership function called *deg* is introduced and the complexity of reasoning in $\tau\mathcal{EL}(deg)$ w.r.t. empty TBoxes (NP-complete or coNP-complete, depending on the reasoning problem) is investigated in detail. In addition, it is shown that, using a construction similar to the one in Definition 2, a CSM $\bowtie$ satisfying certain properties can be used to define a graded membership function:

$$m^{\mathcal{I}}_{\bowtie}(d, C) := \max\{\bowtie_{\emptyset}(C, D) \mid D \in \mathcal{C}_{\mathcal{EL}}(\mathsf{N_C}, \mathsf{N_R}) \text{ and } d \in D^{\mathcal{I}}\}.$$

To ensure that this construction yields a well-defined graded membership function, the CSM must be equivalence invariant, role-depth bounded, and equivalence closed (see [25,4] for definitions of the latter two properties). Finally, the authors of [4] prove that answering relaxed instance queries w.r.t. $\bowtie$ is the same as answering instance queries for threshold concepts $C_{>t}$ in $\tau\mathcal{EL}(m_{\bowtie})$.

In [6] it is shown that, for *computable* CSMs $\bowtie$ satisfying these properties, reasoning in $\tau\mathcal{EL}(m_{\bowtie})$ can effectively be reduced to reasoning in the DL $\mathcal{ALC}$, but the reduction is in general non-elementary. The authors then introduce a class of CSMs $\bowtie$ such that reasoning in $\tau\mathcal{EL}(m_{\bowtie})$ has the same complexity as reasoning in $\tau\mathcal{EL}(deg)$.

### 2.4 Approximate unification

Unification has been introduced as a novel inference service that can be used to detect redundancies in ontologies. Basically, in unification one views some of the concept names in concept

descriptions as variables, which can be replaced by concept descriptions using a *substitution*. The goal is then to make two concept descriptions equivalent by applying the same substitution to both. For example, consider the $\mathcal{FL}_0$ concept descriptions

$$C = A \sqcap \forall r.(X \sqcap \forall s.B) \quad \text{and} \quad D = Y \sqcap \forall r.(Z \sqcap A \sqcap \forall r.B).$$

Obviously, the substitution $\sigma$ that replaces $X$ by $A \sqcap \forall r.B$, $Y$ by $A$, and $Z$ by $\forall s.B$ makes $C, D$ equivalent (w.r.t. the empty TBox):

$$\sigma(C) = A \sqcap \forall r.(A \sqcap \forall r.B \sqcap \forall s.B) \equiv_\emptyset A \sqcap \forall r.(\forall s.B \sqcap A \sqcap \forall r.B) = \sigma(D).$$

Such substitutions are called *unifiers*.

Unification was first investigated in detail for the DL $\mathcal{FL}_0$ [9], and later on for the DL $\mathcal{EL}$ [8]. The unification problem, i.e., deciding whether two concept descriptions with variables have a unifier or not, is ExpTime-complete in $\mathcal{FL}_0$ and NP-complete in $\mathcal{EL}$. Both works basically restrict their attention to the case of an empty TBox. For $\mathcal{EL}$, some attempts have been made to extend the results to unification w.r.t. GCIs [2], but these approaches can at the moment only deal with TBoxes that satisfy a certain restriction on cyclic dependencies between concept names. For $\mathcal{ALC}$, decidability of the unification problem (even w.r.t. the empty TBox) is a longstanding open problem, though it is known that undecidability holds for extensions of $\mathcal{ALC}$ by so-called nominals or the universal role [36].

Approximate unification relaxes the requirement that the two concept descriptions must be made equivalent. Instead, it requires that they are made "almost" equivalent, where the meaning of "almost" is formalized using distance measures between concept descriptions. Such measures are CCMs that map into $[0, +\infty)$ and satisfy some additional properties. Basically, given such a measure and a threshold value, one then asks whether one can lower the distance between two concept descriptions below the threshold value by applying a substitution. This is called the decision problem for approximate unification. For the computation problem, one wants to calculate the lowest achievable distance.

In [7], approximate unification is introduced and then investigated for the DL $\mathcal{FL}_0$ and two concept distance measures that are induced by distance measures between formal languages (see the next section). It is shown that (w.r.t. the empty TBox and these two measures), approximate unification has the same complexity (ExpTime) as unification.

## 3 Concept comparison measures for $\mathcal{FL}_0$

Until recently, the research on concept comparison measures in DLs was mostly concerned with $\mathcal{EL}$ [25,19,34] and more expressive DLs [15]. To achieve equivalence invariance, concept descriptions are usually first translated into an appropriate normal form, and then the structure of the normalized descriptions is compared. For instance, measures that achieve equivalence invariance only for the empty TBox or for acyclic TBoxes in $\mathcal{EL}$ [25,34] make use of the reduced form of $\mathcal{EL}$ concept descriptions introduced in [24]. Extensions to general TBoxes [19] use the so-called canonical model, which is generated by the polynomial-time subsumption algorithm for $\mathcal{EL}$ [3].

Two recent approaches for defining concept comparison measures for $\mathcal{FL}_0$ [30,7] were restricted to the case of the empty TBox. Both approaches employ a formal language-based characterization of equivalence between $\mathcal{FL}_0$ concept descriptions. In the remainder of this paper, we will develop a general approach for defining concept comparison measures for $\mathcal{FL}_0$ concept descriptions that are equivalence invariant also w.r.t. general TBoxes. This is achieved by using a new formal language-based characterization of equivalence in $\mathcal{FL}_0$ w.r.t. general TBoxes [28], where

the semantics of a concept description is characterized using a tuple of (possibly infinite) formal languages. Basically, this tuple serves as a normal form for the concept description. In order to define equivalence invariant measures on $\mathcal{FL}_0$ concept descriptions, it is thus sufficient to define measures that compare such tuples. We will show how tuples of languages can be represented by infinite trees, and then use appropriate weighted tree automata to compute the comparison value.

### 3.1 From $\mathcal{FL}_0$ concept descriptions to tuples of formal languages

In $\mathcal{FL}_0$, subsumption and equivalence can be nicely characterized using language inclusion. This characterization relies on transforming $\mathcal{FL}_0$ concept descriptions into an appropriate normal form as follows. First, the semantics given to concept constructors in $\mathcal{FL}_0$ implies that value restrictions distribute over conjunction, i.e., for all $C, D \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ and $r \in \mathsf{N_R}$ it holds that

$$\forall r.(C \sqcap D) \equiv_\emptyset \forall r.C \sqcap \forall r.D.$$

Using this equivalence as a rewrite rule from left to right, each $\mathcal{FL}_0$ concept description can be translated into an equivalent concept description that is either $\top$ or a conjunction of concept descriptions of the form $\forall r_1 \dots \forall r_n.A$, where $\{r_1, \dots, r_n\} \subseteq \mathsf{N_R}$ and $A \in \mathsf{N_C}$. Such concept descriptions can be abbreviated as $\forall w.A$, where $w$ represents the word $r_1 \dots r_n$. Note that $n = 0$ means that $w$ is the empty word $\varepsilon$, and thus $\forall \varepsilon.A$ corresponds to $A$. Furthermore, a conjunction of the form $\forall w_1.A \sqcap \dots \sqcap \forall w_m.A$ can be written as $\forall L.A$ where $L \subseteq \mathsf{N_R}^*$ is the finite language $\{w_1, \dots, w_m\}$. We use the convention that $\forall \emptyset.A$ corresponds to the top concept $\top$. Thus, if we fix the set of concept names as $\mathsf{N_C} := \{A_1, \dots, A_\ell\}$, then any two concept descriptions $C, D \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ can be represented as

$$\begin{aligned} C &\equiv_\emptyset \forall K_1.A_1 \sqcap \dots \sqcap \forall K_\ell.A_\ell, \\ D &\equiv_\emptyset \forall L_1.A_1 \sqcap \dots \sqcap \forall L_\ell.A_\ell, \end{aligned} \tag{1}$$

where $K_1, L_1, \dots, K_\ell, L_\ell$ are finite languages over the alphabet of role names $\mathsf{N_R}$, i.e., finite subsets of $\mathsf{N_R}^*$. Using this representation, it was shown in [9] that $C \sqsubseteq_\emptyset D$ iff $L_i \subseteq K_i$ for all $i, 1 \leq i \leq \ell$. Since equivalence corresponds to subsumption in both directions, the $\mathcal{FL}_0$ concept descriptions $C, D$ in (1) are equivalent w.r.t. the empty TBox iff $L_i = K_i$ for all $i, 1 \leq i \leq \ell$.

In the presence of a non-empty TBox $\mathcal{T}$, a similar characterization of subsumption and equivalence can be obtained [28], but now the definition of the languages needs to take the GCIs in $\mathcal{T}$ into account. Given an $\mathcal{FL}_0$ concept description $C$ and a TBox $\mathcal{T}$, we define for all $A \in \mathsf{N_C}$ the following language

$$\mathcal{L}_\mathcal{T}(C, A) := \{w \in \mathsf{N_R}^* \mid C \sqsubseteq_\mathcal{T} \forall w.A\},$$

and call this language the value restriction set of $C$ w.r.t. $\mathcal{T}$ and $A$. It can easily be verified that, for all concept names $A_i \in \mathsf{N_C}$, the sets $K_i$ in (1) are actually equal to $\mathcal{L}_\emptyset(C, A_i)$. However, while in the case of the empty TBox these languages are finite, they may be infinite for non-trivial TBoxes. This is illustrated by the following example.

*Example 1.* Let $C$ be the $\mathcal{FL}_0$ concept description $C := A \sqcap \forall s.(A \sqcap B)$ and $\mathcal{T}$ the TBox $\mathcal{T} := \{A \sqsubseteq \forall r.A, B \sqsubseteq \forall s.B\}$. It is easy to see that the value restriction sets for $A$ and $B$ are

$$\mathcal{L}_\mathcal{T}(C, A) = r^* \cup sr^* \quad \text{and} \quad \mathcal{L}_\mathcal{T}(C, B) = ss^*,$$

where we have used standard notation for writing regular expressions to describe these infinite languages.

Just as in the case of the empty TBox, the value restriction sets can be used to characterize equivalence and subsumption w.r.t. general TBoxes (see [28]):

$$C \sqsubseteq_{\mathcal{T}} D \quad \text{iff} \quad \mathcal{L}_{\mathcal{T}}(D, A_i) \subseteq \mathcal{L}_{\mathcal{T}}(C, A_i) \quad (1 \leq i \leq \ell), \tag{2}$$

$$C \equiv_{\mathcal{T}} D \quad \text{iff} \quad \mathcal{L}_{\mathcal{T}}(C, A_i) = \mathcal{L}_{\mathcal{T}}(D, A_i) \quad (1 \leq i \leq \ell). \tag{3}$$

The equivalence (3) shows that, in $\mathcal{FL}_0$, formal languages can be used to represent the semantic content of concept descriptions: up to equivalence, every $\mathcal{FL}_0$ concept description $C \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ is uniquely represented by the tuple of languages

$$\mathcal{L}_{\mathcal{T}}(C) := (\mathcal{L}_{\mathcal{T}}(C, A_1), \ldots, \mathcal{L}_{\mathcal{T}}(C, A_\ell)).$$

We will use this fact to reduce the definition of concept comparison measures between $\mathcal{FL}_0$ concept descriptions w.r.t. a TBox to the definition of measures comparing tuples of languages: given two $\mathcal{FL}_0$ concept descriptions $C, D$, we define $\mathfrak{c}_{\mathcal{T}}(C, D)$ by comparing the tuples $\mathcal{L}_{\mathcal{T}}(C)$ and $\mathcal{L}_{\mathcal{T}}(D)$. One advantage of this approach is that equivalence invariance comes "for free" since equivalent concept descriptions are indistinguishable from the language point of view.

### 3.2 Using tuples of languages to define CCMs

The idea of using tuples of languages to compare $\mathcal{FL}_0$ concept descriptions has already been employed in [30,7], but restricted to the empty TBox. In both works, the general approach used to define such measures consists of the following three steps:

1. Translate the $\mathcal{FL}_0$ concept descriptions $C$ and $D$ into their corresponding tuples of languages $\mathcal{L}_\emptyset(C) = (K_1, \ldots, K_\ell)$ and $\mathcal{L}_\emptyset(D) = (L_1, \ldots, L_\ell)$. For the sake of readability, we will denote these tuples as $\mathbf{K}$ and $\mathbf{L}$, respectively.
2. To compare the tuples $\mathbf{K}$ and $\mathbf{L}$, their components $K_i$ and $L_i$ are compared pairwise, and the values obtained this way are then appropriately combined into a value $s(\mathbf{K}, \mathbf{L})$.
3. Finally, the value $s(\mathbf{K}, \mathbf{L})$ is used to define $\mathfrak{c}_\emptyset(C, D)$.

In the following, we recall the exact definitions of the measures introduced in [30] and [7].

*Example 2.* In [30], the authors' goal is to define concept similarity measures. To this end, given $\mathbf{K}$ and $\mathbf{L}$, they first define an asymmetric measure $s$, which they apply to $(\mathbf{K}, \mathbf{L})$ and $(\mathbf{L}, \mathbf{K})$. The obtained values are then combined using average. For the definition of the asymmetric measure, they propose two possible functions $e_1$ and $e_2$ to compare every pair $(K_i, L_i)$:

- the function $e_1$ checks inclusion: $e_1(K_i, L_i) = 1$ if $L_i \subseteq K_i$, and 0 otherwise;
- the function $e_2$ returns the fraction of the words in $L_i$ that also belong to $K_i$, and thus yields 1 if $L_i \subseteq K_i$, but 0 only if the two languages are disjoint:[1]

$$e_2(K_i, L_i) = \frac{|K_i \cap L_i|}{|L_i|}$$

The asymmetric measure $s$ is then defined as $s(\mathbf{K}, \mathbf{L}) = f(e_j(K_1, L_1), \ldots, e_j(K_\ell, L_\ell))$, where $f$ is the average operator and $j \in \{1, 2\}$. Finally, the CSM $\bowtie_\emptyset$ for $\mathcal{FL}_0$ concept descriptions is defined as

$$\bowtie_\emptyset(C, D) := \frac{s(\mathcal{L}_\emptyset(C), \mathcal{L}_\emptyset(D)) + s(\mathcal{L}_\emptyset(D), \mathcal{L}_\emptyset(C))}{2}.$$

[1] Note that this function is well-defined only for finite languages. Thus, $e_2$ cannot be used in the presence of general TBoxes, where the languages may be infinite.

*Example 3.* In [7], the authors introduce the notion of concept distance measures for $\mathcal{FL}_0$. They obtain such measures by applying a language distance (which is assumed to be a topological metric) to the pairs of languages $(K_i, L_i)$, and then combining these values using a function $f$. In particular, they define two language distances $d_1$ and $d_2$, which we introduce below.

Let $M_1$ and $M_2$ be two languages over an alphabet $\Sigma$. We denote the symmetric difference of $M_1$ and $M_2$ as $M_1 \,\Delta\, M_2$, i.e.,

$$M_1 \,\Delta\, M_2 := (M_1 \setminus M_2) \cup (M_2 \setminus M_1). \tag{4}$$

The language distances $d_1$ and $d_2$ are now defined as

$$\begin{aligned} d_1(M_1, M_2) &:= 2^{-n} && \text{where } n = \min\{|w| \mid w \in M_1 \,\Delta\, M_2\}, \\ d_2(M_1, M_2) &:= \mu(M_1 \,\Delta\, M_2) && \text{where } \mu(M) = \tfrac{1}{2} \sum_{w \in M} (2|\Sigma|)^{-|w|}. \end{aligned}$$

Intuitively, the symmetric difference captures all the discrepancies between two concept descriptions $C$ and $D$ with respect to a concept name $A$. More precisely, if for instance, $w \in \mathcal{L}_\emptyset(C, A) \setminus \mathcal{L}_\emptyset(D, A)$ for some $w \in \mathsf{N_R}^*$, then $C \sqsubseteq_\emptyset \forall w.A$ and $D \not\sqsubseteq_\emptyset \forall w.A$, which amounts to a semantically relevant difference between $C$ and $D$. Based on this intuition, the first distance looks for the shortest such discrepancy, while the second one takes all differences into account, but differences for longer words count less than differences for shorter ones (see [7] for a more detailed explanation).

As already mentioned, these language distances are then used to define a measure $s$ on tuples by setting $s(\mathbf{K}, \mathbf{L}) := f(d_j(K_1, L_1), \ldots, d_j(K_\ell, L_\ell))$ where $j \in \{1, 2\}$. For functions $f$ satisfying certain properties (called *combining* functions in [7]), this yields a concept distance $m_{d,f}$ for $\mathcal{FL}_0$ concept descriptions:

$$m_{d,f}(C, D) := s(\mathcal{L}_\emptyset(C), \mathcal{L}_\emptyset(D))$$

In the two examples above, the definition of a CCM for $\mathcal{FL}_0$ was in the end reduced to define a distance function that compares two languages. Thus, the input for this function is a pair of languages. In general, one may also want to allow for definitions of distance functions on language tuples that do not resort to binary comparisons of the components of the tuples. The inputs for the function are then $2\ell$-tuples of languages. For this reason we now develop means for defining functions that receive tuples of languages as input, which covers both the binary and the general case.

Though developed for the case of the empty TBox, and thus with finite languages in mind, the functions $e_1, d_1, d_2$ of our examples are also well-defined for infinite languages, and thus can also be employed in the more general setting of non-empty TBoxes. However, if we are not only interested in defining, but also in computing the functions, we need to find ways of representing their input (i.e., tuples of possibly infinite languages) in a finite way. In the next section, we show that finite automata working on infinite trees can be used for this purpose.

### 3.3 Finitely representing tuples of languages

Following the ideas in [9,10,28], we will represent tuples of (possibly infinite) languages using infinite trees.

**Definition 4.** *Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a non-empty finite set of symbols. Given a set of labels $L$, an $L$-labeled $\Sigma$-tree is a mapping $t : \Sigma^* \to L$ that assigns a label $t(w) \in L$ to every node $w \in \Sigma^*$. The set of all $L$-labeled $\Sigma$-trees is denoted as $T^\omega_{\Sigma,L}$.*

Intuitively, the nodes of a $\Sigma$-tree $t$ correspond to finite words in $\Sigma^*$, where the empty word $\varepsilon$ represents the root of $t$ and every node $w$ has $k$ children corresponding to the words $w\sigma_1, \ldots, w\sigma_k$. Since for a non-empty alphabet $\Sigma$ the set $\Sigma^*$ of all words over $\Sigma$ is infinite, $\Sigma$-trees are by definition infinite. We use tuples over $\{0,1\}$ as labels to represent tuples of languages over $\Sigma$.

**Definition 5.** *Let $\Sigma$ be a finite set of symbols and $\ell \in \mathbb{N}$. We define the mapping $\gamma_\ell : \left(2^{\Sigma^*}\right)^\ell \to T^\omega_{\Sigma,\{0,1\}^\ell}$ as follows. Given a tuple of languages $\mathbf{L} = (L_1, \ldots, L_\ell)$ over $\Sigma$, $\gamma_\ell(\mathbf{L}) := t_{\mathbf{L}}$ where $t_{\mathbf{L}} : \Sigma^* \to \{0,1\}^\ell$ is the $\Sigma$-tree such that*

$$t_{\mathbf{L}}(w) := (x_1, \ldots, x_\ell), \text{ where } x_i = 1 \text{ iff } w \in L_i \quad (\text{for all } w \in \Sigma^*).$$

It is easy to see that $\gamma_\ell$ is a *bijection* between tuples of languages over the alphabet $\Sigma$ and $\{0,1\}^\ell$-labeled $\Sigma$-trees. Given a tree $t \in T^\omega_{\Sigma,\{0,1\}^\ell}$, the inverse function yields the tuple $\gamma_\ell^{-1}(t) = (L_1, \ldots, L_\ell)$ where $L_i$ consists of the words $w$ for which the $i$th component of $t(w)$ is equal to 1.

Basically, this translation of tuples of languages into trees is used in [28] to represent the tuple of value restriction sets $\mathcal{L}_\mathcal{T}(C)$ of an $\mathcal{FL}_0$ concept description $C$ as an $\mathsf{N_R}$-tree $t_C$. Strictly speaking, the label set employed in [28] is $2^{\mathsf{N_C}}$ for $\mathsf{N_C} = \{A_1, \ldots, A_\ell\}$ rather than $\{0,1\}^\ell$, but it should be clear that, by fixing a linear order $A_1 < A_2 < \ldots < A_\ell$ on $\mathsf{N_C}$, these two representations can be translated into each other. Obviously, a single value restriction set $\mathcal{L}_\mathcal{T}(C, A_i)$ can be represented as a $\{0,1\}$-labeled $\mathsf{N_R}$-tree $t_{C,A_i}$, where the words belonging to this language receive label 1 and the others label 0.

The following example illustrates this representation of value restriction sets by trees using the concept description $C$ and the TBox $\mathcal{T}$ of Example 1.

*Example 4.* Recall from Example 1 that $\mathcal{L}_\mathcal{T}(C, A) = r^* \cup sr^*$ and $\mathcal{L}_\mathcal{T}(C, B) = ss^*$. To express the tuple of these languages as a tree, we assume that $r$ is the first symbol of the alphabet and $s$ is the second, and that $A < B$. Then $\mathcal{L}_\mathcal{T}(C) = (r^* \cup sr^*, ss^*)$, and this tuple is represented by the tree sketched on the left-hand side of Figure 1. For better readability, we have labeled the edges with the symbols $r$ and $s$. As an example for the labeling, consider the node corresponding to the word $sr$. It has label $(1,0)$ since this word belongs to $r^* \cup sr^*$, but not to $ss^*$. The extension of this tree to infinity is obtained as follows. On the one hand, the outgoing *dotted* edges tell us that all the nodes below are labeled with the tuple $(0,0)$. Notice, for example, that there are no words starting with $rs$ or $srs$ in any of the two languages. On the other hand, the nodes $rrr$, $srr$ and $sss$ are the roots of infinite trees representing the tuples of languages $(r^*, \emptyset)$, $(r^*, \emptyset)$ and $(\emptyset, s^*)$, respectively.

The tree on the right-hand side of the figure represents the language $\mathcal{L}_\mathcal{T}(C, A)$, which is obtained from $t_C$ by projecting the label-tuples to the first component.

Using the same approach, given two concept descriptions $C, D$, the pair of tuples $(\mathcal{L}_\mathcal{T}(C), \mathcal{L}_\mathcal{T}(D))$ can obviously be represented as an infinite $\mathsf{N_R}$-tree $t_{(C,D)} : \mathsf{N_R}^* \to \{0,1\}^\ell \times \{0,1\}^\ell$.

As mentioned before, our goal is to represent such input tuples in a finite way. Using infinite trees obviously does not solve this problem. Thus, we need to develop an approach for representing such trees in a finite way. For general tuples of infinite languages and thus arbitrary $\Sigma$-trees this is clearly not possible. However, a closer look at the trees $t_C$ constructed in [28] shows that they are actually *regular* trees, which admit a finite representation. Therefore, we restrict our attention to the class of regular trees. We start by formally defining the notion of a regular tree, and then show that regular trees can always be represented using certain kinds of tree automata.

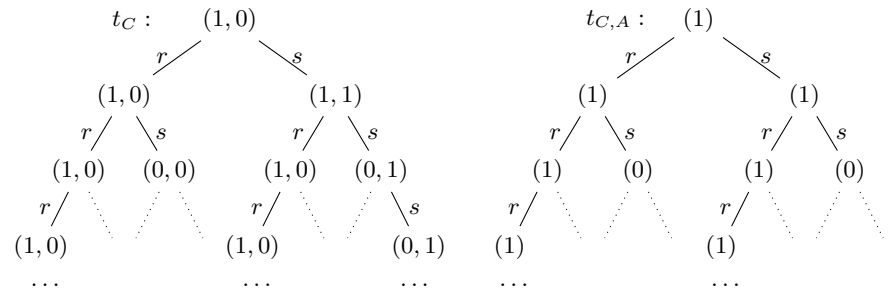


**Fig. 1.** Tuples of languages as infinite trees.

**Definition 6 (regular tree).** *Let $t$ be a tree in $T^{\omega}_{\Sigma,L}$. Given a node $w \in \Sigma^*$, the* subtree *$t_w : \Sigma^* \to L$ of $t$ is defined as $t_w(v) := t(wv)$ for all $v \in \Sigma^*$. We say that $t$ contains the subtree $t_w$. Then, $t$ is a* regular *tree if it contains finitely many distinct subtrees.*

There are different ways to represent regular trees in a finite way [35]. Here, we use *looping tree automata* for this purpose.

**Definition 7 (Looping tree automaton (LTA)).** *A* looping tree automaton *is a tuple $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ where $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite set of symbols, $Q$ is a finite set of states, $L$ is a finite set of labels, $\Delta \subseteq Q \times L \times Q^k$ is the transition relation and $I \subseteq Q$ is a set of initial states. A run of this automaton on a tree $t \in T^{\omega}_{\Sigma,L}$ is a $Q$-labeled $\Sigma$-tree $r : \Sigma^* \to Q$ such that $r(\varepsilon) \in I$ and*

$$(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \in \Delta$$

*for all $w \in \Sigma^*$. The tree language $\mathcal{L}(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of all trees $t \in T^{\omega}_{\Sigma,L}$ such that $\mathcal{A}$ accepts $t$, i.e., $\mathcal{A}$ has a run on $t$.*

In general, LTAs recognize *sets* of trees. Therefore, to uniquely represent a tree we only consider those recognizing singleton sets.

**Definition 8.** *Let $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ be a looping tree automaton. We say that $\mathcal{A}$* represents *the infinite tree $t \in T^{\omega}_{\Sigma,L}$ if $\mathcal{L}(\mathcal{A}) = \{t\}$.*

It is easy to see that trees that can be represented by looping tree automata are indeed regular. In fact, LTAs are Rabin tree automata [29,35] with trivial acceptance conditions, and it is well-known that non-empty tree languages recognized by Rabin tree automata always contain a regular tree. Thus, if such an automaton recognizes the singleton set $\{t\}$, then $t$ must be regular. Conversely, we can show that any regular tree can be represented in this way.

**Proposition 1.** *Let $t \in T^{\omega}_{\Sigma,L}$ be an $L$-labeled $\Sigma$-tree. Then, $t$ is regular iff it can be represented by an LTA.*

*Proof.* We have already seen that the if-direction holds. To show the only-if direction, assume that $t$ is a regular tree. By Definition 6 it thus contains only finitely many distinct subtrees, say $t^0, t^1, \ldots, t^m$ where we assume without loss of generality that $t^0 = t$. For all $1 \leq i \leq m$, we denote the direct subtrees of $t^i$ as $t^i_{\sigma_1}, \ldots, t^i_{\sigma_k}$. Note that these are also subtrees of $t$, and thus belong to the set $\{t^0, t^1, \ldots, t^m\}$. We build the looping tree automaton $\mathcal{A}_t = (\Sigma, Q_t, L, \Delta_t, \{t^0\})$ as follows: $Q_t := \{t^0, t^1, \ldots, t^m\}$ and $\Delta_t := \{(t^i, t^i(\varepsilon), t^i_{\sigma_1}, \ldots, t^i_{\sigma_k}) \mid 1 \leq i \leq m\}$.

In the following, we show that that $t = t^0$ is the only tree accepted by $\mathcal{A}_t$. Initially, we prove that $\mathcal{A}_t$ accepts $t$, by inductively defining a run $r$ of $\mathcal{A}_t$ on $t$. Set $r(\varepsilon) = t^0 = t_\varepsilon$. Assume that for $w \in \Sigma^*$, the state $r(w)$ has already been defined and it holds that $r(w) = t_w = t^j$ for some $0 \leq j \leq m$. Note that $t^j_{\sigma_i} = t_{w\sigma_i}$ for every $1 \leq i \leq k$. Since $t(w) = t_w(\varepsilon) = t^j(\varepsilon)$ and $(t^j, t^j(\varepsilon), t^j_{\sigma_1}, \ldots, t^j_{\sigma_k}) \in \Delta$, we define $r(w\sigma_i) = t^j_{\sigma_i} = t_{w\sigma_i}$. In this way, the run $r$ of $\mathcal{A}_t$ on $t$ is inductively defined, and thus $t \in \mathcal{L}(\mathcal{A}_t)$.

Next, assume that $\mathcal{A}_t$ has a run $r'$ on a tree $t'$. We will inductively show that $r' = r$ and $t' = t$. Note that $r'(\varepsilon) = t_0 = r(\varepsilon)$, since otherwise $r'$ would not be a run. The induction hypothesis is that $r'(w) = r(w) = t^j$ for some $0 \leq j \leq m$. Recall that, by construction of $r$, $r(w) = t_w$. Hence, $(r'(w), t'(w), r'(w\sigma_1), \ldots, r'(w\sigma_k)) \in \Delta$ implies that $t'(w) = t^j(\varepsilon) = t_w(\varepsilon) = t(w)$ and that $(r'(w\sigma_1), \ldots, r'(w\sigma_k)) = (t^j_{\sigma_1}, \ldots, t^j_{\sigma_k}) = (r(w\sigma_1), \ldots, r(w\sigma_k))$. Thus, for every $w \in \Sigma^*$ we have that $r'(w) = r(w)$ and $t'(w) = t(w)$ and hence $r' = r$ and $t' = t$, implying that $\mathcal{A}_t$ accepts exactly $t$ and has a unique run on it. □

The automaton $\mathcal{A}_t$ constructed in the above proof actually has a very specific syntactic shape (see Definition 9 below), which ensures that it accepts only one tree.

**Definition 9 (Representing looping tree automaton (rLTA)).** *A* representing *looping tree automaton is a looping tree automaton $\mathcal{A} = (\Sigma, P, L, \Delta, \{p_s\})$ such that $\Delta$ satisfies the following condition:*

- *for every $p \in P$, there exists a unique symbol $l_p \in L$ and a unique tuple $(p_1, \ldots, p_{|\Sigma|}) \in P^{|\Sigma|}$ such that $(p, l_p, p_1, \ldots, p_{|\Sigma|}) \in \Delta$.*

The following proposition states some obvious consequences of this definition and the proof of Proposition 1.

**Proposition 2.** *Let $\mathcal{A}$ be an rLTA and $t$ a regular tree. Then*

1. *$t$ can be represented by some rLTA $\mathcal{A}_t$.*
2. *$L(\mathcal{A})$ is a singleton set consisting of a regular tree $t_\mathcal{A}$ and $\mathcal{A}$ has a unique run $r_\mathcal{A}$ on $t_\mathcal{A}$.*

*Proof.* The first claim is immediate after observing that the automaton $\mathcal{A}_t$ introduced in the proof of Proposition 1 is an rLTA. For the second claim, completely analogously to the proof of Proposition 1, we can prove that $\mathcal{A}$ has a run $r$ on some tree $t$, and for any run $r'$ of $\mathcal{A}$ on some tree $t'$ it holds that $r' = r$ and $t' = t$.

In [28] it is shown that, given an $\mathcal{FL}_0$ concept description $C$ and a TBox $\mathcal{T}$, the tree $t_C$ encoding the tuple $\mathcal{L}_\mathcal{T}(C)$ can be represented by an rLTA.

**Theorem 1 ([28]).** *Let $C$ be an $\mathcal{FL}_0$ concept description and $\mathcal{T}$ a TBox. Then, one can construct a representing looping tree automaton that represents $t_C$ in time exponential in the size of $C$ and $\mathcal{T}$.*

In case we are given a general LTA $\mathcal{A}$, we should like to know whether it actually represents a tree (i.e., recognizes a singleton set), and if the answer is affirmative construct an rLTA that represents the same tree.

**Lemma 1.** *Let $\mathcal{A}$ be an LTA. We can decide in polynomial time whether $\mathcal{A}$ represents a tree. If $\mathcal{A}$ represents a tree $t \in T^\omega_{\Sigma,L}$, then we can construct an rLTA representing $t$ in polynomial time.*

*Proof.* Given $\mathcal{A}$, we remove superfluous states by applying the emptiness test for looping automata [12,10] and check whether $\mathcal{L}(\mathcal{A}) = \emptyset$. If this is the case, $\mathcal{A}$ does not represent a tree. Otherwise, we check whether the automaton accepts a unique tree. If the answer is affirmative, we obtain an automaton $\mathcal{A}^r$ by removing all but one transition for every state. Obviously, $\mathcal{A}^r$ is an rLTA and $\mathcal{L}(\mathcal{A}^r) \subseteq \mathcal{L}(\mathcal{A})$. If $\mathcal{A}$ represents a tree, $\mathcal{A}^r$ is the rLTA we are looking for.

Before providing the exact algorithm below, a definition is due. Claiming that an LTA has no superfluous states is the informal way of saying that the LTA is trim. An LTA $\mathcal{A}$ is called *trim* if every state can be used in some run of $\mathcal{A}$. It is easy to see that every LTA can be transformed into a trim LTA that is equivalent in the sense of having the same runs.

**Algorithm for deciding whether a given LTA represents a tree.**

Given an LTA $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$:

- Construct an equivalent trim LTA $\mathcal{A}' = (\Sigma, Q', L, \Delta', I')$ [10, Lemma 2]. If the resulting automaton has no initial states, then $L(\mathcal{A}) = \emptyset$, and thus $\mathcal{A}$ does not represent a tree.
- Otherwise, compute the binary relation $\sim$ on $Q'$ (that is inspired from automata minimization) as follows:
  - $B_0 = \{(q, q') \in Q'^2 \mid \exists (q, \sigma_1, ...), (q', \sigma_2, ...) \in \Delta' \text{ with } \sigma_1 \neq \sigma_2\}$
  - For $i = 1, 2, \ldots,$ set

$$B_i = B_{i-1} \cup \{(q, q') \in Q'^2 \mid \exists (q, \sigma, q_1, \ldots, q_k), (q', \sigma, q'_1, \ldots, q'_k) \in \Delta' \text{ and } 1 \leq i \leq k \text{ s.t. } (q_i, q'_i) \in B_{i-1}\}.$$

  The iteration becomes stable and thus terminates after $m \leq |Q'^2|$ steps. Define $\sim := Q'^2 \setminus B_m$.
- Check whether $q \sim q'$ for every $q, q' \in I'$. The answer is positive iff $\mathcal{A}$ represents a tree.

The following lemma proves correctness of the algorithm.

**Lemma 2.** *A trim LTA $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ represents a tree iff $I \neq \emptyset$ and $q \sim q'$ for every $q, q' \in I$.*

*Proof.* Assume that $\mathcal{A}$ does not represent a tree. This means that either it does not accept any tree, or it accepts more than one. In the first case, since $\mathcal{A}$ is trim, we get that $I = \emptyset$. In the second case, there are at least two trees $t_1, t_2$ accepted by $\mathcal{A}$. Let $w = \sigma_1 \ldots \sigma_n$ be a minimal word s.t. $t_1(w) \neq t_2(w)$ and $r_1, r_2$ be runs of $\mathcal{A}$ on $t_1, t_2$ respectively. Thus, we get that there are transitions $(r_1(w), t_1(w), ...), (r_2(w), t_2(w), ...) \in \Delta$ with $t_1(w) \neq t_2(w)$. By the construction in the algorithm, $(r_1(w), r_2(w)) \in B_0 \subseteq B_m$. For every proper prefix $v$ of $w$, since $t_1(v) = t_2(v)$ (by minimality of $w$) we get that $(r_1(v), r_2(v)) \in B_m$. In particular, $(r_1(\varepsilon), r_2(\varepsilon)) \in B_m$, and since $r_1(\varepsilon), r_2(\varepsilon) \in I$ the proof of this direction is complete.

For the other direction, if $I = \emptyset$ then obviously $\mathcal{A}$ does not accept any trees. Assume that $q \not\sim q'$, i.e., $(q, q') \in B_m$ for some $q, q' \in I$. Then, let $l$ be the least number such that $(q, q') \in B_l$. If $l = 0$, there exist $(q, \sigma, \ldots), (q', \sigma', \ldots) \in \Delta$ with $\sigma \neq \sigma'$, and since the automaton is trim, we get that $\mathcal{A}$ accepts at least two trees, one with root $\sigma$ and one with $\sigma'$. If $l \geq 1$, exist $(q, \sigma, q_1, \ldots, q_k), (q', \sigma, q'_1, \ldots, q'_k) \in \Delta$ with $(q_i, q'_i) \in B_{l-1}$ for some $1 \leq i \leq k$. Iterating the above argument, we get a word $w \in \Sigma^*$ (with length at most $l$) and a pair $(p, p') \in B_0$ s.t. $p$ is a $w$-successor of $q$ and $p'$ of $q'$ and, as before, we derive that $\mathcal{A}$ accepts at least two trees (with the difference existing in the node $w$ instead of the root). □

The results of this section show that we can restrict the attention to rLTAs when representing regular trees.

# 4 Using weighted looping tree automata to assign a value to a tuple of languages

Our goal is now to assign values from a (numerical or other) domain to tuples of (possibly infinite) languages that can be represented by regular trees. Consequently, we need a device that takes as input such a tree and returns a value. Weighted looping tree automata are such devices: they assign values (from a so-called semiring) to infinite trees. In the next subsection, we introduce the special type of weighted tree automata that we will use together with the necessary notions (semirings, discounting, etc.). We show how the language distances $d_1, d_2$ and the function $e_1$ introduced in the previous section can be realized using such automata. Then, we turn to the problem of how to actually compute the value assigned by such an automaton to a regular tree that is represented by an rLTA.

## 4.1 Weighted looping tree automata

In order to assign a value to a tree, weighted tree automata make use of transitions that are equipped with weights. These weights are usually elements of a semiring such that one can add and multiply weights. An extensive survey of weighted tree automata can be found in [20]. In a setting where the automata are required to work on infinite trees, the underlying semiring should admit suitable infinite sums and products [31]. In the context of infinite trees, it is also useful to employ discounting. This has been used for modeling systems with non-terminating behavior [1] in order to assign different degrees of importance to incidents that happen later in time. In our setting, discounting can be used to assign less importance to differences that occur for longer words, i.e., further down in the tree.

**Semirings.** The weight structures underlying our weighted tree automata are totally complete commutative semirings [31].

**Definition 10.** *A* semiring $\mathcal{S} = (S, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ *consists of a set $S$, two binary operations $\oplus$ and $\otimes$, and two constant elements $\mathbb{0}$ and $\mathbb{1}$ such that:*

1. *$(S, \oplus, \mathbb{0})$ is a commutative monoid,*
2. *$(S, \otimes, \mathbb{1})$ is a monoid,*
3. *multiplication distributes over addition from left and right,*
4. *$\mathbb{0} \otimes a = a \otimes \mathbb{0} = \mathbb{0}$ for all $a \in S$.*

*A semiring is called* commutative *if $a \otimes b = b \otimes a$ for all $a, b \in S$.*

Next, assume that addition can be suitably extended to infinite sums, i.e., the semiring $\mathcal{S}$ is equipped with infinitary sum operations $\bigoplus_I : S^I \to S$, for any index set $I$, such that for all $I$ and all families $(a_i \mid i \in I)$ of elements of $S$ the following hold:

$$\bigoplus_{i \in \emptyset} a_i = \mathbb{0}, \qquad \bigoplus_{i \in \{j\}} a_i = a_j, \qquad \bigoplus_{i \in \{j,k\}} a_i = a_j \oplus a_k \text{ for } j \neq k,$$

$$\bigoplus_{j \in J} \left( \bigoplus_{i \in I_j} a_i \right) = \bigoplus_{i \in I} a_i, \text{ if } \bigcup_{j \in J} I_j = I \text{ and } I_j \cap I_k = \emptyset \text{ for } j \neq k,$$

$$\bigoplus_{i \in I} (c \otimes a_i) = c \otimes \left( \bigoplus_{i \in I} a_i \right), \qquad \bigoplus_{i \in I} (a_i \otimes c) = \left( \bigoplus_{i \in I} a_i \right) \otimes c.$$

The semiring $\mathcal{S}$ together with the operations $\bigoplus_I$ is called *complete*.

A complete semiring is said to be *totally complete*, if it is endowed with countably infinite product operations satisfying for all sequences $(a_i \mid i \geq 0)$ of elements of $S$ the following conditions:

$$\bigotimes_{i\geq 0} \mathbb{1} = \mathbb{1}, \quad a_0 \otimes \bigotimes_{i\geq 0} a_{i+1} = \bigotimes_{i\geq 0} a_i, \quad \bigotimes_{i\geq 0} a_i = \bigotimes_{i\geq 0} a_i{}',$$

where $a_0{}' = a_0 \otimes \ldots \otimes a_{n_1},\ a_1{}' = a_{n_1+1} \otimes \ldots \otimes a_{n_2},\ \ldots$ for an increasing sequence $0 < n_1 < n_2 < \ldots$ of natural numbers, and

$$\bigotimes_{j\geq 1} \left( \bigoplus_{i\in I_j} a_i \right) = \bigoplus_{(i_1,i_2,\ldots)\in I_1\times I_2\times\ldots} \left( \bigotimes_{j\geq 1} a_{i_j} \right),$$

where $I_1, I_2, \ldots$ are arbitrary index sets.

A *totally commutative complete semiring* is a commutative and totally complete semiring that additionally satisfies:

$$\bigotimes_{i\geq 0} (a_i \otimes b_i) = \left( \bigotimes_{i\geq 0} a_i \right) \otimes \left( \bigotimes_{i\geq 0} b_i \right).$$

*Examples* The following semirings are totally commutative complete:

- the semiring $(\mathbb{N} \cup \{+\infty\}, +, \cdot, 0, 1)$ of natural numbers extended with positive infinity $+\infty$,
- the *tropical semiring* $Trop = (\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$ and the *arctic semiring* $Arc = (\mathbf{N} \cup \{+\infty, -\infty\}, \sup, +, -\infty, 0)$ with the binary operations extended in the natural way to infinitary operations,
- the real counterparts of the aforementioned semirings, $\mathbb{R}_{\inf} = (\mathbb{R}_{\geq 0} \cup \{+\infty\}, \inf, +, +\infty, 0)$ and $\mathbb{R}_{\sup} = (\mathbb{R}_{\geq 0} \cup \{+\infty, -\infty\}, \sup, +, -\infty, 0)$,
- the *Viterbi* semiring $([0, 1], \sup, \cdot, 0, 1)$,
- every complete distributive lattice.

All of the above examples but Viterbi can be found in [31]. To the best of our knowledge, whether the Viterbi semiring is totally commutative complete has not been investigated in the literature before. To prove that this is indeed the case, it suffices to makes the following two observations:

- It is well-known (see for example [22]) that the infinite product $\prod_{i\geq 0} a_i$ converges in case $\sum_{i\geq 0}(1 - a_i)$ converges, and is equal to 0 if $\sum_{i\geq 0}(1 - a_i) = +\infty$. Since $0 \leq a_i \leq 1$, we have that $\sum_{i\geq 0}(1 - a_i)$ either converges or is equal to $+\infty$, and thus the infinite product is well-defined.
- For any index set $I$ and any family $(a_i \mid i \in I)$ of elements in $[0, 1]$ it holds that

  $$\sup_{i\in I} a_i = e^{-\inf_{i\in I}\{-\log a_i\}},$$

  and for any sequence $(a_i \mid i \geq 0)$ of elements in $[0, 1]$

  $$\prod_{i\geq 0} a_i = e^{-\sum_{i\geq 0}(-\log a_i)}.$$

Thus, Viterbi being a totally commutative complete semiring is a corollary of $\mathbb{R}_{\text{inf}}$ being one.

Even though in Definition 1 we required $S$ to be equipped with a partial order $\leq_S$, it is often enough if a *preorder* is available. On one hand, for the relaxed instance and approximate unification problems, *strict orders*, not partial ones are utilized. Given a partial order or a preorder $\preceq$, a strict order $\prec$ is induced in the same way by setting $a \prec b \iff a \preceq b \wedge b \not\preceq a$. On the other hand, a partial order can always be derived from a preorder by considering the quotient set, identifying elements $a$ and $b$ s.t. $a \preceq b \wedge b \preceq a$.

For every semiring $\mathcal{S} = (S, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ a natural preorder $\preceq$ is induced, by setting $a \preceq b$ if there exists some $c \in S$ such that $a \oplus c = b$. Note, however, that in all the examples above, the induced preorder is actually a partial order.

**Discounting.** In the setting of semirings, discounting is defined by using semiring endomorphisms. This approach was originally used for weighted automata on infinite words by Droste and Kuske in [16], and extended to weighted automata on infinite trees by Mandrali and Rahonis [26].

**Definition 11.** *Let $\mathcal{S} = (S, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ be a semiring. A mapping $f : S \to S$ is called an* endomorphism *if $f(a \oplus b) = f(a) \oplus f(b)$ and $f(a \otimes b) = f(a) \otimes f(b)$ for all $a, b \in S$, and $f(\mathbb{0}) = \mathbb{0}$, $f(\mathbb{1}) = \mathbb{1}$. The set $End(\mathcal{S})$ of all endomorphisms of $\mathcal{S}$ is a monoid with composition $\circ$ as binary operation and the identity mapping id as unit.*

For $\mathbb{R}_{\text{sup}}$, it was proved in [16] that every endomorphism is of the form $\overline{p}(a) = p \cdot a$ for some $p \in [0, +\infty)$, and conversely, every $p \in [0, +\infty)$ defines an endomorphism of $\mathbb{R}_{\text{sup}}$ in this way. The same result can be shown for $\mathbb{R}_{\text{inf}}$ as well [17]. Finally, it is not difficult to see that, a similar result holds for the Viterbi semiring.

**Lemma 3.** *In the Viterbi semiring $([0, 1], \sup, \cdot, 0, 1)$, every endomorphism is of the form $\tilde{p}(a) = a^p$ for some $p \in [0, +\infty)$, and conversely every $p \in [0, +\infty)$ defines an endomorphism of Viterbi.*

*Proof.* Initially, observe that for every $a, b \in [0, 1]$ and for every $p \in [0, +\infty)$ it holds that

- $\tilde{p}(\sup\{a, b\}) = (\sup\{a, b\})^p = \sup\{a^p, b^p\} = \sup\{\tilde{p}(a), \tilde{p}(b)\}$,
- $\tilde{p}(a \cdot b) = (a \cdot b)^p = a^p \cdot b^p = \tilde{p}(a) \cdot \tilde{p}(b)$,
- $\tilde{p}(0) = 0^p = 0$ and
- $\tilde{p}(1) = 1^p = 1$.

Thus, every $p \in [0, +\infty)$ defines an endomorphism of Viterbi.

Next, assume that $\Phi$ is an endomorphism of Viterbi, and define

$$\phi(x) = -\log(\Phi(e^{-x})).$$

We will show that $\phi$ is an endomorphism of $\mathbb{R}_{\text{inf}}$. Indeed, for every $x, y \in \mathbb{R}_{\text{inf}}$ it holds that

$$\begin{aligned}\phi(\inf\{x, y\}) &= -\log(\Phi(e^{-\inf\{x,y\}})) = -\log(\Phi(\sup\{e^{-x}, e^{-y}\})) \\ &= -\log(\sup\{\Phi(e^{-x}), \Phi(e^{-y})\}) = \inf\{-\log(\Phi(e^{-x})), -\log(\Phi(e^{-y}))\} \\ &= \inf\{\phi(x), \phi(y)\}\end{aligned}$$

and also

$$\begin{aligned}\phi(x + y) &= -\log(\Phi(e^{-(x+y)})) = -\log(\Phi(e^{-x} \cdot e^{-y})) \\ &= -\log(\Phi(e^{-x}) \cdot \Phi(e^{-y})) = -\log(\Phi(e^{-x})) - \log(\Phi(e^{-y})) \\ &= \phi(x) + \phi(y).\end{aligned}$$

Thus, $\phi$ is indeed an endomorphism of $\mathbb{R}_{\inf}$. Consequently, by [17], we know that there exists some $p \in [0, +\infty)$, such that $\phi(x) = p \cdot x$ for every $x \in [0, +\infty)$. Hence, for every $x \in [0, +\infty)$ we get that

$$\phi(x) = p \cdot x \implies -\log(\Phi(e^{-x})) = p \cdot x \implies \Phi(e^{-x}) = e^{-p \cdot x} = (e^{-x})^p.$$

Since the image of the interval $[0, +\infty)$ under the function $e^{-x}$ is the interval $(0, 1]$, for every $a \in (0, 1]$ we have that $\Phi(a) = a^p$. Finally, since by definition of endomorphism $\Phi(0) = 0 = 0^p$, we get that $\Phi(a) = a^p$ holds in the complete interval, i.e., for every $a \in [0, 1]$. □

**Definition 12.** *Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of symbols and $\mathcal{S}$ a semiring. A* discounting *for $\Sigma$ and $\mathcal{S}$ is a tuple $\Phi \in (End(\mathcal{S}))^k$.*[2]

For a discounting $\Phi = (\phi_1, \ldots, \phi_k)$ and for every word $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n} \in \Sigma^*$, we define the endomorphism $\phi_w$ of $\mathcal{S}$ induced by $\Phi$ and $w$ as $\phi_w = \phi_{i_1} \circ \phi_{i_2} \circ \cdots \circ \phi_{i_n}$, where for $w = \varepsilon$ the empty composition is $id$.

**Weighted looping tree automata.** In the following, $\mathcal{S}$ is assumed to be a totally complete commutative semiring. An *infinitary tree series* $h$ over $L$ and $\mathcal{S}$ is a mapping $h : T^{\omega}_{\Sigma,L} \to S$. The class of all infinitary tree series over $L$ and $\mathcal{S}$ is denoted by $\mathcal{S}\langle\langle T^{\omega}_{\Sigma,L}\rangle\rangle$.

**Definition 13 (Weigthed looping tree automaton with discounting $\Phi$).** *A* weighted looping tree automaton with discounting $\Phi$ *($\Phi$-wLTA) over $\mathcal{S}$ is a tuple $\mathcal{M} = (\Sigma, Q, L, in, wt)$ where $Q$ is a finite state set, $L$ is a finite set of labels, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite set of symbols, $in : Q \to S$ is the initial distribution, and $wt : Q \times L \times Q^k \to S$ is a mapping assigning weights to the transitions of the automaton.*

Given a $\Phi$-wLTA $\mathcal{M} = (\Sigma, Q, L, in, wt)$ over $\mathcal{S}$, a *run* of $\mathcal{M}$ on a tree $t \in T^{\omega}_{\Sigma,L}$ is a mapping $r : \Sigma^* \to Q$. We denote the set of all runs of $\mathcal{M}$ on $t$ by $R_{\mathcal{M}}(t)$. Given a run $r$, we denote the transition $(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k))$ by $\overrightarrow{r}(w)$. The *weight* of the run $r$ at $w \in \Sigma^*$ is defined as $wt(r, w) := wt(\overrightarrow{r}(w))$. The *$\Phi$-weight* (or simply weight) of $r$ is defined as

$$weight(r) := in(r(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r, w)).$$

Finally, the $\Phi$-behavior (or simply behavior) of $\mathcal{M}$ is the infinitary tree series $||\mathcal{M}|| \in \mathcal{S}\langle\langle T^{\omega}_{\Sigma,L}\rangle\rangle$ whose coefficients are determined for every $t \in T^{\omega}_{\Sigma,L}$ by

$$(||\mathcal{M}||, t) := \bigoplus_{r \in R_{\mathcal{M}}(t)} weight(r).$$

If we take $\phi_i = id$ for every $i = 1, \ldots, k$, then we are left with a "normal" wLTA over $\mathcal{S}$ in the sense of [31], and thus dispense with the prefix $\Phi$- in the notation.

If $|L| = 1$, then $T^{\omega}_{\Sigma,L}$ consists of a single tree $t_{ul}$, which we will call the *unlabeled tree* since the labels are then irrelevant. In this case, we omit the label from the transitions of a $\Phi$-wLTA $\mathcal{M}$ and write $R_{\mathcal{M}}$ for its runs, omitting $t_{ul}$. Also note that then $\|\mathcal{M}\|$ is a single element of $S$ rather than a tree series.

[2] In the literature, more general forms of discounting have been introduced, where the tuple of endomorphisms to be used depends also on the label of a node, but here we restrict our attention to the simpler form of discounting introduced above.

**Expressing language distance functions.** The functions $d_1, d_2$ introduced in Example 3 and the function $e_1$ of Example 2 take a pair of languages over an alphabet $\Sigma$ as input. Thus, to represent this kind of input in a tree, we use the label set $L_2 := \{0,1\}^2$. We show that $d_2$ as well as a vital component of $d_1$ can be expressed by weighted looping automata with discounting over $\mathbb{R}_{\inf}$. The function $d_1$ itself and the function $e_1$ can be expressed using the Viterbi semiring.

*Example 5.* The first language distance described in [7] is $d_1(K,N) = 2^{-n}$ where $n = \min\{|w| \mid w \in K \,\Delta\, N\}$. We introduce a wLTA (without discounting) that, given a tree $t$ representing the tuple of languages $(K,N)$, computes the minimum $n$ rather than $2^{-n}$ itself. Given $n$, the exponentiation can be done by external computation. Consider the wLTA $\mathcal{M}_1 = (\Sigma, Q, L_2, in_1, wt_1)$ over $\mathbb{R}_{\inf} = (\mathbb{R}_{\geq 0} \cup \{+\infty\}, \inf, +, +\infty, 0)$, where $Q = \{q_0, q_1\}$, $in_1(q_0) = +\infty$, $in_1(q_1) = 0$ and

$$wt_1(q, l, p_1, \ldots, p_k) = \begin{cases} 1 & \text{if } q = q_1,\ l \in \{(0,0),(1,1)\},\ p_i = q_1 \text{ for some } 1 \leq i \leq k \\ & \qquad \text{and } p_j = q_0 \text{ for } j \neq i \\ 0 & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\},\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ 0 & \text{if } q = q_0,\ l \in \{0,1\}^2,\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ +\infty & \text{otherwise} \end{cases}$$

Intuitively, each run using only transitions with non-infinite weights selects one path in the tree, which it labels with $q_1$ until an element in the symmetric difference is found. The transitions up to this point in the selected path receive weight 1, and all other transitions have weight 0. Thus, adding up the weights (with the multiplication $\otimes = +$ of $\mathbb{R}_{\inf}$) gives us the distance from the root to the node where the difference was detected, i.e., the length of the word in the symmetric difference (or $+\infty$ in case no difference is found on the chosen path). By building the infimum over all runs, the length of the shortest word in the symmetric difference is found.

*Example 5'.* Actually, we can compute the exact value of $d_1$ by making use of the Viterbi semiring. Consider the wLTA $\mathcal{M}_1' = (\Sigma, Q, L_2, in_1', wt_1')$ over the Viterbi semiring $([0,1], \sup, \cdot, 0, 1)$ where $Q = \{q_0, q_1\}$, $in_1'(q_0) = 0$, $in_1'(q_1) = 1$ and

$$wt_1'(q, l, p_1, \ldots, p_k) = \begin{cases} \frac{1}{2} & \text{if } q = q_1,\ l \in \{(0,0),(1,1)\},\ p_i = q_1 \text{ for some } 1 \leq i \leq k \\ & \qquad \text{and } p_j = q_0 \text{ for } j \neq i \\ 1 & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\},\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ 1 & \text{if } q = q_0,\ l \in \{0,1\}^2,\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that this automaton works completely analogously to the previous one, computing $(\frac{1}{2})^n$ instead of $n$.

*Example 6.* The second distance described in [7] is $d_2(K,N) = \mu(K \,\Delta\, N)$, where $\mu(M) = \frac{1}{2}\sum_{w \in M}(2|\Sigma|)^{-|w|}$. We introduce a $\Phi$-wLTA that, given a tree $t$ representing the tuple of languages $(K,N)$, computes $\mu(K \,\Delta\, N)$. Consider the $\Phi$-wLTA $\mathcal{M}_2 = (\Sigma, Q, L_2, in_2, wt_2)$ over $\mathbb{R}_{\inf}$ where $Q = \{q_0, q_1\}$, $in_2(q_0) = in_2(q_1) = 0$ and

$$wt_2(q, l, p_1, \ldots, p_k) = \begin{cases} 0 & \text{if } q = q_0,\ l \in \{(0,0),(1,1)\} \\ \frac{1}{2} & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\} \\ +\infty & \text{otherwise} \end{cases}$$

Finally, the discounting $\Phi = (\phi_1, \ldots, \phi_k)$ is defined as $\phi_i = \overline{\frac{1}{2|\Sigma|}}$ for every $i = 1, \ldots, k$, where $\overline{\frac{1}{2|\Sigma|}}(a) = \frac{1}{2|\Sigma|} \cdot a$ for $a \in \mathbb{R}_{\geq 0}$ and $\overline{\frac{1}{2|\Sigma|}}(+\infty) = +\infty$.

It is easy to see that there is a unique run $r_0$ with non-infinite weight, the one that assigns $q_0$ to the nodes labeled with $(0,0)$ or $(1,1)$, i.e., words that do not belong to $K \,\Delta\, N$, and $q_1$ to the ones labeled with $(1,0)$ or $(0,1)$, i.e., words that belong to $K \,\Delta\, N$. The discounting multiplies

the weight of every word $w \in \Sigma^*$ with $(\frac{1}{2|\Sigma|})^{|w|}$. If the word does not belong to $K \,\Delta\, N$ it gets a zero weight. If it does belong to $K \,\Delta\, N$, it gets weight $\frac{1}{2}$. "Multiplying" in $\mathbb{R}_{\text{inf}}$ (i.e., summing over all words in $\Sigma^*$), we obtain exactly $\mu(K \,\Delta\, N)$ as weight for this run.

*Example 7.* Recall the function $e_1$ from Example 2 where $e_1(K, N) = 1$ if $N \subseteq K$, and 0 otherwise. As was the case for $d_1$, given a tree $t$ representing the tuple of languages $(K, N)$ we can either consider a wLTA over $\mathbb{R}_{\text{inf}}$ that helps computing $e_1(K, N)$, or make use of a wLTA over the Viterbi semiring to compute the exact value. In this example, we examine the second case.

Consider the wLTA $\mathcal{M}_e = (\Sigma, Q, L_2, in_e, wt_e)$ over the Viterbi semiring $([0, 1], \sup, \cdot, 0, 1)$ where $Q = \{q_0\}$, $in(q_0) = 1$ and

$$wt(q_0, l, p_1, \ldots, p_k) = \begin{cases} 1 & \text{if } l \in \{(0,0), (1,0), (1,1)\} \\ 0 & \text{otherwise} \end{cases}$$

Note that, since $|Q| = 1$, there is exactly one run $r$ of $\mathcal{M}_e$ on $t$. The intuition behind this construction is the following: we have that $N \not\subseteq K$ iff $\exists w \in \Sigma^*$ such that $w \in N \wedge w \notin K$ iff $\exists w \in \Sigma^*$ such that $t(w) = (0, 1)$. In other words, if there exists a node of $t$ labeled with $(0, 1)$ we have that $N \not\subseteq K$. From the automaton point of view, a node labeled with $(0, 1)$ requires a transition with weight 0, and thus $weight(r) = 0$. On the other hand, if only the labels $(0, 0), (1, 0)$ and $(1, 1)$ appear, implying that $N \subseteq K$, every transition has weight 1, and thus $weight(r) = 1$. Hence, by computing $(||\mathcal{M}||, t)$ we know $e_1(K, N)$.

### 4.2 Computing the behavior on regular trees

Given a $\Phi$-wLTA $\mathcal{M}$ over a semiring $\mathcal{S}$ and an rLTA $\mathcal{A}$ representing a regular tree $t$, we want to compute the behavior of $\mathcal{M}$ on $t$, i.e., $(||\mathcal{M}||, t)$. In a first step, we reduce this problem to the problem of computing the behavior of a $\Phi$-wLTA on the unlabeled tree. To be more precise, we combine the two automata $\mathcal{M}$ and $\mathcal{A}$ into a single $\Phi$-wLTA $\mathcal{M}_\mathcal{A}$ that works on the unlabeled tree $t_{ul}$ such that $(||\mathcal{M}||, t) = (||\mathcal{M}_\mathcal{A}||, t_{ul})$.

**Theorem 2.** *Given a $\Phi$-wLTA $\mathcal{M} = (\Sigma, Q, L, in, wt)$ over $\mathcal{S}$ and an rLTA $\mathcal{A} = (\Sigma, P, L, \Delta, \{p_s\})$ representing a regular tree $t$, one can construct in polynomial time a $\Phi$-wLTA $\mathcal{M}_\mathcal{A}$ over $\mathcal{S}$ working on the unlabeled tree $t_{ul}$ such that $(||\mathcal{M}||, t) = (||\mathcal{M}_\mathcal{A}||, t_{ul})$.*

*Proof.* Let $\mathcal{S} = (S, \oplus, \otimes, \mathbb{0}, \mathbb{1})$. By the definition of rLTAs, for every state $p \in P$ there exists a unique letter $l_p \in L$ such that $(p, l_p, \ldots) \in \Delta$. Additionally, by Proposition 2 it holds that $\mathcal{A}$ has a unique run, say $\theta$, on $t$. For simplicity, for every $w \in \Sigma^*$ we denote $\theta(w) \in P$ by $p_w$.

We define the $\Phi$-wLTA $\mathcal{M}_\mathcal{A} = (Q \times P \times L, \Sigma, in', wt')$ over $\mathcal{S}$ as follows:

$$in'(q, p, l) := \begin{cases} in(q) & \text{if } p = p_s \text{ and } l = l_{p_s} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

$$wt'\big((q_0, p_0, l_0), (q_1, p_1, l_1), \ldots, (q_k, p_k, l_k)\big) := \begin{cases} wt(q_0, l_0, q_1, \ldots, q_k) & \text{if } (p_0, l_0, p_1, \ldots, p_k) \in \Delta \\ \mathbb{0} & \text{otherwise} \end{cases}$$

To prove that $(\|\mathcal{M}\|, t) = (\|\mathcal{M}_\mathcal{A}\|, t_{ul})$, it is sufficient to show that there exists an injection $\tau : R_\mathcal{M}(t) \to R_{\mathcal{M}_\mathcal{A}}$ such that $weight(r) = weight(\tau(r))$ for every $r \in R_\mathcal{M}(t)$ and $weight(r') = \mathbb{0}$ for every $r' \in R_{\mathcal{M}_\mathcal{A}} \setminus im(\tau)$, where $im(\tau)$ stands for the *image* set of the mapping $\tau$.

More precisely, the injection is defined as follows. Given a run $r \in R_{\mathcal{M}}(t)$, we define $\tau(r) = r'$ by setting $r'(w) = (r(w), p_w, l_{p_w})$. We have $in'(r'(\varepsilon)) = in'(r(\varepsilon), p_\varepsilon, l_{p_\varepsilon}) = in(r(\varepsilon))$,xx and for all $w \in \Sigma^*$:

$$\begin{aligned} wt(r', w) &= wt'(\overrightarrow{r'}(w)) = wt'(r'(w), r'(w\sigma_1), \ldots, r'(w\sigma_k)) \\ &= wt'((r(w), p_w, l_{p_w}), (r(w\sigma_1), p_{w\sigma_1}, l_{p_{w\sigma_1}}), \ldots, (r(w\sigma_k), p_{w\sigma_k}, l_{p_{w\sigma_k}})) \\ &= wt(r(w), l_{p_w}, r(w\sigma_1), \ldots, r(w\sigma_k)) = wt(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \\ &= wt(r, w). \end{aligned}$$

Thus, we obtain

$$\begin{aligned} weight(r') &= in'(r'(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r', w)) \\ &= in(r(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r, w)) = weight(r). \end{aligned}$$

Now suppose that $r' \in R_{\mathcal{M}_\mathcal{A}} \setminus im(\tau)$. In other words, for every $r \in R_{\mathcal{M}}(t)$, $r' \neq \tau(r)$, and hence

$$\exists z \in \Sigma^*,\ r'(z) \neq (r(z), p_z, l_{p_z}) \tag{5}$$

From $r'$ we define three mappings $r_0 : \Sigma^* \to Q$, $p_0 : \Sigma^* \to P$, $l_0 : \Sigma^* \to L$ by setting $r'(w) = (r_0(w), p_0(w), l_0(w))$ for every $w \in \Sigma^*$. Obviously, $r_0 \in R_{\mathcal{M}}(t)$ (since any mapping from $\Sigma^*$ to $Q$ is a run of $\mathcal{M}$ on $t$). Then, from (5), we get that $\exists z \in \Sigma^*$ such that $p_0(z) \neq p_z$ or $l_0(z) \neq l_{p_z}$ (otherwise it would be the case that $r' = \tau(r_0)$), and assume without loss of generality that $z$ has minimal length. We distinguish two cases.

- $z = \varepsilon$. This implies that $p_0(\varepsilon) \neq p_\varepsilon$ or $l_0(\varepsilon) \neq l_{p_\varepsilon}$. In both cases, $in(r'(\varepsilon)) = \mathbb{0}$ and thus $weight(r') = \mathbb{0}$, since $\mathbb{0} \otimes a = \mathbb{0}$ for all $a \in S$.
- $z = v\sigma_i$. This implies that $p_z \neq p_0(z)$ or $l_{p_z} \neq l_0(z)$. In the first case, we have that $(p_v, l_{p_v}, \ldots, p_0(z), \ldots) \neq (p_v, l_{p_v}, \ldots, p_{v\sigma_i}, \ldots)$. Since $(p_v, l_{p_v}, \ldots, p_{v\sigma_i}, \ldots)$ is the unique $p_v$-transition, we get that $(p_v, l_{p_v}, \ldots, p_0(z), \ldots) \notin \Delta$, yielding $wt(r', v) = \mathbb{0}$.
  In the second case, $(p_z, l_0(z), \ldots) \neq (p_z, l_{p_z}, \ldots)$ and thus $(p_z, l_0(z), \ldots) \notin \Delta$, yielding $wt(r', z) = \mathbb{0}$. In both cases, we get that $weight(r') = \mathbb{0}$

Finally, since $(S, \oplus, \mathbb{0})$ is a commutative monoid, we get that:

$$\begin{aligned} (\|\mathcal{M}_\mathcal{A}\|, t_{ul}) &= \bigoplus_{r' \in R_{\mathcal{M}_\mathcal{A}}} weight(r') = \bigoplus_{r' \in im(\tau)} weight(r') \\ &= \bigoplus_{r \in R_{\mathcal{M}}(t)} weight(\tau(r)) = \bigoplus_{r \in R_{\mathcal{M}}(t)} weight(r) \\ &= (\|\mathcal{M}\|, t) \end{aligned}$$

□

Thus, it remains to show how the behavior of a $\Phi$-wLTA working on the unlabeled tree can be computed. For wLTAs (without discounting) over complete distributive lattices this was done in [11]. In the next section, we show how the behavior of a $\Phi$-wLTA over the semiring $\mathbb{R}_{\text{inf}}$ can be computed.

## 5 Computing the behavior on the unlabeled tree in $\mathbb{R}_{\text{inf}}$

Concentrating on $\mathbb{R}_{\text{inf}}$ is motivated, on the one hand, by the fact that our motivating examples (the distance functions $d_1$ and $d_2$ and the similarity function $e_1$) can be expressed using wLTA

with discounting over this semiring. On the other hand, discounting for this semiring is well-understood [16] and nicely behaved. Note, however, that our algorithms can be extended to the Viterbi semiring. Still, in order to get analogous complexity results, further computability and/or precision considerations have to be taken into account (see comments at the end of each section).

Recall that, for $\mathbb{R}_{\inf}$, all endomorphisms are of the form $\overline{p}(a) = p \cdot a$ for $p \in \mathbb{R}_{\geq 0}$, and thus the discounting is of the form $\Phi = (\overline{p_1}, \ldots, \overline{p_k})$. Given $w = \sigma_{i_1} \ldots \sigma_{i_m} \in \Sigma^*$, we set $p_w = p_{i_1} \cdot \ldots \cdot p_{i_m}$ where the empty product (case $w = \varepsilon$) is 1. Then $\phi_w(a) = \phi_{i_1} \circ \cdots \circ \phi_{i_m}(a) = p_{i_1} \cdot \ldots \cdot p_{i_m} \cdot a = \overline{p_w}(a)$, and thus $\phi_w = \overline{p_w}$. It is easy to see that, for $p > 0$, $\overline{p}$ distributes over inf and $\sum$. In the following, we assume that $p_i \neq 0$ for $i = 1, \ldots, k$, and we will write $p_w \cdot a$ instead of $\phi_w(a)$.

A *q-run* $r$ of $\mathcal{M}$ is a run with $r(\varepsilon) = q$. We denote the set of all $q$-runs of $\mathcal{M}$ as $R(q)$. The *running weight* of a $q$-run is defined like its weight, but without taking the initial distribution into account, i.e., $rweight(r) := \sum_{w \in \Sigma^*} p_w \cdot wt(r, w)$, and thus $weight(r) = in(q) + rweight(r)$. Consequently, if we define

$$\mu(q) := \inf_{r \in R(q)} rweight(r) \quad \text{(for every } q \in Q\text{)}$$

then $(\|\mathcal{M}\|, t_{ul}) = \min_{q \in Q} \{in(q) + \mu(q)\}$. Hence, in order to compute the behavior of $\mathcal{M}$ on $t_{ul}$, it suffices to calculate the values $\mu(q)$ for all $q \in Q$.

The following lemma provides recursive equations that are useful to achieve this goal.

**Lemma 4.** *For every state $q \in Q$ it holds that*

$$\mu(q) = \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \mu(q_i) \right\}.$$

*Proof.*

$$\begin{aligned}
\mu(q) &= \inf_{r \in R(q)} rweight(r) = \inf_{r \in R(q)} \sum_{w \in \Sigma^*} p_w \cdot wt(r, w) \\
&= \inf_{r \in R(q)} \left\{ p_\varepsilon \cdot wt(r, \varepsilon) + \sum_{w \in \Sigma^+} p_w \cdot wt(r, w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{w \in \Sigma^+} p_w \cdot wt(r, w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{w \in \Sigma^*} p_{\sigma_1 w} \cdot wt(r, \sigma_1 w) + \cdots + \sum_{w \in \Sigma^*} p_{\sigma_k w} \cdot wt(r, \sigma_k w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{i=1}^{k} \sum_{w \in \Sigma^*} p_{\sigma_i w} \cdot wt(r, \sigma_i w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \inf_{\substack{(r_1, \ldots, r_k) \in \\ R(q_1) \times \cdots \times R(q_k)}} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r_i, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \inf_{\substack{(r_1, \ldots, r_k) \in \\ R(q_1) \times \cdots \times R(q_k)}} \sum_{i=1}^{k} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r_i, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} \inf_{r \in R(q_i)} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r, w) \right\}
\end{aligned}$$

$$= \min_{(q_1,\ldots,q_k)\in Q^k} \left\{ wt(q,q_1,\ldots,q_k) + \sum_{i=1}^{k} p_i \cdot \inf_{r\in R(q_i)} \sum_{w\in\Sigma^*} p_w \cdot wt(r,w) \right\}$$
$$= \min_{(q_1,\ldots,q_k)\in Q^k} \left\{ wt(q,q_1,\ldots,q_k) + \sum_{i=1}^{k} p_i \cdot \mu(q_i) \right\}$$

□

Note that the exact same computations can be made for the Viterbi semiring $([0,1], \sup, \cdot, 0, 1)$, with the difference that $\inf, \min, +, \sum, \infty, 0$ are replaced by $\sup, \max, \cdot, \prod, 0, 1$ respectively, and that endomorphisms are of the form $\tilde{p}(a) = a^p$. Thus, we would obtain equations of the form

$$\mu(q) = \max_{(q_1,\ldots,q_k)\in Q^k} \left\{ wt(q,q_1,\ldots,q_k) \cdot \prod_{j=1}^{k} \mu(q_i)^{p_j} \right\}. \tag{6}$$

Our approach for computing the values $\mu(q)$ depends on the kind of discounting used.

### 5.1 Behavior for nondecreasing discounting

In this section we assume that the discounting is *nondecreasing*, i.e., $p_i \geq 1$ for all $i = 1, \ldots, k$. Note that absence of discounting corresponds to the special case where $p_i = 1$ for all $i = 1, \ldots, k$.

If the discounting is nondecreasing, then we have for every run $r \in R_{\mathcal{M}}$ that

$$rweight(r) = \sum_{w\in\Sigma^*} p_w \cdot wt(r,w) \geq \sum_{w\in\Sigma^*} wt(r,w),$$

where in the latter infinite sum only finitely many distinct non-negative real numbers occur. Consequently, this sum (and thus the original sum as well) is a finite number iff only 0 is used infinitely often in the sum. Therefore, a run $r$ has finite weight iff, from a certain depth on, it has only zero-weight transitions. Consequently, we can restrict our attention to deciding for each state $q$ whether such a (finite weight) $q$-run exists, and compute the smallest weight among all of them.

The first step consists of computing the set of states in $Q$ that admit a run with only zero-weight transitions. Clearly, these are exactly the states $q$ for which $\mu(q) = 0$. By keeping only transitions with weight 0 and then applying the emptiness test for LTAs [12] to the resulting automaton, these states can easily be computed.

More precisely, the computation can be done as follows. Let $\Delta_0 \subseteq Q^{k+1}$ be the set containing only the transitions in $\mathcal{M}$ with zero weight:

$$\Delta_0 := \{(q,q_1,\ldots,q_k) \in Q^{k+1} \mid wt(q,q_1,\ldots,q_k) = 0\}$$

and $B_0$ the subset of $Q$ containing all the states that have no transition in $\Delta_0$, i.e.,

$$B_0 := \{q \in Q \mid \forall (q_1,\ldots,q_k) \in Q^k.\ (q,q_1,\ldots,q_k) \notin \Delta_0\}$$

Then, we define the following iteration for $i \geq 0$:

$$B_{i+1} := B_i \cup \{q \in Q \mid \forall (q,q_1,\ldots,q_k) \in \Delta_0.\ \exists i.\ q_i \in B_i\}$$

The iteration becomes stable after at most $\ell \leq |Q|$ steps. The set $Q_\ell = Q \setminus B_\ell$ is then the set of states that admit a run with only zero-weight transitions, as the following lemma shows.

**Lemma 5.** $q \in Q_\ell \iff \exists r \in R(q)$ *that has only transitions with weight* 0.

*Proof.* Let $q \in Q_\ell$. This means that there is a transition $(q, q_1, \ldots, q_k) \in \Delta_0$ such that $(q_1, \ldots, q_k) \in (Q_\ell)^k$. Iterating this argument for the successor states one can build the wanted run $r$.

For the opposite direction, suppose that $q \notin Q_\ell$ and thus $q \in B_\ell$. Assume that $j$ is the least index such that $q \in B_j$. By induction on $j$ we will prove that there is no $q$-run that has only zero-transitions, i.e., transitions from $\Delta_0$. If $j = 0$, i.e., $q \in B_0$, then there is no zero-weight transition starting from $q$, and thus no $q$-run with only zero-weight transitions. If $j > 0$, this implies that for every $(q, q_1, \ldots, q_k) \in \Delta_0$ exists some $i$ such that $q_i \in B_{j-1}$. By the induction hypothesis, there is no $q_i$-run with only zero-weight transitions, and thus the same holds for $q$. □

The following lemma is a straightforward consequence of the previous one and the definition of $\mu(q)$.

**Lemma 6.** $q \in Q_\ell \iff \mu(q) = 0$

*Proof.* Suppose that $q \in Q_\ell$. Then $\exists r \in R(q)$ such that $rweight(r) = 0$, and thus $\mu(q) = 0$. Conversely, $q \notin Q_\ell$ implies that $\forall r \in R(q)$ there is a transition with non-zero weight. Let $a_0$ be the least non-zero weight among all transitions in the automaton. Then, $\forall r \in R(q)$, $\sum_{w \in \Sigma^*} wt(r, w) \geq a_0$, and thus $\mu(q) \geq a_0 > 0$. □

Summing up, the above construction gives us the next lemma.

**Lemma 7.** *The set of states* $Q_{\mu=0} := \{q \in Q \mid \mu(q) = 0\}$ *can be computed in polynomial time.*

A run with finite weight does not use a transition with weight $+\infty$ and below a certain depth in the tree it contains only states that belong to $Q_{\mu=0}$. Thus, the states used in the run must have access to states in $Q_{\mu=0}$ through transitions with finite weight. To be more precise, define the set $Q_{acc}$ of states that *have access to* $Q_{\mu=0}$ to be the least subset of $Q$ such that $(i)$ $Q_{\mu=0} \subseteq Q_{acc}$ and $(ii)$ if $q_i \in Q_{acc}$ for every $i = 1, \ldots, k$ and $wt(q, q_1, \ldots, q_k) \neq +\infty$ then $q \in Q_{acc}$. States $q$ that have access to $Q_{\mu=0}$ have a $q$-run with finite running weight, and hence $\mu(q) < +\infty$. If $q$ does not have access to $Q_{\mu=0}$, then $\mu(q) = +\infty$. By using an approach inspired by Dijkstra's shortest path algorithm, we can compute the states that have access to $Q_{\mu=0}$ together with their $\mu$-value in polynomial time.

Initially, note that in case $Q_\ell = \emptyset$, no state has access to $Q_\ell$ and thus $\mu(q) = \infty$ for all $q \in Q$.

To compute $\mu(q)$ for all the states $q$ we use the following algorithm. Set $S_0 = Q_\ell$ and consider the function

$$m^0(q) = \begin{cases} 0, & \text{if } q \in S_0 \\ \min\limits_{(q_1,\ldots,q_k) \in (S_0)^k} wt(q, q_1, \ldots, q_k), & \text{otherwise} \end{cases}$$

Next, for $i > 0$, iteratively do the following:

- $S_i := S_{i-1} \cup \{s_i\}$, for some $s_i = \arg\min_{q \notin S_{i-1}} m^{i-1}(q)$.
- For all $q \notin S_i$, update their $m$ value:

$$m^i(q) := \min \left\{ m^{i-1}(q), \min_{(q_1,\ldots,q_k) \in (S_i)^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{j=1}^{k} p_j \cdot m^{i-1}(q_j) \right\} \right\}$$

  while for all $q \in S_i$, $m^i(q) := m^{i-1}(q)$.

This iteration terminates after $f = |Q \setminus Q_\ell|$ steps. Moreover, since by Lemma 4 we know that $\mu(q)$ corresponds to:

$$\mu(q) = \min_{(q_1,\ldots,q_k)\in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{j=1}^{k} p_j \cdot \mu(q_j) \right\},$$

based on the definition of $m^i$ and the fact that $(S_i)^k \subseteq Q^k$ it can be shown by induction on $i$ that:

$$\mu(q) \leq m^i(q), \text{ for all } q \in Q \text{ and } i \geq 0 \tag{7}$$

We now show that, upon termination, $m^f(q) = \mu(q) = \inf_{r\in R(q)} rweight(r)$ holds for all $q \in Q$. From this, the behavior of $\mathcal{M}$ can be directly computed, since it can then be expressed as:

$$\|\mathcal{M}\| = \min_{q\in Q} \big(in(q) + m^f(q)\big)$$

**Lemma 8.** *For all $i \geq 0$ and $s \in S_i$, it holds that:*

1. *$\mu(s) \leq \mu(q)$ for all $q \notin S_i$, and*
2. *$\mu(s) = m^i(s)$.*

*Proof.* We prove our claims by induction on $i$.

*Base case.* $i = 0$. Since $s \in S_0$ and $S_0 = Q_\ell$, by Lemma 6 and the definition of $m^0$, it follows that $\mu(s) = m^0(s) = 0$. In addition, let $q \notin S_0$ be a state in $Q$. Since $\mu(q) \in (\mathbb{R}_{\geq 0} \cup \{\infty\})$, this means that $\mu(s) \leq \mu(q)$.

*Induction step.* We show our claims hold for all $i > 0$, based on the assumption that they hold for all numbers smaller than $i$.

Let $S_i = S_{i-1} \cup \{s_i\}$. The application of induction yields $\mu(s) = m^{i-1}(s)$ and $\mu(s) \leq \mu(s_i)$ for all $s \in S_{i-1}$. Since $m^i(s) = m^{i-1}(s)$ for all $s \in S_{i-1}$, this means that $\mu(s) = m^i(s)$. Hence, it remains to show that the claims hold for $s_i$.

1. We want to show that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$. Suppose for a contradiction that there exists $s' \notin S_i$ such that $\mu(s') < \mu(s_i)$. Without loss of generality, $s'$ is selected such that $\mu(s') \leq \mu(q)$ for all $q \notin S_i$. Based on Lemma 4, $\mu(s')$ can be expressed as:

$$\mu(s') = wt(s', q_1^0, \ldots, q_k^0) + \sum_{j=1}^{k} p_j \cdot \mu(q_j^0), \tag{8}$$

   for some tuple $(q_1^0, \ldots, q_k^0) \in Q^k$. Let us first show that $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$. Suppose on the contrary that $q_j^0 \notin S_{i-1}$ for some $1 \leq j \leq k$. Then, it clearly holds that $0 < \mu(s') \leq \mu(q_j^0)$ by the way $s'$ was selected. We distinguish the following two cases:
   - $wt(s', q_1^0, \ldots, q_k^0) > 0$.Consequently, $\mu(s') > \mu(q_j^0)$ must hold[3], a contradiction.
   - $wt(s', q_1^0, \ldots, q_k^0) = 0$. To avoid the same contradiction as before, it must be the case that $\mu(q_\lambda^0) = 0$ for all $\lambda \neq j$. Hence, $\mu(s') = \mu(q_j^0)$, and $q_j^0$ could have been chosen as $s'$. By iterating this argument, while assuming the corresponding transition has zero-weight, we will end up with $s'$ having a run with only zero-weight transitions, i.e., belonging to $Q_{\mu=0}$[4]. Thus, we obtain a contradiction since $s' \notin S_0 = Q_\ell = Q_{\mu=0}$.

[3] Recall that $p_j \geq 1$.

[4] This iteration terminates in at most $| Q \setminus Q_\ell |$ steps.

Then, having $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$ implies that $\mu(q_j^0) = m^{i-1}(q_j^0)$ for all $1 \leq j \leq k$ (by induction). Recall that $m^i(s')$ corresponds to the expression:

$$m^i(s') = \min \left\{ m^{i-1}(s'), \min_{(q_1,\ldots,q_k)\in(S_i)^k} \left\{ wt(s', q_1, \ldots, q_k) + \sum_{j=1}^{k} p_j \cdot m^{i-1}(q_j) \right\} \right\}$$

Since $(S_{i-1})^k \subseteq (S_i)^k$, this means that the value corresponding to the inner minimization in the previous expression is not greater than $\mu(s')$. Therefore, using (7) we obtain:

$$m^i(s') = \mu(s') < \mu(s_i) \leq m^i(s_i) = m^{i-1}(s_i) \tag{9}$$

Moreover, if $m^i(s') < m^{i-1}(s')$ holds, this must have been a direct consequence of introducing $s_i$ to form $S_i$. In other words, $m^i(s')$ is updated by using the value $m^{i-1}(s_i)$, which means that $m^{i-1}(s_i) \leq m^i(s')$. The latter is obviously not consistent with (9). Therefore, in order to still be consistent with $\mu(s') < \mu(s_i)$, the equality $m^i(s') = m^{i-1}(s')$ must hold. But then, it follows that $m^{i-1}(s') < m^{i-1}(s_i)$ which is a contradiction since $s_i$ was selected to obtain $S_i$. Thus, we can conclude that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$.

2. Consider $\mu(s_i)$ expressed as in (8) (substitute $s'$ by $s_i$). Since we have just shown that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$, similarly as before, it can be proved that $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$ holds for $s_i$ as well. Then, the same argument used in 1. yields that $m^i(s_i) \leq \mu(s_i)$. Thus, $m^i(s_i) = \mu(s_i)$ holds (using (7)).

□

Summing up, computing $Q_\ell$ requires quadratic time and computing $m^f(q)$ for every $q \in Q$ requires cubic time. Since the behavior of $\mathcal{M}$ on $t_{ul}$ can easily be computed from the values $\mu(q)$ for $q \in Q$, this yields the following theorem.

**Theorem 3.** *The behavior of a $\Phi$-wLTA with nondecreasing discounting $\Phi$ over $\mathbb{R}_{\inf}$ on the unlabeled tree can be computed in polynomial time.*

Note that a similar algorithm can be utilized for the behavior over the Viterbi semiring. As was done for obtaining Equation (6), replace $\inf, \min, +, \sum, \infty, 0$ by $\sup, \max, \cdot, \prod, 0, 1$ respectively. Also recall that endomorphisms are of the form $\tilde{p}(a) = a^p$. Other than this, the algorithm and the proof of its correctness is the same. However, since we might have to compute numbers as small as $x^{p^{|Q|}}$, the algorithm might take exponential time because of their computation.

### 5.2 Behavior for contracting discounting

In this section, we assume a *contracting* discounting, i.e., $p < \frac{1}{k}$, where $k = |\Sigma|$ and $p = \max_{i=1,\ldots,k} p_i$.

Recall that it suffices to compute the value $\mu(q)$ for every $q \in Q$. To achieve this, we generalize the approach used in [7] for the special case of $d_2$. Let $Q = \{q_1, \ldots, q_n\}$. For each $q_i \in Q$, the unknown value $\mu(q_i)$ is associated to a variable $x_i$. Additionally, let $I = \{1, \ldots, n\}$. Then, Lemma 4 states that $(\mu(q_1), \ldots, \mu(q_n))$ is a solution of the following system of equations:

$$x_i = \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \right\} \tag{10}$$

Before we continue with solving this system of equations, we recall some notions from Metric Topology. Formal definitions can be found in any book on the subject, for example [13,23].

A metric space $(X, d)$ consists of a set $X$ equipped with a *metric* $d$, i.e., a function that assigns a non-negative number to any two elements of $X$ (and satisfies some extra properties). The Chebyshev distance, $d_\infty$, is a metric on $\mathbb{R}^n$ which for $\boldsymbol{a} = (a_1, \ldots, a_n)$, $\boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$ is defined as

$$d_\infty(\mathbf{a}, \mathbf{b}) = \max_{i=1,\ldots,n} |a_i - b_i|.$$

It is well known that the metric space $(\mathbb{R}^n, d_\infty)$ is complete.

**Definition 14.** *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 $d(f(a), f(b)) \leqslant \lambda d(a, b)$ for any $a, b \in X$.*

Since we use contracting discounting, Banach's fixed point theorem can be used to show that the system (10) has a *unique* solution in $\mathbb{R}$.

**Theorem 4 (Banach's Fixed Point Theorem [13]).** *Let $(X, d)$ be a complete metric space and a function $f : X \to X$ be a contraction on $X$. Then there exists a unique $a \in X$ such that $f(a) = a$.*

For all $q_i \in Q$, we define a function $f_i : \mathbb{R}^n \to \mathbb{R}$ as follows:

$$f_i(a_1, \ldots, a_n) = \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot a_{i_j} \right\}$$

Next, we define the vector function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ as:

$$f(a_1, \ldots, a_n) = (f_1(a_1, \ldots, a_n), \ldots, f_n(a_1, \ldots, a_n))$$

Clearly, a vector $\mathbf{a} = (a_1, \ldots, a_n)$ is a solution of the system of equations in (10) iff it is a fixed point of $f$. Thus, it is enough to show that $f$ is indeed a contraction on a complete metric space $(\mathbb{R}^n, d)$.

**Lemma 9.** *The function $f$ defined above is a contraction on $(\mathbb{R}^n, d_\infty)$.*

*Proof.* Given $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ and $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$, recall that $d_\infty(\mathbf{a}, \mathbf{b}) = \max_{i\in I} |a_i - b_i|$.

For every $i \in I$ we have:

$$\begin{aligned} f_i(\mathbf{a}) &= \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot a_{i_j} \right\} \\ &= wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} \end{aligned}$$

for a particular $(i_1^0, \ldots, i_k^0) \in I^k$, and likewise

$$f_i(\mathbf{b}) = wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1}.$$

Since for $\mathbf{a}$, the minimum is achieved for $(i_1^0, \ldots, i_k^0)$, it holds that:

$$wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} \leq wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^1}.$$

Without loss of generality, assume that $f_i(\mathbf{b}) \leq f_i(\mathbf{a})$. Thus, we have:

$$\begin{aligned}
|f_i(\mathbf{a}) - f_i(\mathbf{b})| &= f_i(\mathbf{a}) - f_i(\mathbf{b}) \\
&= wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} - \left( wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1} \right) \\
&\leq wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^1} - \left( wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1} \right) \\
&= \sum_{j=1}^{k} p_j \cdot (a_{i_j^1} - b_{i_j^1}) \\
&\leq \sum_{j=1}^{k} \max_{i=1,\ldots,k} p_i \cdot \max_{i \in I} |a_i - b_i| \\
&= k \cdot p \cdot \max_{i \in I} |a_i - b_i| = k \cdot p \cdot d_\infty(\mathbf{a}, \mathbf{b})
\end{aligned}$$

Overall, we get that $|f_i(\mathbf{a}) - f_i(\mathbf{b})| \leq k \cdot p \cdot \ d_\infty(\mathbf{a}, \mathbf{b})$ for every $i \in I$, and thus

$$d_\infty(f(\mathbf{a}), f(\mathbf{b})) = \max_{i \in I} |f_i(\mathbf{a}) - f_i(\mathbf{b})| \leq k \cdot p \cdot \ d_\infty(\mathbf{a}, \mathbf{b}).$$

Since $p < \frac{1}{k}$, implying $k \cdot p < 1$, we have that $f$ is a contraction. □

From Banach's fixed point theorem, we have that $f$ has a unique fixed point, and thus the system of equations (10) has a unique solution. Thus, to compute the values $\mu(q)$ for $q \in Q$ it is sufficient to compute this unique solution. This can be realized using Linear Programming [33], basically by the same approach used in [7].

**Definition 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:*

$$\begin{aligned}
\textit{objective}: &\quad \textit{min/max } z = c_1 x_1 + \ldots + c_n x_n \\
\textit{restrictions}: &\quad a_{1,1} x_1 + \ldots + a_{1,n} x_n \gtreqless b_1 \\
&\qquad \vdots \\
&\quad a_{m,1} x_1 + \ldots + a_{m,n} x_n \gtreqless b_m
\end{aligned}$$

*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 [33].

From the above system of equations 10 we can derive an LP problem. Consider for every $i \in I,\ (i_1, \ldots, i_k) \in I^k$ the inequation

$$x_i \leq wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \tag{11}$$

and the objective

$$z = \max \sum_{i \in I} x_i. \tag{12}$$

**Lemma 10.** *The LP problem consisting of the inequations* (11)*, and the objective* (12) *has the unique solution*

$$\{x_i \mapsto \mu(q_i) \mid i \in I\}.$$

*Proof.* Initially, observe that the above vector is in the feasible region, since it satisfies the restrictions (11). 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 $\mathbf{a}$ is a solution that maximizes the objective function then, for every $i \in I$, at least one of the inequalities (11) holds as an equality.

*Proof (Claim).* Suppose on the contrary that $\mathbf{c}$ is a solution that maximizes z, but for some $i \in I$, inequalities $c_i \leq wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot c_{i_j}$ are strict for all $(i_1, \ldots, i_k) \in I^k$. This would mean that the value of $c_i$ can be increased, and all inequalities would still hold; the increase in $c_i$ might increase the right-hand side of some other inequality, but since the left-hand side remains the same, all restrictions are satisfied. Thus, a new point $\mathbf{c}'$ has been produced that satisfies all the restrictions of the LP problem and additionally gives a larger value for the objective function. This is a contradiction to our initial assertion about $\mathbf{c}$. This completes the proof of the claim.

As a result, any points that are solutions to the LP problem, satisfy the condition $x_i = \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \right\}$ for all $i \in I$. Thus, they correspond to solutions of the system of equations (10).

Finally, since there is a unique such solution, the solution of the LP problem is this unique solution, the vector $(\mu(q_1), \ldots, \mu(q_n))$. □

Since solutions of Linear Programming problems can be computed in polynomial time, this is also the case for the values $\mu(q)$, and thus for the behavior.

**Theorem 5.** *The behavior of a $\Phi$-wLTA with contracting discounting $\Phi$ over $\mathbb{R}_{\inf}$ on the unlabeled tree can be computed in polynomial time.*

Note that for the Viterbi semiring we get analogous equations:

$$y_i = \max_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) \cdot \prod_{j=1}^{k} {y_{i_j}}^{p_j} \right\}$$

By taking the logarithm of these equations, we derive equations of the form (10). However, the numbers might no longer be rational, and thus further computational issues should be taken into consideration.

## 6 Conclusion

We have seen that concept comparison measures are important components of several approaches for approximation in DLs. Given two concepts $C, D$, such a measure assigns to them a value that expresses how well they compare. In general, these values come from a partially ordered set, but in most approaches to approximation considered so far, measures that map

into the real numbers, and often only into the real interval $[0, 1]$, are used. An important requirement for such measures is that they respect the semantics of concepts, i.e., are invariant under equivalence in the sense that, if we replace $C, D$ by equivalent concepts $C', D'$, then the returned similarity comparison value is the same. To be useful in practice, another important requirement on the measures is that they are computable.

The main technical contribution of this paper is the development of a general framework for defining concept comparison measures for the DL $\mathcal{FL}_0$ that are computable and invariant under equivalence w.r.t. general TBoxes. Our framework is based on a characterization of equivalence w.r.t. general $\mathcal{FL}_0$ TBoxes that uses tuples of formal languages. These tuples can be expressed by infinite trees, which in turn are represented by looping tree automata. Assigning a comparison value to a pair of $\mathcal{FL}_0$ concepts in an equivalence invariant way thus boils down to assigning a value to a tree that is represented by a looping tree automaton. We use weighted tree automata with discounting for this purpose, and reduce the problem of computing the comparison value to the problem of computing the behavior of such an automaton on the unlabeled infinite tree. If the weights of the automaton come from the semiring $\mathbb{R}_{\text{inf}}$, then this behavior can be computed in polynomial time provided that the employed discounting is nondecreasing or contracting. An obvious topic for future research is thus to extend these results to discounting that is neither contracting nor nondecreasing, or to other semirings as weight structures.

While the use of our framework guarantees that the obtained concept comparison measures are equivalence invariant and computable, the user of the framework needs to ensure (by appropriately defining the weighted automaton) that the obtained values make sense in the intended application. Nevertheless, it might be helpful to provide the user with automated tools for checking whether the defined measure satisfies certain properties, such as the properties often required for concept similarity measures [25]. In our framework, this boils down to deciding certain properties of weighted tree automata with discounting.

Finally, if concept comparison measures defined using our framework are employed within one of the approximation approaches sketched in Section 2.3, one can investigate whether the important inference problems in this approach are guaranteed to be decidable. For example, assume that a concept similarity measure defined using our approach is employed to relax instance queries in $\mathcal{FL}_0$. Can we extend our computability result for the measure to a decidability result for the relaxed instance problem? If the answer is affirmative, what is the exact complexity of the relaxed instance problem in this setting?

## References


1. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: Proc. of the 30th Int. Coll. on Automata, Languages and Programming (ICALP 2003). Lecture Notes in Computer Science, vol. 2719, pp. 1022–1037. Springer (2003)
2. Baader, F., Borgwardt, S., Morawska, B.: Extending unification in $\mathcal{EL}$ towards general TBoxes. In: Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012). pp. 568–572. AAAI Press/The MIT Press (2012)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005). pp. 364–369. Morgan Kaufmann, Los Altos (2005)
4. Baader, F., Brewka, G., Fernández Gil, O.: Adding threshold concepts to the description logic $\mathcal{EL}$. In: Proc. of the 10th Int. Symp. on Frontiers of Combining Systems (FroCoS 2015). Lecture Notes in Computer Science, vol. 9322, pp. 33–48. Springer (2015)
5. 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)
6. Baader, F., Fernández Gil, O.: Decidability and complexity of threshold description logics induced by concept similarity measures. In: Proc. of the 32nd Annual ACM Symp. on Applied Computing (SAC 2017), to appear. ACM (2017)

7. Baader, F., Marantidis, P., Okhotin, A.: Approximate unification in the description logic $\mathcal{FL}_0$. In: Proc. of the 15th Eur. Conf. on Logics in Artificial Intelligence (JELIA'2016). Lecture Notes in Computer Science, vol. 10021, pp. 49–63. Springer (2016)
8. Baader, F., Morawska, B.: Unification in the description logic $\mathcal{EL}$. Logical Methods in Computer Science 6(3) (2010)
9. Baader, F., Narendran, P.: Unification of concept terms in description logics. J. of Symbolic Computation 31(3), 277–305 (2001)
10. Baader, F., Okhotin, A.: On language equations with one-sided concatenation. Fundamenta Informaticae 126(1), 1–35 (2013)
11. Baader, F., Peñaloza, R.: Automata-based axiom pinpointing. J. of Automated Reasoning 45(2), 91–129 (2010)
12. 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 (2001)
13. Banach, S.: Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. Fund. Math. 3(1), 133–181 (1922)
14. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In: Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004). pp. 298–302. IOS Press (2004)
15. d'Amato, C., Fanizzi, N., Esposito, F.: A semantic similarity measure for expressive description logics. In: Pettorossi, A. (ed.) Proc. of Convegno Italiano di Logica Computazionale (CILC05) (2005)
16. Droste, M., Kuske, D.: Skew and infinitary formal power series. Theoretical Computer Science 366(3), 199–227 (2006)
17. Droste, M., Rahonis, G.: Weighted automata and weighted logics with discounting. Theoretical Computer Science 410(37), 3481–3494 (2009)
18. Ecke, A., Peñaloza, R., Turhan, A.Y.: Answering instance queries relaxed by concept similarity. In: Proc. of the 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2014). pp. 248–257. AAAI Press (2014)
19. Ecke, A., Peñaloza, R., Turhan, A.Y.: Similarity-based relaxed instance queries. J. of Applied Logic 13(4, Part 1), 480–508 (2015), special Issue for the Workshop on Weighted Logics for AI 2013
20. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, pp. 313–403. Springer Berlin Heidelberg (2009)
21. Hoehndorf, R., Schofield, P.N., Gkoutos, G.V.: The role of ontologies in biological and biomedical research: A functional perspective. Brief. Bioinform. 16(6), 1069–1080 (2015)
22. Knopp, K.: Theory and Application of Infinite Series (1951)
23. Kreyszig, E.: Introductory Functional Analysis With Applications. Wiley Classics Library, John Wiley & Sons (1978)
24. Küsters, R.: Non-standard Inferences in Description Logics, Lecture Notes in Artificial Intelligence, vol. 2100. Springer-Verlag (2001)
25. Lehmann, K., Turhan, A.Y.: A framework for semantic-based similarity measures for $\mathcal{ELH}$-concepts. In: Proc. of the 13th Eur. Conf. on Logics in Artificial Intelligence (JELIA'2012). Lecture Notes in Computer Science, vol. 7519, pp. 307–319. Springer (2012)
26. Mandrali, E., Rahonis, G.: Recognizable tree series with discounting. Acta Cybernetica 19(2), 411–439 (2009)
27. Pan, J.Z., Ren, Y., Zhao, Y.: Tractable approximate deduction for OWL. Artif. Intell. 235, 95–155 (2016)
28. Pensel, M.: An automata based approach for subsumption w.r.t. general concept inclusions in the description logic $\mathcal{FL}_0$. Master's thesis, Chair for Automata Theory, TU Dresden, Germany. See http://lat.inf.tu-dresden.de/research/mas. (2015)
29. Rabin, M.O.: Automata on Infinite Objects and Church's Problem. American Mathematical Society, Boston, MA, USA (1972)
30. Racharak, T., Suntisrivaraporn, B.: Similarity measures for $\mathcal{FL}_0$ concept descriptions from an automata-theoretic point of view. In: 6th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES). pp. 1–6 (2015)
31. Rahonis, G.: Weighted muller tree automata and weighted logics. J. Autom. Lang. Comb. 12(4), 455–483 (2007)

32. Schild, K.: A correspondence theory for terminological logics: Preliminary report. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91). pp. 466–471. Morgan Kaufmann, Los Altos (1991)
33. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons (1999)
34. Suntisrivaraporn, B.: A similarity measure for the description logic $\mathcal{EL}$ with unfoldable terminologies. In: 5th Int. Conf. on Intelligent Networking and Collaborative Systems. pp. 408–413. IEEE (2013)
35. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 133–192. The MIT Press (1990)
36. Wolter, F., Zakharyaschev, M.: Undecidability of the unification and admissibility problems for modal and description logics. ACM Trans. on Computational Logic 9(4) (2008)