TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden
Faculty of Computer Science
Institute for Automata Theory

Measuring description logic models using weighted tree automata

A thesis in fulfillment of the requirements for the degree of
Bachelor of Science

Jakob Krude

Matr-No.: 4818249
born 21.08.1999 in Berlin

Submitted on the 23.07.2021

First reviewer:
Dr.-Ing. Stefan Borgwardt

Second reviewer:
Prof. Heiko Vogler

Declaration of authorship

I hereby declare that I wrote the bachelor thesis I submitted today to the examination board of the Faculty of Computer Science on the topic:

Measuring description logic models
using weighted tree automata

completely on my own and that I did not use any sources and tools othert han those indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

23.07.2021

Abstract

Description Logics are used in a range of different fields. The main research areas study the applications, extensions and their complexity. This thesis is dedicated to evaluate the quality of models for Description Logics. For this purpose, it is shown how weighted alternating tree automata can be used to measure features of models. Afterwards, based on these automata, methods are proposed to enforce features of models to given thresholds. Finally, an alternating tree automaton is defined which accepts exactly those models that conform to the given feature restrictions.

Contents

1 Introduction

Structuring and formalizing knowledge is an essential step in understanding a domain and represent its complex relations. Description Logics (DLs) are becoming the de facto standard for such purposes. The key aspect of DLs is to define *concepts* and their relations. A concept can be described by defining rules that specify which properties must apply to elements of the concept. These rules can be atomic, so-called concept names or define relationships (*roles*) between elements. More complex concepts can be built by combining two or more using concept-constructions. To improve organisation further, it is possible to define hierarchies between concepts as sub- and superconcepts.

For example, one could model a small society using DLs. "Person" and "Job" would be atomic concepts. The concept of a worker ("Worker") is a person that works at a job, here "worksAt" is a relationship. A person can live ("livesIn") in a home ("Home"). Building on these concepts a member ("Member") of the society should have a job and is living in a home. Describing the relation between people we can add "childOf" and require that every child of a member is a member too.

Once formalized, it can be tested whether all rules are fulfilled for different societies. A given society would be a set of individuals belonging to the various concepts and relations. Such a society is called an interpretation. If all given criteria are fulfilled it is considered to be a model. For a description of a domain, there may be more than one interpretation satisfying it. For instance, there could be many different societies that fulfill the above conditions. As illustrated, description logics offer many possibilities to describe the individual concepts, their relations and hierarchies. However, many characteristics of models can be influenced only conditionally. For example, an interpretation could meet all the formal definitions but become unintentionally large. For instance, one could be interested in very small societies with less than 50 members. These requirements do not directly affect if a concept is fulfilled, but in which way it is fulfilled. In this respect, it can be interesting to compare different models regarding some features, e.g. by measuring their size or number of role connections. In this regard, models that are as small and simple as possible could play an important role in understanding the satisfiability of concepts. This is because interrelationships and dependencies become more obvious in a model that is as concise as possible. In particular, this is relevant in the context of larger and more complicated ontologies, where contradictions and unintended side effects may occur.

To solve all stated problems, defining relevant features, measuring and constraining models, this thesis establishes the necessary prerequisites, and afterwards, several methods are presented for measuring models. Specifically, this approach is based on the description logic $\mathcal{ALC}$, which makes it particularly versatile. Measuring models will be realized by using weighted alternating tree automata. In this sense, models are considered to be tree-shaped. The underlying idea in measuring models will therefore be to think of properties of models as properties of trees. Trees bring an advantage in terms of a higher degree of structure compared to general graphs. However, this may resolve in redundant elements, that are equal with respect to some concept. They can be thought of as duplicates of other elements and are only necessary to keep trees cycle-free. Therefore these redundancies should be excluded from any measurements and must be detected beforehand. Finally, several methods are introduced to apply the measured properties as restrictions to models. This thesis is based mainly on the results and automata from [1].

Determining similarities between concepts is a growing interest [7]. However, most approaches focus mainly on the composition of concepts rather than the structure of models. Furthermore, there are some extensions to $\mathcal{ALC}$ that allow additional requirements. For example number restrictions in $\mathcal{ALCN}$ allow to restrict the number of relationships between concepts. However, none of these approaches is as versatile in measuring and thus constraining models as described in this thesis. Feature-based comparisons of models have not yet been studied and bring a new aspect to the field of description logics.

2 Preliminaries

2.1 $\mathcal{ALC}$

Description Logic is a broad family of languages. Individual instances differ in expressiveness, complexity, decidability, or objective. $\mathcal{ALC}$ is a well-studied DL [3] and basis for many more. $\mathcal{ALC}$-concepts are built inductively from the already mentioned atomic concept and role names. The set N_C represents all concept names and N_R all role names. Every other concept is constructed using one of following methods: Let C, D be $\mathcal{ALC}$-concepts and r an $\mathcal{ALC}$-role name. Then:

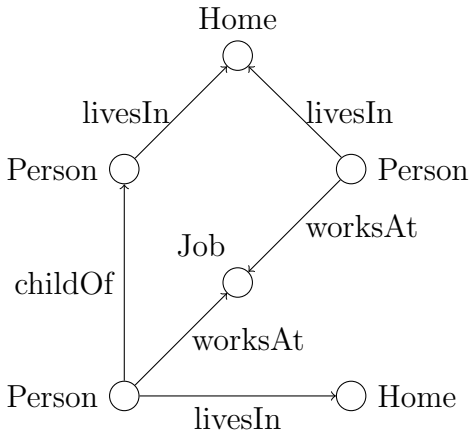
- $\neg C$ is a concept;
- $C \sqcap D$ is a concept;
- $C \sqcup D$ is a concept;
- $\exists r.C$ is a concept and
- $\forall r.C$ is a concept.

Additionally, $\top$ and $\bot$ are $\mathcal{ALC}$-concepts. As mentioned in the introduction an interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty set called interpretation domain, and $\cdot^I$ the interpretation function. The interpretation function assigns to every $A \in N_C$ a set $A^I \subseteq \Delta^I$ and to each role $r \in N_R$ a binary relation $r^I \subseteq \Delta^I \times \Delta^I$. Further, $\cdot^I$ is extended to concepts as follows:

Syntax	Semantics
$\top$	Δ^I
$\bot$	$\emptyset$
$\neg C$	$\Delta^I \setminus C^I$
$C \sqcap D$	$C^I \cap D^I$
$C \sqcup D$	$C^I \cup D^I$
$\exists r.C$	$\{d \in \Delta^I \mid \exists e \in \Delta^I, (d,e) \in r^I \wedge e \in C^I\}$
$\forall r.C$	$\{d \in \Delta^I \mid \forall e \in \Delta^I : (d,e) \in r^I \implies e \in C^I\}$

C^I is the *extension* of C. The notion of an extension of a concept C describes all elements in C^I. All hierarchies between $\mathcal{ALC}$-concepts are described as *general concept inclusions* (GCIs) of the form $C \sqsubseteq D$. A set of GCIs is called *TBox*. If every $C \sqsubseteq D$ in a given TBox $\mathcal{T}$ is satisfied by an interpretation I, that is $C^I \subseteq D^I$, I is called a *model* of $\mathcal{T}$. Accordingly, I is said to be a model of a concept C w.r.t. $\mathcal{T}$ if $C^I \neq \emptyset$ and I is a model of $\mathcal{T}$.

Revisiting the example from the introduction, a possible formalization for the society could be: Worker $\equiv$ Person $\sqcap \exists$worksAt.Job and Member $\equiv$ Worker $\sqcap \exists$livesIn.Home, where Person, Job, Home are concepts names and worksAt, livesIn are role names. As mentioned in the introduction, every child of a member is a member too: $\exists$childOf.Member $\sqsubseteq$ Member. It is common to represent an interpretation as directed graph, where every node represents an element from the interpretation domain and an edge represents a relationship. Concept names to which an element belongs are expressed as node labels. A possible interpretation for the given society description could look like:



Note that only concept names and role names are stated explicitly, compound concept extensions follow implicitly, for example the bottom left element is in the extension Member. This interpretation would be a model for $C = \forall$childOf.($\exists$livesIn.Home) and the already mentioned TBox, in other words there is a child for which all parents live in a house.

Alternatively, every GCI could be defined as a concept, e.g. $\neg(\exists$childOf.Member$) \sqcup$ Member. This possibility to interpret TBoxes as a concept is often used for the sake of convenience and simplicity. Specifically, a given TBox $\mathcal{T}$ can be expressed as concept $C_\mathcal{T} = \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D$. Then an interpretation I satisfies $\mathcal{T}$ iff for all elements d in Δ^I applies $d \in C_\mathcal{T}$.

In the following we assume that every $\mathcal{ALC}$-concept is in negation normal form. An $\mathcal{ALC}$-concept C is in negation normal form (nnf) if any negation occurs only directly in front of concept names. The transformation for any concept into negation normal form can be done in linear time, $NNF(C)$ will be used as notation for such a transformation.

$sub(C)$ denotes all subconcepts of C and is defined by induction over the

structure of C:

- If $C = A \in N_C \cup \{\top, \bot\}$, then $sub(C) = \{A\}$
- If $C = C_1 \sqcap C_2$ or $C = C_1 \sqcup C_2$, then $sub(C) = \{C\} \cup sub(C_1) \cup sub(C_2)$
- If $C = \neg D$ or $C = \exists r.D$ or $C = \forall r.D$, then $sub(C) = \{C\} \cup sub(D)$

$sig(C)$ denotes the set of all role and concept names used in C. Both definitions are taken from [3, Def. 3.10 and Def. 6.3]. As a note it should be mentioned that $\mathcal{ALC}$ allows to describe specific individuals by concept assertions and role assertions. A finite set of $\mathcal{ALC}$ concept and role assertions is called an $\mathcal{ALC}$-ABox. However, this thesis is focused only on $\mathcal{ALC}$-TBoxes.

2.2 Alternating Tree Automata

There are a number of different definitions used to describe *alternating tree automata*. This work is based on the definitions and results from [1], in particular on the automaton $\mathcal{A}_{C,\mathcal{T}}$. The main purpose of $\mathcal{A}_{C,\mathcal{T}}$ is to test the satisfiability of an $\mathcal{ALC}$-concept C with respect to an $\mathcal{ALC}$-TBox $\mathcal{T}$, where $\mathcal{T}$ is given as $\{\top \sqsubseteq C_{\mathcal{T}}\}$. To achieve this, conditions described in C and $C_{\mathcal{T}}$ are checked step by step on the input tree. For example, given a concept $\exists r.A$ the automaton requires one child node that is labeled with r and fullfills every condition of A. However, in order to measure various features, a model is needed in which successor nodes can be addressed directly, as possible in [4]. Although not directly expressible through the model used in this thesis, Section 4 presents a way to simulate similar structures.

We recall the definitions from [1, Def. 1]. First the notion of a tree and tree domain is defined. A *tree domain* is a prefix-closed, non-empty set $D \subseteq \mathbb{N}^*$, written as dom_T, where for every $u \in \mathbb{N}^*, i \in \mathbb{N}$ if $ui \in D$, then $u \in D$. Let Σ be an alphabet. A Σ-labeled *tree* is a pair (dom_T, T), where T is a function mapping from dom_T to Σ. Every element u in dom_T is called a node. If there exists no $i \in \mathbb{N}$ with $ui \in dom_T$, u is called a leaf. For every $i \in \mathbb{N}$, if $ui \in dom_T$ then ui is called a child of u. $Tree(\mathcal{P}(\Sigma))$ is the set of all $\mathcal{P}(\Sigma)$-trees.

Let $\pi(v) = u_0, u_1, \ldots, u_n$ be the *path* from the root to some node $u \in dom_T$, with $u_0 = \epsilon$ and $u_n = u$. Formally $\forall k \in \{0, \ldots n-1\} : \exists i \in \mathbb{N} : u_k i = u_{k+1}$. Due to the properties of a tree for every $u \in dom_T$ there is exactly one $\pi(u)$.

Given an alphabet Σ and a set of states Q, the transition function is defined as $\delta : Q \to TC(\Sigma, Q)$, where $TC(\Sigma, Q)$ is one of the following: `true`, `false`,

σ or $\neg\sigma$ for $\sigma \in \Sigma$; $q_1 \vee q_2$; $q_1 \wedge q_2$ for $q_1, q_2 \in Q$ or $\Diamond q, \Box q$ for $q \in Q$.

Definition 1 (alternating tree automaton)**.** *An alternating tree automaton (ata) $\mathcal{A}$ working on $\mathcal{P}(\Sigma)$-trees is a tuple $\mathcal{A} = (\Sigma,\ Q,\ q_0,\ \delta)$, where*

1. *Σ is a finite alphabet;*
2. *Q is a finite set of states;*
3. *$q_0 \in Q$ is the initial state and*
4. *δ is the transition function $Q \to TC(\Sigma, Q)$.*

The execution of an ata $\mathcal{A}$ over a given $\mathcal{P}(\Sigma)$-labeled tree is called a run R over $\mathcal{A}$. Because transitions like $q_1 \wedge q_2$ are allowed R itself is a $(dom_T \times Q)$-labeled tree, such that $\epsilon \in dom_R, R(\epsilon) = (\epsilon, q_0)$, and for all $u \in dom_R$ with $R(u) = (v, q)$ we have:

- $\delta(q) \neq \texttt{false}$;
- if $\delta(q) = \sigma$, then $\sigma \in T(v)$; and if $\delta(q) = \neg\sigma$, then $\sigma \notin T(v)$;
- if $\delta(q) = q_1 \wedge q_2$, then there exists $i_1, i_2 \in \mathbb{N}$ with $R(ui_1) = (v, q_1)$ and $R(ui_2) = (v, q_2)$;
- if $\delta(q) = q_1 \vee q_2$, then there exists $i_1, i_2 \in \mathbb{N}$ with $R(ui_1) = (v, q_1)$ or $R(ui_2) = (v, q_2)$;
- if $\delta(q) = \Diamond q'$, then there exists $i, j \in \mathbb{N}$ with $R(ui) = (vj, q')$ and
- if $\delta(q) = \Box q'$, then for every $i \in \mathbb{N}$ with $vi \in dom_T$ there exists $j \in \mathbb{N}$ with $R(ui) = (vj, q')$.

These descriptions follow the definitions from [1, Def. 1].

Formally, $\mathcal{A}_{C,\mathcal{T}}$ is specified as an alternating parity tree automaton. The parity property affects the acceptance for runs. Specifically, only those runs are accepted for which the largest priority, which occurs infinitely often along a branch, is even. However, for $\mathcal{A}_{C,\mathcal{T}}$ the priority function is defined as $\Omega(q) = 0$, i.e. everything is accepted. As described in the following sections, this thesis works on the initial part of tree models, which can only contain finitely long branches. Consequently, in this thesis the parity property is omitted and every run is accepted.

A tree T is *accepted* by an ata $\mathcal{A}$ if there exists a run on $\mathcal{A}$ over T. Consequently, the language $L(\mathcal{A})$ is the set of all accepted trees. Also, not explicitly mentioned complex transition conditions, like $\Diamond(A \vee \neg q) \wedge \Box\texttt{false}$, can be allowed without changing the expressiveness. Such an automaton that allows

complex transitions can be transformed into an equivalent automaton using only simple transitions by introducing new states for each subformula [8]. The emptiness problem for ata, that is the decision problem whether $L(\mathcal{A}) = \emptyset$, is in ExpTime [8].

A *subtree* T' of a tree T, is similar to a tree, only that an arbitrary $u \in \mathbb{N}^*$ can be the root. Therefore the tree domain is only prefix closed beyond u, i.e. for every $v \in \mathbb{N}^*, i \in \mathbb{N}$ if $vi \in D$ then $v \in D$ only applies for every $vi \neq u$. However every node $v \in dom_T \setminus \{u\}$ has to be a descendant of u, i.e. there exists $\gamma \in \mathbb{N}^* : v = u\gamma$. Further $dom_{T'} \subseteq dom_T$ and for all $v \in dom_{T'} : T'(v) = T(v)$. A *sub-run* R starting from $q \in Q$ at $u \in dom_T$ is a $(dom_T \times Q)$ labeled tree, where $R(\epsilon) = (u, q)$, from where on the same rules apply as for a normal run. In other words, a sub-run can start from any node in a tree at any state in the automaton.

2.3 Using Alternating tree automata on $\mathcal{ALC}$ models

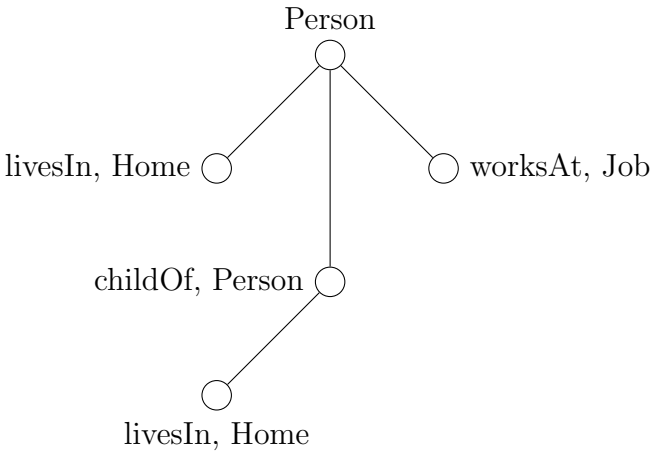
In the following, some symbols will be frequently reused and therefore not explicitly defined each time. If not stated otherwise C is an $\mathcal{ALC}$-concept, $\mathcal{T}$ is an $\mathcal{ALC}$-TBox in the form $\{\top \sqsubseteq C_\mathcal{T}\}$ and $\Sigma = sig(C) \cup sig(C_\mathcal{T})$. As introduced N_C is the set of all concept names and N_R the set of role names, respectively $N_R^* \coloneqq N_R \cap \Sigma$ and $N_C^* \coloneqq N_C \cap \Sigma$ denote those used in C and $C_\mathcal{T}$. Additionally, when u and ui are used to denote nodes of a tree T, ui refers to a child node of u, i.e. $i \in \mathbb{N}$. As already introduced, this work is based on $\mathcal{ALC}$ tree models. Tree models are formally introduced in [3, Def. 3.20]. There a tree is a graph $G = (V, E)$ such that

- V contains a unique root, i.e. there is no edge to this node in V and
- for every $e \in E$ there is at most one $f \in E$ with $(f, e) \in V$.

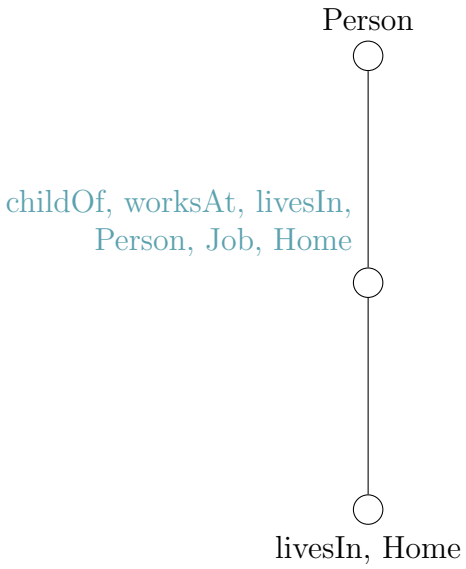
A *tree model* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ for some concept C and TBox $\mathcal{T}$ is a model and additionally: $(\Delta^I, \bigcup_{r \in N_R} r^\mathcal{I})$ is a tree. Limiting all models to be tree models is not a limitation, because it is well known that $\mathcal{ALC}$ has the tree model property [3, Theorem 3.24]. It states that every satisfiable concept C has a tree-shaped model where the root is an element of C. The definition of a tree domain does not allow labeled edges, therefore role names will be pushed into labels of child nodes.

Revisiting the example of a society, defining C as $\exists$worksAt.Job$\sqcap\exists$childOf.Person and $\mathcal{T}$ as $\{\text{Person} \sqsubseteq \exists\text{livesIn.Home}\}$, a tree-shaped model could look like:

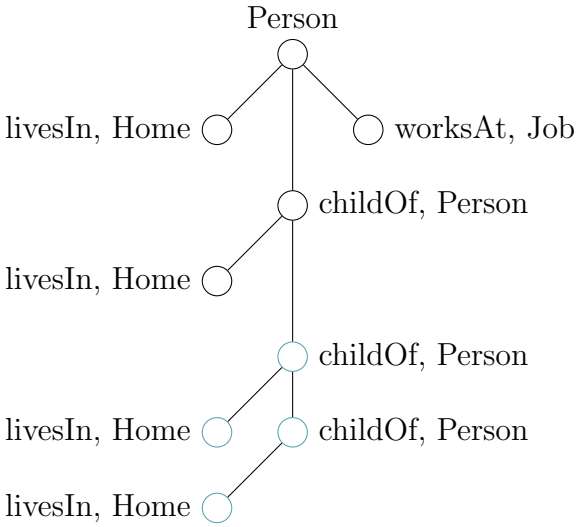
Example 1



However other models are imaginable that are not as intuitive:



Too many concept names for one node

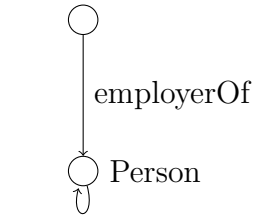


Unnecessarily deep model

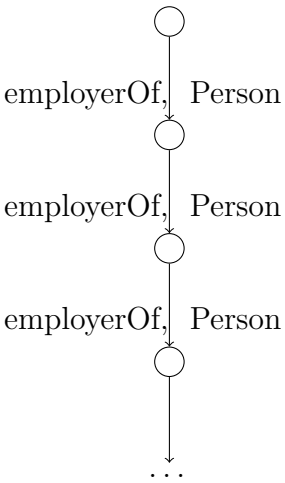
3 Detecting the initial part

In this section the main problem of redundant elements in a tree model is addressed. As mentioned in [1, p. 6], cycles in a given TBox $\mathcal{T}$ can enforce infinite tree models. This is a well-known consequence of unravelling an $\mathcal{ALC}$-model into a tree model. The resulting tree will have infinitely long branches that contain repetitions of nodes that are equal in the sense of concept satisfaction w.r.t. C and $C_{\mathcal{T}}$. For the majority of features, measuring the full tree model would be misleading and only the initial part is of concern. For example let's consider a society with strong work hierarchies $C = \exists$employerOf.Person and $\mathcal{T} = \{\text{Person} \sqsubseteq \exists\text{employerOf.Person}\}$.

Example 2



Model with cycle as a graph.



Tree-shaped model.

Measuring the number of people that are an employer of someone in the tree model, would result in infinitely many people instead of two. In order to avoid those problems, an extension $\mathcal{A}'_{C,\mathcal{T}}$ to $\mathcal{A}_{C,\mathcal{T}}$ defined in [1, Def. 3] is given. The main idea behind $\mathcal{A}'_{C,\mathcal{T}}$ will be to identify equivalence of nodes in a given tree based on which combination of concepts and sub-concepts within C and $\mathcal{T}$ are fulfilled. If a node fulfills some combination of concepts which was already fulfilled before, the automaton will halt and require the label §. To keep track of which concepts have already been fulfilled, a tuple of ones and zeros is carried along, where each bit represents a concept.

Definition 2. *Let C be an $\mathcal{ALC}$-concept, and a $\mathcal{T}$ be an $\mathcal{ALC}$-TBox given as $\{\top \sqsubseteq C_{\mathcal{T}}\}$ with both C and $C_{\mathcal{T}}$ in negation normal form.*
Let $S = sub(C) \cup sub(C_{\mathcal{T}})$.
Let $S_{CC} \coloneqq \{\ C_1 \sqcap \cdots \sqcap C_k \sqcap NNF(\neg D_1) \sqcap \cdots \sqcap NNF(\neg D_j)$
$\mid \{C_1, \ldots, C_k\} = U; \{D_1, \ldots, D_j\} = S \setminus U for\ U \in \mathcal{P}(S)\}$.

Let $cc_1, \ldots, cc_n$ be an arbitrary enumeration for $cc_i \in S_{CC}$.
The ata $\mathcal{A}'_{\mathcal{C},\mathcal{T}} = (\Sigma, Q, q_0, \delta)$is defined as follows:

- $\Sigma = sig(C) \cup sig(C_\mathcal{T}) \cup \{\S\}$
- $Q = Q_D \cup Q_R \cup \{q_0, q_{r\geq 1}\} \cup Q_\mathcal{T} \cup Q_{cc} \cup Q_{check}$
  - $Q_D = \{q_D, q_{D'} \mid D \in S, D' = NNF(\neg D)\}$
  - $Q_R = \{q_r, q_{\neg r} \mid r \in N_R \cap \Sigma\}$
  - $Q_{cc} = \{q_{cc_i} \mid cc_1, \ldots, cc_n \in S_{CC}\}$
  - $Q_\mathcal{T} = \{(q_\mathcal{T}, \omega) | \ \omega \in \{0,1\}^n\}$, *where* $n = |S_{CC}|$
  - $Q_{check} = \{(q_{check_i}, \omega) \mid \omega \in \{0,1\}^n, i \in \{1, \ldots, n\}\}$

  The transition function is defined as in $\mathcal{A}_{C,\mathcal{T}}$:

  $\delta(q_\sigma) = \sigma$ $\quad$ $\delta(q_{\neg\sigma}) = \neg\sigma$
  $\delta(q_{C_1 \sqcap C_2}) = q_{C_1} \wedge q_{C_2}$ $\quad$ $\delta(q_{C_1 \sqcup C_2}) = q_{C_1} \vee q_{C_2}$
  $\delta(q_{\exists r.C}) = \Diamond(q_r \wedge q_C)$ $\quad$ $\delta(q_{\forall r.C}) = \Box(\neg q_r \vee q_C)$
  $\delta(q_\top) = \texttt{true}$ $\quad$ $\delta(q_\bot) = \texttt{false}$

  And extended with:

  - $\delta(q_0) = q_C \wedge (q_\mathcal{T}, (0)^n)$
  - $\delta((q_\mathcal{T}, \omega)) = \bigvee_{1 \leq i \leq n} (q_{cc_i} \wedge (q_{check,i}, \omega))$
  - $\delta((q_{check_i}, \omega)) = \S$ *if* $\omega[i] = 1$
  - $\delta((q_{check,i}, \omega)) = \neg\S \wedge (q_{C_\mathcal{T}} \wedge \Box(q_{r\geq 1} \wedge (q_\mathcal{T}, \omega')))$ *if* $\omega[i] = 0$, *where* $\omega' = \omega[i \to 1]$
  - $\delta(q_{r\geq 1}) = \bigvee_{r \in N_R \cap \Sigma} q_r$

Note that cc_i is a conjunction of $\mathcal{ALC}$-concepts, where q_{cc_i} follows the transitions of $\mathcal{A}_{C,\mathcal{T}}$. As mentioned earlier, for all nodes must be tested if they satisfy $C_\mathcal{T}$. Previously this was implemented by the transition $\delta(q_\mathcal{T}) = q_{C_\mathcal{T}} \wedge \Box q_\mathcal{T}$ defined in $\mathcal{A}_{C,\mathcal{T}}$. In contrast, the automaton $\mathcal{A}'_{C,\mathcal{T}}$ first guesses the concept combination from S_{CC} that is satisfied by the current node. Subsequently $q_{check,i}$ tests if the concept combination was satisfied before. If that is true a cycle was detected and the automaton halts. Only if this is not the case, the further expansion $\delta(q_\mathcal{T}) = q_{C_\mathcal{T}} \wedge \Box(q_\mathcal{T}, \omega')$ takes place. Finally, $q_{r\geq 1}$ ensures that for every node beside the root at least one role name is part of the label. This will be important for the later definition of a representation.

Definition 3. *Given an $\mathcal{ALC}$-concept C, an $\mathcal{ALC}$-TBox $\mathcal{T}$ and a $\mathcal{P}(\Sigma)$-labeled tree T. Two nodes $u, v \in dom_T$ are considered* conceptual-equal *if there exists a $cc \in S_{CC}$ for which there are two sub-runs R_v, R_u that start at (v, q_{cc}) and (u, q_{cc}).*

Definition 4 (Accepted / Initial part)**.** *Let $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ be an ata working on $\mathcal{P}(\Sigma)$-trees and T be a $\mathcal{P}(\Sigma)$-tree. Let C be an $\mathcal{ALC}$-concept, and $\mathcal{T}$ an $\mathcal{ALC}$-TBox given as $\{\top \sqsubseteq C_{\mathcal{T}}\}$. Let $u \in dom_T$ be a node of T and $\pi(u)$ the path from the root to u. u is within the* accepted part *of T if one of the following is true:*

1. *There is no pair u_i, u_j in $\pi(u)$ where u_i and u_j are conceptual-equal.*
2. *There are u_i, u_j $(j > i)$ in $\pi(u)$ that are conceptual-equal for some $cc \in S_{CC}$ with sub-runs R_{u_i}, R_{u_j} and:*
   - *if $u \notin dom_T$ than R_{u_j} would not qualify as a run anymore,*
   - *no other conceptual-equal pair $u_k, u_l \in \pi(u)$ exists with $k, l < j$.*

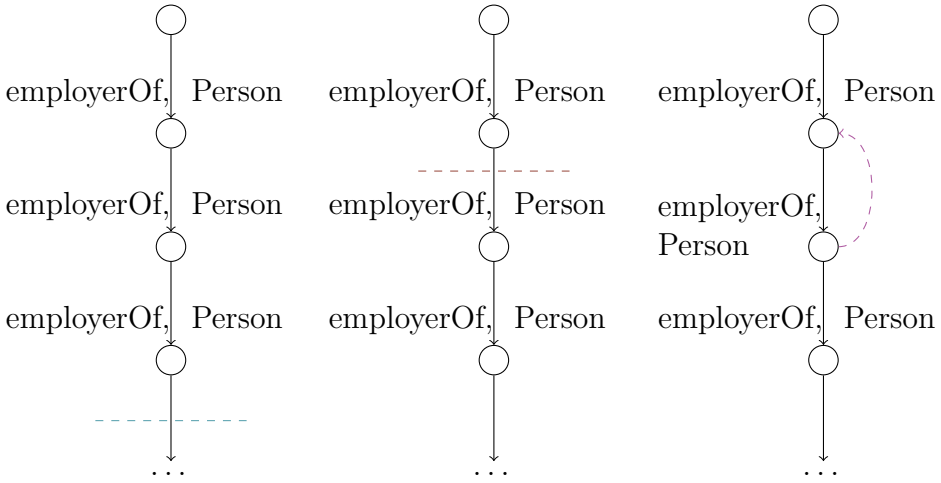
The accepted part of a tree is denoted as T_{acc}. Note that even though redundant some nodes are part of the accepted part in order to properly detect cycles, however they can be ignored in further sections and are not part of T_{init}. With T_{init} we denote the set

$$\{u \in T_{acc} \mid \text{there are no } u_i, u_j \text{ in } \pi(u) \text{ that are conceptual-equal}\}.$$

In order to use the recognition of conceptually equal nodes in later automata, $\mathcal{A}'_{C,\mathcal{T}}$ requires cycles to be marked by a §, i.e. for all conceptual equal nodes u_i, u_j in the accepted part $\S \in T(u_j)$ has to be true. Finally, with $Cyc(T)$ we denote the set of conceptual-equal nodes:

$$\{(v, u) \in T_{init} \times T_{acc} \mid v, u \text{ are conceptual-equal} \wedge \S \in T(u)\}.$$

In order to get a better understanding of the introduced terms, let's revisit the previous example.



The second and third node fulfill the same concept combination: $cc = \exists \text{employerOf.Person} \sqcap \text{Person}$. In order to test if this is true, the fourth node has to be part of the tree that is accepted by $\mathcal{A}'_{C,\mathcal{T}}$.

Definition 5 (Representation)**.** *Given an $\mathcal{ALC}$-concept C, an $\mathcal{ALC}$-TBox $\mathcal{T}$ and a $\mathcal{P}(\Sigma)$-labeled tree T.*
T represents *a tree model $\mathcal{I}$ of C w.r.t. $\mathcal{T}$ if there exists a set $\mathcal{I}_{init} \subseteq \Delta^{\mathcal{I}}$ for which there exists a bijection $\rho : T_{init} \to \mathcal{I}_{init}$ with the following properties:*

1. $\rho(\epsilon)$ *is the root of* $\mathcal{I}$
2. $\forall r \in N_R : \forall d \in \mathcal{I}_{init}, e \in \Delta^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}} \implies \exists u \in T_{init}, ui \in dom_T : (\rho(u) = d) \wedge (\rho(ui) = e) \wedge (\{s \mid (d,e) \in s^I\} = T(ui) \cap N_R)$.
3. $\forall u \in T_{init} : \forall ui \in dom_T : \forall r \in N_R : r \in T(ui) \iff (\rho(u), \rho(ui)) \in r^{\mathcal{I}}$
4. $\forall A \in N_C : \forall u \in T_{init} : A \in T(u) \iff \rho(u) \in A^{\mathcal{I}}$

This definition expresses the close relationship between a $\mathcal{P}(\Sigma)$-labeled tree and a tree model for C w.r.t. $\mathcal{T}$. Specifically, there are two sets T_{init} and $\mathcal{I}_{init}$ which are structurally equal. This means that every node can be uniquely assigned to an element in the interpretation domain, such that no edge may occur in the model that does not occur in the tree and vice versa. Note that this definition implies that in each label of a child node there must be at least one role name. This is due to the fact that each edge in the tree model represents a role relationship. In addition, it is required that a concept name as label equals concept membership in the model.

Theorem 1. *Given an $\mathcal{ALC}$-concept C, an $\mathcal{ALC}$-TBox $\mathcal{T}$ in the form $\{\top \sqsubseteq C_{\mathcal{T}}\}$ and a $\mathcal{P}(\Sigma)$-labeled tree T.*

1. *$T \in L(\mathcal{A}'_{C,\mathcal{T}}) \implies T$ represents a tree model of C w.r.t. $\mathcal{T}$,*
2. *$\mathcal{I}$ is a tree model of C w.r.t. $\mathcal{T} \implies$ there exists a tree T that represents $\mathcal{I}$ and $T \in L(\mathcal{A}'_{C,\mathcal{T}})$.*

Proof.

"1." Let T be a $\mathcal{P}(\Sigma)$-labeled tree that is in $L(\mathcal{A}'_{C,\mathcal{T}})$, i.e. there is a run R over $\mathcal{A}'_{C,\mathcal{T}}$ of T. First an interpretation I is constructed for which it is proven that I is a model of C w.r.t. $\mathcal{T}$. Secondly we unravel the interpretation I into a tree model $\mathcal{I}$ and finally it is shown that this tree model is represented by T. The Interpretation I is constructed as

$$\Delta^I = T_{init},$$
$$A^I = \{u \in T_{init} \mid A \in T(u)\},$$
$$r^I = \{(u, ui) \in T_{init} \times T_{init} \mid r \in T(ui)\} \cup \{(u, v) \mid (v, ui) \in Cyc(T), r \in T(ui)\}.$$

Claim 86 in [5, p. 93 f.] already showed for $\mathcal{A}_{C,\mathcal{T}}$ that for any $u \in \Delta^I$ and $E \in sub(C) \cup sub(C_{\mathcal{T}})$, if (u, q_E) occurs as a label of a node in R, then $u \in E^I$. Note that Δ^I is no longer defined as dom_T but T_{init}. Therefore, for $E = \exists r.F$ there can be $w \in dom_R$ with $R(w) = (u, q_E)$ where $R(wj) = (ui, q_F)$ but $\S \in T(ui)$ and hence $ui \notin \Delta^I$. This means the pair $(u, ui) \notin r^I$ and thus $\exists r.F$ for $u \in \Delta^I$ is no longer fulfilled. With induction over the size of E, as done in [5], it can be shown that the claim still holds. If $E = \exists r.F$ and $\S \notin T(ui)$, then the same argumentation as in claim 86 applies. If $\S \in T(ui)$ we know that there is a $v \in T_{init}$ with $(v, ui) \in Cyc(T)$, where v and ui are conceptual-equal for some $cc \in S_{CC}$. Recalling the structure of cc as $C_1 \sqcap \cdots \sqcap C_n \sqcap \neg D_1 \sqcap \cdots \sqcap \neg D_m$, either $F = C_i$ or $F = D_j$ for some i or j. As stated in the assumption, (ui, q_F) is a label of a node in R and thus $F = C_i$ is true for some $i \in \{1, \ldots, n\}$ since $F \in sub(C) \cup sub(C_{\mathcal{T}})$. Furthermore, $\S \notin T(v)$ and as a result $v \in F^I$. By definition of r^I, $(u, v) \in r^I$ applies for all $r \in T(ui)$ from which it follows that $u \in E^I$. As in claim 86 $\epsilon \in C^I$ follows from $\delta(q_0) = q_C \wedge (q_{\mathcal{T}}, \omega)$. Considering that there is always exactly one part of the disjunction from $\delta(q_{\mathcal{T}}, \omega) = \bigvee_{1 \leq i \leq n}(q_{\mathrm{cc}_i} \wedge (q_{\mathrm{check},i}, \omega))$ that is fulfilled, $q_{\mathcal{T}}$ is tested for every $u \in T_{init}$ thus $\forall u \in \Delta^I : u \in C^I_{\mathcal{T}}$. This shows that I is indeed a model of C w.r.t. $\mathcal{T}$.

As mentioned, I will be unravelled into $\mathcal{I}$ as described in [3, Def. 3.21]. First ϵ is chosen as root for $\mathcal{I}$, as we already know that $\epsilon \in C^I$. Accordingly, $\Delta^{\mathcal{I}}$ will be constructed from d-paths, which are sequences of nodes $\epsilon, \ldots, u_{n-1}$, such that (u_i, u_{i+1}) is in some r^I. $A^{\mathcal{I}}$ and $r^{\mathcal{I}}$ for all $r \in N_r$ and $A \in N_C$ will be constructed as in [3, Def. 3.21]. As last step it has to be shown that T indeed represents $\mathcal{I}$. This can be simply shown by specifying $\rho(u) \mapsto \pi(u)$, for all $u \in T_{init}$, i.e. each element is mapped to its path from the root. Let $\mathcal{I}_{init} = \{\rho(u) \mid u \in T_{init}\}$. Every path $\pi(u)$ is unique in T and therefore all requirements for $\rho : T_{init} \to \mathcal{I}_{init}$ immediately follow from the definition of I and hence $\mathcal{I}$.

"2." Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a tree model of C w.r.t. $C_{\mathcal{T}}$. Let $d_0 \in \Delta^{\mathcal{I}}$ be the root of $\mathcal{I}$. As already described in Section 4, C and $C_{\mathcal{T}}$ can contain only finitely many existence restrictions ($\exists$). Therefore it can be assumed that for every $d \in \Delta^{\mathcal{I}}$ there are only a finite number of elements $e \in \Delta^{\mathcal{I}}$ with $(d, e) \in r^{\mathcal{I}}$, for any $r \in N_R$. Let $d \in \Delta^{\mathcal{I}}$ be an arbitrary element and $d_1, \ldots, d_n$ an enumeration of all elements such that (d, d_i) is in some $r^{\mathcal{I}}$. We can construct a tree T inductively from $\mathcal{I}$ using a mapping $\rho^{-1} : \Delta^{\mathcal{I}} \to dom_T$.

$$\rho^{-1}(d_0) = \epsilon$$

$$\rho^{-1}(d_i) = \rho^{-1}(d)i$$

$$T(\rho^{-1}(d_0)) = \{A \mid d_0 \in A^{\mathcal{I}}\}$$

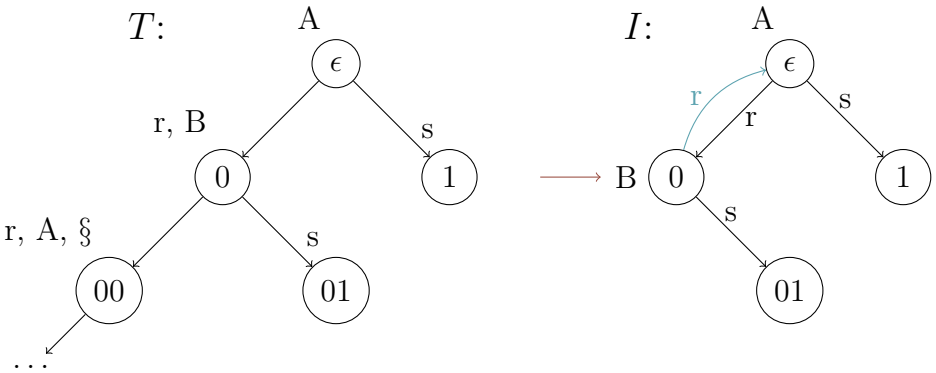
$$T(\rho^{-1}(d_i)) = \{r \mid (d, d_i) \in r^{\mathcal{I}}\} \cup \{A \mid d_i \in A^{\mathcal{I}}\}$$

Using Definition 3 we can define the set of all conceptual-equal nodes in T as $Cyc = \{(v, u) \mid v, u \text{ are conceptual-equal}, \exists \gamma \in \mathbb{N}^* : u = v\gamma\}$. In the same way T_{init} is well-defined over T. Let $\mathcal{I}_{init} = \{d \in \Delta^{\mathcal{I}} \mid \rho^{-1}(d) \in T_{init}\}$. As apparent, T represents $\mathcal{I}$, where ρ is the inverse of ρ^{-1} restricted to $T_{init} \to \mathcal{I}_{init}$. For T to be accepted by $\mathcal{A}'_{C,\mathcal{T}}$, all cycles must be marked with §. Therefore, $\forall (v, u) \in Cyc$ add § as label to $T(u)$. Finally, for this tree a run R over $\mathcal{A}'_{C,\mathcal{T}}$ can be constructed starting from $R(\epsilon) = (\epsilon, q_0)$ and just following the transition function δ defined for $\mathcal{A}'_{C,\mathcal{T}}$. To be complete, every node in T_{init} beside the root will have at least one role name as a label, which follows by the definition of ρ. This concludes the proof.

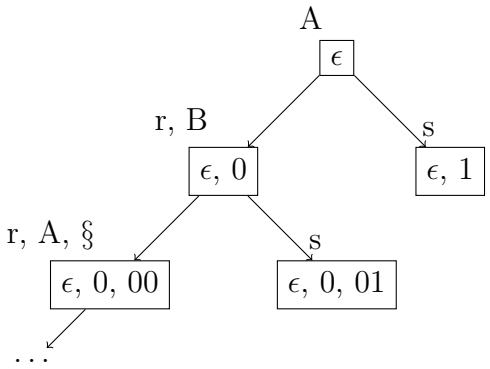
□

The first part ("1.") of the proof can best be illustrated by an example. Let $C = \exists r.B \sqcup \exists s.\top$ and $\mathcal{T} = \{A \sqsubseteq \exists r.B, B \sqsubseteq \exists r.A, B \sqsubseteq \exists s.\top\}$

Example 3



From T the model I is built: $(0, \epsilon) \in r^I$ closes the detected cycle.



The model I was unraveled into the tree model $\mathcal{I}$, by substituting every element with a d-path.

The size of S is polynomial, therefore the size of S_{CC} is exponential. It follows that $\omega \in \{0,1\}^{|S_{CC}|}$ has exponentially many digits, and thus the set of all ω is double-exponential in the size of C and $C_{\mathcal{T}}$. As stated previously the emptiness-test for ata is in ExpTime. This concluded that the decision problem $L(\mathcal{A}'_{C,\mathcal{T}}) \neq \emptyset$ is in 3ExpTime. As a result, the proposed extension for $\mathcal{A}_{C,\mathcal{T}}$ that additionally detects cycles has a double exponential blowup.

4 Indexed trees

As a last step before measuring tree models one more extension is introduced: addressing children of a node directly. This is necessary whenever a feature affects more than one branch, for example measuring the maximum degree of branching. The model for alternating tree automata in [4] would be a better fit for this purpose, but would require a full translation of the previous work. This is not necessary because we can simulate a similar behaviour with the already introduced transition options. Desired is a transition equivalent to "go the i-th child and test q'".

However, since the previous alphabet over all sub-concepts is not ranked,there needs to be a limit for the number of children of a node. As shown in [2], it is sufficient to limit the maximum number of children to the number of existential restrictions in C and $C_{\mathcal{T}}$. In the following paragraph $\mathbb{N}_k$ shall be defined as $\mathbb{N}_k \coloneqq \{1, \ldots, k\}$. Let $\Sigma_k = \Sigma \cup \{\#\} \cup \mathbb{N}_k$, where k is the number of existential restrictions in C and $C_{\mathcal{T}}$. Clearly, not all tree models have exactly k many successors for all nodes. The nodes that only fill up the number of children are therefore called dummy nodes in the following and are marked by the symbol #.

Definition 6. *(Indexed tree) A $\mathcal{P}(\Sigma_k)$-labeled tree is called* indexed *if for every $u \in dom_T$ the following characteristics apply:*

- $|T(u) \cap \mathbb{N}_k| = 1$*;*
- *if $\# \notin T(u)$ then $\forall i \in \mathbb{N}_k : \exists j \in \mathbb{N} : i \in T(uj)$;*

  let v, w be children of u, with $i \in T(v)$ and $j \in T(w)$;

  - *if $j \geq i$ and $\# \in T(v)$ then $\# \in T(w)$ and*
  - *if $j = i$, then v and w are conceptual-equal.*

The automaton $\mathcal{A}_{idx}$ will enforce each of those requirements. Note that the last requirement is building on the previously introduced notion of conceptual-equality. Therefore, $\mathcal{A}_{idx}$ needs the ability to check for concept-satisfaction. In order to achieve this, the required states and transitions are reused from $\mathcal{A}'_{C,\mathcal{T}}$.
$\mathcal{A}_{idx} = (\Sigma_k, Q, q_{rec}, \delta)$, where $Q = Q_D \cup Q_R \cup Q_{cc} \cup \{q_i, (q_d, i) | i \in \mathbb{N}_k\} \cup \{q_{rec}, q, q'\}$ and δ is defined as:

- $\delta(q_{rec}) = \# \vee (\neg\# \wedge q \wedge q' \wedge \Box q_{rec})$
- $\delta(q) = \bigwedge\limits_{i \in \mathbb{N}_k} (\Diamond q_i \wedge (\bigvee_{cc \in S_{CC}} \Box(\neg i \vee q_{cc})))$

- $\delta(q') = \bigwedge_{i \in \mathbb{N}_k} (\Diamond(\neg\# \wedge i) \vee \Box q_{d,i})$
- $\delta(q_{d,i}) = (\bigvee_{1 \leq j < i} j) \vee \#$
- $\delta(q_i) = i \wedge \bigwedge_{j \in \mathbb{N}_k, j \neq i} \neg j$

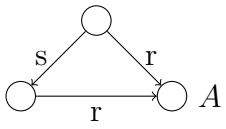
This approach has the disadvantage of dummy nodes, which would have to be treated separately when defining transitions. One could change all transitions of $\mathcal{A}'_{C,\mathcal{T}}$ and the following automata to ignore dummy nodes, for example $\delta(q) = \Diamond q'$ becomes $\delta(q) = \Diamond(\neg\# \wedge q')$ and $\delta(q) = \Box q'$ would be changed to $\delta(q) = \Box(\neg\# \vee q')$. However, in the interest of readability and comprehensibility we will assume that every tree is indexed and dummy nodes are ignored,i.e. $\Diamond$ and $\Box$ work as described. Finally, we can simulate the desired transition "test q' for the i-th child" as $\delta(q) = \Diamond(i \wedge \neg\# \wedge q')$ and write $\delta(q) = \Diamond_i q'$ as abbreviation. In the same sense $\delta(q) = \Box_i q'$ is interpreted as "for all children with index i test q'", formally $\delta(q) = \Box(\neg i \vee \neg\# \vee q')$.

5 Measuring description logic models

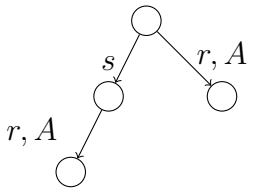
Measured features described in this section are divided into three categories:

1. features comparing branches of trees;
2. features comparing individual nodes of trees and
3. features that cover the entire tree.

Formally features of trees are measured, however due to the definition of "represents", this corresponds to a feature of a tree model (and specifically their initial part). Every feature will be described as a function, where $f(T_{init}) = n, n \in \mathbb{N}$ is the measured feature of T. For every type of feature, an example is given and an automaton that computes $f(T_{init})$. Every example includes one general graph-based feature and a concrete idea for tree model over $\mathcal{ALC}$. Based on these measurements, requirements can be imposed on models. Utilizing measurements as restrictions on tree models can be achieved in two ways. Either by an ata, which tests some threshold directly during measurement, or by combining a measuring automata with the cut-point approach described in [1, Def. 7]. In the following section, every tree T is assumed to represent a tree model, where cycles are marked by §. Keep in mind that all measurements take place on tree model, and the outcome may differ from a measurement of a model defined as a graph. For example:



Model as graph.
Extension size of $A = 1$.



Model as tree.
Extension size of $A = 2$.

Desired is an automaton that marks certain nodes with weights, such that the sum of all weights of a run over this automaton is the measurement for the given feature. A weighted alternating tree automaton is nearly identical to the already defined alternating tree automaton, except for weighted transitions. In addition to all options for transitions defined in $TC(\Sigma, Q)$ the transition function may also contain a non-negative integer.

Definition 7 (Weighted alternating tree automata)**.** *A weighted alternating tree automaton (wata) $\mathcal{A}$ working on $\mathcal{P}(\Sigma)$-trees is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$, where 1. Σ is a finite alphabet; 2. Q is a finite set of states, $q_0 \in Q$ is the*

initial state; and 3. $\delta : Q \to wTC(\Sigma, Q)$ *is the transition function, where* $q \mapsto wTC(\Sigma, Q)$ *is either* $n \in \mathbb{N}$ *or any option from* $TC(\Sigma, Q)$.

This does not change anything for the definition of runs. Nodes in a run labeled with $n \in \mathbb{N}$ do not need to satisfy any condition and can be leaves. As before the definition closely follows [1, Def. 6], with the omission of the parity function. In order to achieve the desired functionality, the $\Box$-fixation has to be defined. Just as in [1, Def. 6], for any run R, the $\Box$-fixation is a tree R' with $dom'_R \subseteq dom_R$ and $R'(u) = R(u)$ for all $u \in dom'_R$, only that for $v \in dom_R$ if $R(v) = (u, q)$, such that $\delta(q) = \Box q'$, then there is at most one $j \in \mathbb{N}$ with $vj \in dom_R$. Now the behaviour of the weighted automaton can be expressed as a function $\|\mathcal{A}\|$: Tree($\mathcal{P}(\Sigma)$)$\to \mathbb{N}$. The weight of a $\Box$-fixation R' of a run R is defined as

$$\text{weight}_{\mathcal{A}}(R') = \sum_{u \in dom_{R'}, R'(u)=(d,q), \delta(q,T(u))=n \in \mathbb{N}} n.$$

If infinitely many values $n > 0$ occur in R', the weight of R' is ∞; otherwise it is the finite sum of all weights in R'. The weight of a run R on T is $\text{weight}_{\mathcal{A}}(R) = \sup_{R' \ \Box-\text{fixation of R}} \text{weight}_{\mathcal{A}}(R')$, and the behaviour of $\mathcal{A}$ is $\|\mathcal{A}\|(T) = \min_{R \ run \ on \ T} \text{weight}_{\mathcal{A}}(R)$.
In other words, $\text{weight}_{\mathcal{A}}(R')$ are all weights in R' combined, $\text{weight}_{\mathcal{A}}(R)$ is the highest weight of all $\Box$-fixations and finally $\|\mathcal{A}\|(T)$ is the minimum over all runs.

5.1 Measuring tree model with weighted alternating tree automata

We want to measure all features only in relation to the initial part of an $\mathcal{ALC}$ model, therefore no child of a node with label § is included. For every proposed feature and weighted alternating tree automaton $\mathcal{A}$ that computes $f(T_{init})$ will apply $\|\mathcal{A}\|(T) = f(T_{init})$.

Features comparing branches

In the first approach presented here, features are considered in which branches are compared. In general graph terms, an automaton is given which measures the depth of a tree, and specifically in the context of $\mathcal{ALC}$, an automaton which measures the maximum number of different role names per branch. The depth of a tree describes the longest path from the root to any leaf (or a parent of a node labeled with §). Formally $depth(T) = \max\{|\pi(u)| \mid u \in T_{init}\}$, where $|\pi(u)| = |\epsilon, u_1, \ldots, u_n| = n$.

$\mathcal{A}_{depth} = (\Sigma, \{q, q_{next}\}, q_{next}, \delta)$, where δ is defined as

- $\delta(q_{next}) = \S \vee \Box q$;
- $\delta(q) = 1 \wedge q_{next}$.

Either no child exists or 1 is added as weight. However, according to the $\Box$-fixation only one child is in any $\Box$-fixation for $\Box q$. The supremum of all R' is the one, where every decision resulted in the maximum overall weight, that is precisely the longest path from the root to any leaf (or a parent of a node labeled with §). $\|A\|(T)$ is the minimum of all weights of runs, therefore if $\Box q$ is expanded, although the node was labeled with §, there is a run that stops and the weight is smaller or equal. This concludes that $\|\mathcal{A}_{depth}\|(T)$ is indeed the depth of T. As is apparent, the previous results, cycle-recognition and indexing trees allow for very simple automata.

In the context of tree models, the greatest number of unique role names per branch could be a practical measurement. A set of states is required that keeps track of all visited role names. Let N_R^* be $\Sigma \cap N_R$. Let n be the size of N_R^*. In this case $rNames(T) = \max\{r(\ \pi(u)\ ) \mid u \in T_{init}\}$, where $r(\pi(u)) = |\{s \in N_R | s \in T(v); v \in \pi(u)\}|$. Let $Q = \{(q, \omega), (q_{r,n+1}, q_{next}, \omega) \mid \omega \in \mathbb{N}^n\}$, $Q_R = \{q_r, \omega \mid r \in N_R^*, \omega \in \mathbb{N}^n\}$. Let idx be a function from $N_R^* \to \{1, \ldots, n\}$ $\mathcal{A}_{\#roles} = (\Sigma, Q \cup Q_R \cup \{q_0\}, q_0, \delta)$, where δ is defined as

- $\delta(q_0) = (q_{next}, (0)^n)$;
- $\delta(q_{next}, \omega) = \S \vee \Box(q, \omega)$;
- $\delta(q, \omega) = (q_{r,1}, \omega)$.
- $\delta(q_{r,i}, \omega) = (\neg r \wedge (q_{r,i+1}, \omega)) \vee (r \wedge (q_i, \omega'))$
- $\delta(q_i, \omega) = 1 \wedge (q_{r,i+1}, \omega')$ if $\omega[i] = 0$, where $\omega' = \omega[i \to 1]$
- $\delta(q_i, \omega) = (q_{r,i+1}, \omega)$ otherwise
- $\delta(q_{r,n+1}, \omega) = (q_{next}, \omega)$

The definition for $\mathcal{A}_{\#roles}$ is extending $\mathcal{A}_{depth}$ in is such a way, that instead of emitting 1 for every visited node u, it is tested step by step for all $r \in N_R^*$ whether $r \in T(u)$. In fact, the difference could be interpreted as an added if-clause: Only add one as weight if $r \in T(u)$ was not seen before. Therefore, the same argumentation can be applied and $\|\mathcal{A}_{\#roles}\|(T)$ is the greatest number of unique role names per branch.

As an example, consider a society for which the different types of relationships are modeled ("childOf", "sibling" etc.). One branch of a tree model of such

a society is a sequence of individuals, so that all of them are related. The depth of such a tree model, is the greatest degree of separation between the root-individual and any other individual in the model. The number of different role names in such a branch is the diversity of relations.

Features comparing individual nodes

Another interesting property of trees is the degree of branching, it describes the greatest number of children per node over the complete tree. Measuring this feature requires counting the number of children, and therefore uses the established notion of indexed trees. Here k is the number of existential restrictions in $sub(C) \cup sub(C_\mathcal{T})$, as described in Section 4. Note that if a node has only i children ($i < k$) then only for every $j \in \{1, \ldots, i\}$ there is a child with an index j. The formal definition of *degree* is $degree(T) = \max\{d(u) \mid u \in T_{init}\}$, where $d(u) = |\{j \in \{1, \ldots, k\} \mid \exists i \in \mathbb{N} : ui \in dom_T \wedge j \in T(ui)\})|$. Let $Q = \{(q_{measure}, j),\ (q, j) \mid j \in \{0, \ldots, k\}\} \cup \{(q_{next,i}, j) \mid i \in \{1, \ldots, k\}, j \in \{0, \ldots, k\}\}$.
$\mathcal{A}_{degree} = (\Sigma, Q, (q, 0), \delta)$, where δ is defined as

- $\delta(q, j) = \S \vee (\neg\S \wedge (q_{measure}, j))$
- $\delta(q_{measure}, j) = \bigvee_{1 \leq i \leq k} (\Diamond_i \texttt{true} \wedge \Box_{i+1} \texttt{false} \wedge (q_{next,i}, j)) \vee \Box \texttt{false}$
- $\delta((q_{next,i}, j) = i - j \wedge \Box(q, i)$ if $i > j$
- $\delta((q_{next,i}, j) = \Box(q, j)$ otherwise

Either no child exists ($\Box\texttt{false}$) or there is one child with the highest index i. If the number of children i is higher then the previous maximum j the difference is added as weight. Again, per $\Box$-fixation only one child is part of any $\Box$-fixation for $\Box(q, j)$. That way, the sum of all weights in a branch will be the highest number of children measured, and the supremum of all $\Box$-fixations is the branch with the greatest sum of weights. It should be noted that runs can differ only by which child with index i is selected for $\Diamond_i\texttt{true}$ and therefore all runs have the same weight.

Comparing individual nodes in the context of $\mathcal{ALC}$ can be used to measure the maximum number of concept names per node. To be precise let $cNames$ be the feature function with $cNames(T) = \max\{c(u) \mid u \in T_{init}\}$, where $c(u) = |\{A \in N_C | A \in T(u)\}|$. Let $N_C^* = \Sigma \cap N_C$, the set of all concept names in C and $C_\mathcal{T}$. Let $CC_{names} = \{(cc, n) \mid cc = A_1 \wedge \ldots A_n \wedge \neg B_1 \wedge \ldots \neg B_m;\ \{A_1, \ldots, A_n\} = U, \{B_1, \ldots, B_m\} = N_C^* \setminus U;\ \forall U \in \mathcal{P}(N_C^*)\}$.
$\mathcal{A}_{\#names} = (\Sigma, Q, (q, 0), \delta)$, where δ is defined as

- $\delta(q, j) = \S \vee (\neg\S \wedge (q_{measure}, j))$
- $\delta(q_{measure}, j) = \bigvee_{(cc,i) \in CC_{names}} (cc \wedge (q_{next,i}, j))$
- $\delta(q_{next,i}, j) = i - j \wedge \Box(q, i)$ if $i > j$
- $\delta(q_{next,i}, j) = \Box(q, j)$ otherwise

Instead of measuring the number of children, $\mathcal{A}_{\#names}$ will measure the number of concept names. To accomplish this, the combination of concept names that are part of the label has to be guessed. Again, there exists only one combination that can be fulfilled because every combination contains all concept names and differs by at least one negation.

Features affecting the entire tree

Not all properties compare parts of the tree, but apply to its entirety. For example, the size of the initial part, i.e. the number of all nodes preceding a cycle. Let $size(T) = |T_{init}|$ be the function that describes the size of the initial part. Let k be the number of existential restrictions in $sub(C) \cup sub(C_{\mathcal{T}})$.
$\mathcal{A}_{size} = (\Sigma, \{q_{next}, q\}, q_{next}, \delta)$, where δ is defined as

- $\delta(q_{next}) = \S \vee (\neg\S \wedge 1 \wedge q)$
- $\delta(q) = \bigwedge_{1 \leq i \leq k} \Box_i q$

To ensure that all nodes and thus all weights are included in a run, $\Box q$ must be simulated by addressing every child individually. As noted in Section 4, it is not possible to prevent multiple child nodes from having the same index. However, it can be guaranteed that all nodes with the same index behave the same with respect to a concept combination. In the case that two child nodes have the same index, the node with the largest subtree will be taken into account in the total weight.

In regard of tree models over some $\mathcal{ALC}$-concept C and $\mathcal{ALC}$-TBox $\mathcal{T}$, this method can be applied to measure the extension of a given concept $E \in sub(C) \cup sub(C_{\mathcal{T}})$. The negation $\neg E$ transformed to negation normal form will be denoted as $E' = NNF(\neg E)$. Let $size_E(T)$ be the feature function with $size_E(T) = |\{u \in T_{init} \mid \exists \text{ sub-run over } \mathcal{A}_{size,E} \text{ starting from } (u, q_E)\}$. Let Q_D, Q_R be the set of states introduced in $\mathcal{A}'_{C,\mathcal{T}}$.
$\mathcal{A}_{size,D} = (\Sigma, \{q_{next}, q, q_{measure}\} \cup Q_E \cup Q_R, q_{next}, \delta)$, where δ is defined as

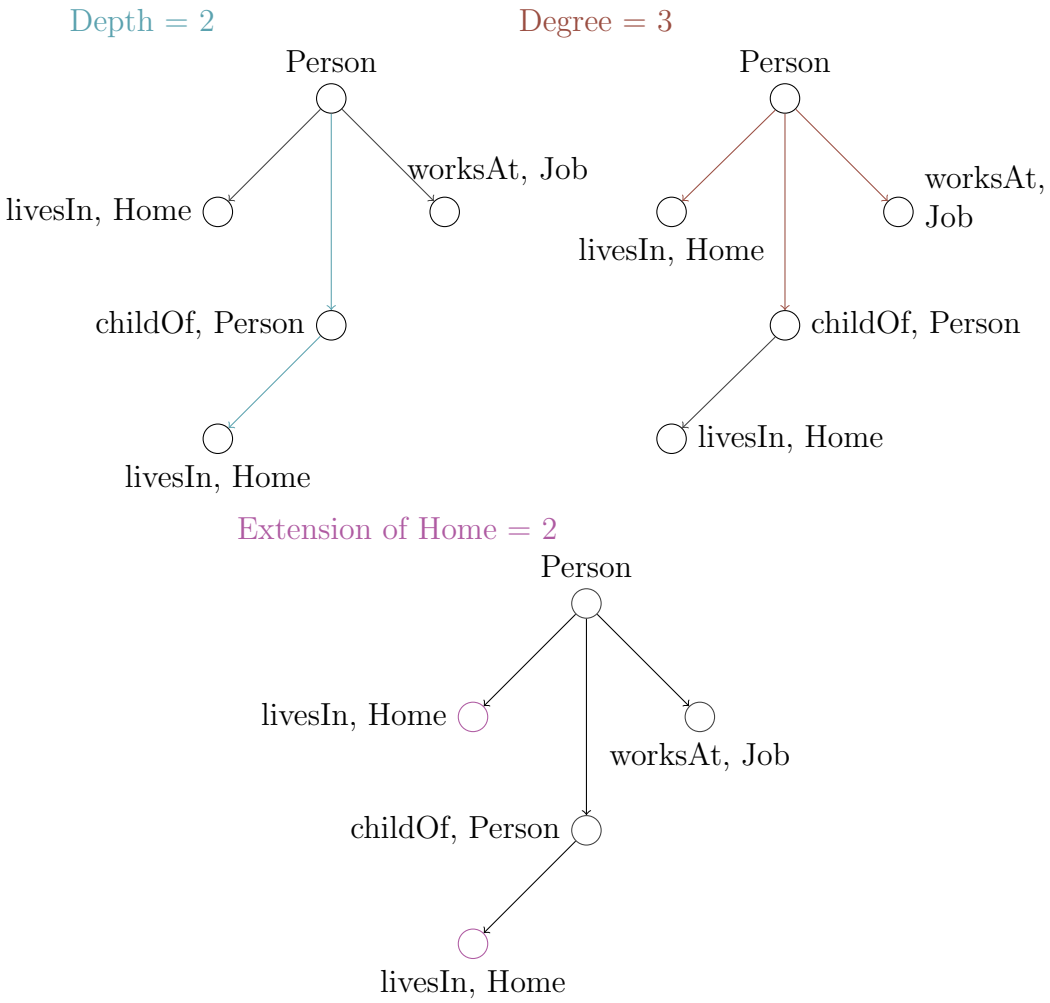
- $\delta(q_{next}) = \S \vee (\neg\S \wedge q_{measure} \wedge q)$

- $\delta(q) = \bigwedge_{1 \leq i \leq k} \Box_i q$
- $\delta(q_{measure}) = (q_E \wedge 1) \vee q_{E'}$

For all states $q_D \in Q_D$, the relevant transitions from $\mathcal{A}'_{C,\mathcal{T}}$ are reused.

In all three examples, the underlying idea of the abstract automaton can be reused for the specific $\mathcal{ALC}$ related feature and is intended to serve as a general template for features of this category. In the following, the example of a society from the introduction is revisited in order to demonstrate the defined features. Let $C = \exists$worksAt.Job $\sqcap \exists$childOf.Person and $\mathcal{T}$ as $\{\text{Person} \sqsubseteq \exists\text{livesIn.Home}\}$.

Example 4



5.2 Using measurements as model requirement

Building on these automata, two methods can be formulated to utilize measurements as model requirements. Let f be a feature function and $n \in \mathbb{N}$ some threshold we want to apply over T. Desired is an ata $\mathcal{A}_{f,\leq n}$, such that $T \in L(\mathcal{A}_{f,\leq n}) \iff f(T_{init}) \leq n$. Especially for automata that compare nodes, it is more intuitive to modify the weighted automaton to directly check a threshold. We can reformulate $\mathcal{A}_{degree}$ without weights, such that it tests for every node if the number of children is smaller than n for some $n \in \mathbb{N}_k$. Therefore, $\mathcal{A}_{degree,\leq n}$ is a normal ata and $\Box$-fixations are no longer necessary. $\mathcal{A}_{degree,\leq n} = (\Sigma, \{q, q_{next}, q_{\#children}\}, q, \delta)$, where δ is defined as

- $\delta(q) = q_{\#children} \wedge q_{next}$
- $\delta(q_{next}) = \S \vee \Box q$
- $\delta(q_{\#children}) = \Box_{n+1}\texttt{false}$

Every tree is accepted where no node u in T_{init} exists, with some $ui \in dom_T$ and ui has the index $n+1$. The same idea can be applied to $\mathcal{A}_{depth}$ only that a counter that keeps track of the remaining allowed depth is necessary. Again n will be the threshold, which in this case is the highest allowed depth. $\mathcal{A}_{depth,\leq n} = (\Sigma, \{(q,i), (q_{next}, i) \mid 0 \leq i \leq n\}, (q_{next}, n), \delta)$, where δ is defined as

- $\delta(q_{next}, i) = \S \vee (q, \omega)$;
- $\delta(q, i) = \Box\texttt{false}$ if $i = 0$.
- $\delta(q, i) = \Box(q_{next}, i-1)$ if $i > 0$.

For both $\mathcal{A}_{degree,\leq n}$ and $\mathcal{A}_{depth,\leq n}$ it is easy to show that they accept exactly those trees where the initial part satisfies the feature requirement.

Even though more intuitive, for some cases this approach is not universally applicable. For other cases where constructing the weighted alternating tree automaton is simpler, the cut-point automaton $\mathcal{A}_{\leq n}$ can be used. For the sake of completeness it is specified here once again. The original definition can be found in [1, Def. 7].

Definition 8. *Given a wata $\mathcal{A} = (\Sigma, Q, q_0, \delta)$, the cut-point automaton $\mathcal{A}_{\leq n} = (\Sigma, Q', q_0', \delta')$ for the threshold $n \in \mathbb{N}$ is an ata defined as follows:*
$Q' = \{(q,i) \in Q \times \mathbb{N} \mid i \leq n\} \cup \{q_0'\}$, *with δ' defined as*

$$\delta'(q_0') = \bigvee_{0 \leq i \leq n} (q_0, i)$$

$$\delta'((q,i)) = \delta(q) \text{ if } \delta(q) = \sigma, \neg\sigma, \texttt{true}\ , \texttt{false}$$

$$\delta'((q,i)) = \texttt{true} \quad \text{if } \delta(q) = j \leq i$$

$$\delta'((q,i)) = \texttt{false} \text{ if } \delta(q) = j > i$$

$$\delta'((q,i)) = \Diamond(q',i) \text{ if } \delta(q) = \Diamond q'$$

$$\delta'((q,i)) = \Box(q',i) \text{ if } \delta(q) = \Box q'$$

$$\delta'((q,i)) = (q_1,i) \vee (q_2,i) \text{ if } \delta(q) = q_1 \vee q_2$$

$$\delta'((q,i)) = \bigvee_{0 \leq j \leq i} (q_1,j) \wedge (q_2, i-j) \text{ if } \delta(q) = q_1 \wedge q_2$$

As stated in [1, Prop. 8] $L(\mathcal{A}_{\leq n}) = \{T \in Tree(\mathcal{P}(\Sigma)) \mid \|\mathcal{A}\|(T) \leq n\}$. The cut-point automaton $\mathcal{A}_{\leq n}$ has $O(n \cdot q)$ states, where q is the number of states of the weighted automaton $\mathcal{A}$. Thus, if n is encoded unary, this construction is polynomial, otherwise it is exponential.

By using these methods, requirements can now be placed on models in order to avoid unwanted features as described in the beginning in Section 2.3. Let $\mathcal{A}_{depth \leq 3, \#names \leq 1}$ be the disjoint union of $\mathcal{A}_{depth, \leq n}$ and $\mathcal{A}_{\#names \leq 1}$, where $\mathcal{A}_{\#names \leq 1}$ is the cut-point construction for $\mathcal{A}_{\#names}$. For this automaton, only the desired first tree from Example 1 would be accepted.

Final results

First $\mathcal{A}'_{C,\mathcal{T}}$ was defined, which accepts exactly those trees that represent the initial part of a tree model. For these initial parts different watas were specified and finally two possibilities to make requirements based on them. This was done assuming that trees are indexed, as shown in Section 4. Finally, we can combine all approaches. Let T be a $\mathcal{P}(\Sigma)$-labeled tree, $\mathcal{T}$ an $\mathcal{ALC}$-TBox of the form $\{\top \sqsubseteq C_{\mathcal{T}}\}$, C an $\mathcal{ALC}$-concept, f a feature function from $T_{init} \to \mathbb{N}$ and $\mathcal{A}_f$ some wata computing f. Accordingly $\mathcal{A}_{f,\leq n}$ accepts exactly those trees for which $f(T_{init}) \leq n$. The ata $\mathcal{A}'_{C,T,f,\leq n}$ is the disjoint union of $\mathcal{A}'_{C,\mathcal{T}}$ and $\mathcal{A}_{f,\leq n}$, such that q_α is the initial state and $\delta(q_\alpha) = q_0 \wedge q'_0$, where q_0 is the initial state of $\mathcal{A}'_{C,\mathcal{T}}$ and q'_0 the initial state of $\mathcal{A}_{f,\leq n}$.

Theorem 2. *$L(\mathcal{A}'_{C,T,f,\leq n}) \neq \emptyset \iff$ there exists a tree model of C w.r.t. $\mathcal{T}$ represented by T and $f(T_{init}) \leq n$.*

Proof (sketch). $L(\mathcal{A}'_{C,T,f,\leq n})$ will be the intersection of $L(\mathcal{A}'_{C,\mathcal{T}})$ and $L(\mathcal{A}_{f,\leq n})$.

"$\Rightarrow$" Let $L(\mathcal{A}'_{C,T,f,\leq n}) \neq \emptyset$, i.e. there exists a tree T, with $T \in L(\mathcal{A}'_{C,T,f,\leq n})$. Therefore applies $T \in L(\mathcal{A}'_{C,\mathcal{T}})$ and by Theorem 1 follows that T rep-

resents a tree model of C w.r.t. $\mathcal{T}$. Additionally T will be in $L(\mathcal{A}_{f,\leq n})$ concluding that $f(T_{init}) \leq n$

"$\Leftarrow$" Let T be a representation of a tree model of C w.r.t. $\mathcal{T}$ and $f(T_{init}) \leq n$. From Theorem 1 follows that $T \in L(\mathcal{A}'_{C,\mathcal{T}})$. According to the definition of $\mathcal{A}_{f,\leq n}$, the tree will also be in $L(\mathcal{A}_{f,\leq n})$. Therefore, $T \in L(\mathcal{A}'_{C,T,f,\leq n})$ and thus $L(\mathcal{A}'_{C,T,f,\leq n}) \neq \emptyset$.

□

$\mathcal{A}_{f,\leq n}$ could either be a direct construction, i.e. $\mathcal{A}_{degree,\leq n}$, or build utilizing the cut-point construction. If the construction of $\mathcal{A}_{f,\leq n}$ is at most double-exponential in the size of C and $C_{\mathcal{T}}$ then testing $L(\mathcal{A}'_{C,T,f,\leq n}) \neq \emptyset$ is in 3ExpTime. This follows from the 3ExpTime complexity for $\mathcal{A}'_{C,\mathcal{T}}$. This is not to say that it cannot be decided faster, possible optimisations are suggested in the conclusion. Note that this approach of combing $\mathcal{A}'_{C,\mathcal{T}}$ with $\mathcal{A}_{f,\leq n}$ can be extended to a set of feature measuring automata $\{\mathcal{A}_{f_1,\leq n_1}, \ldots, \mathcal{A}_{f_m,\leq n_m}\}$, where $\delta(\alpha) = q_0 \wedge q'_{0,1} \wedge \cdots \wedge q_{0,m}$. Respectively the complexity for this combination will be the maximum complexity of $\mathcal{A}'_{C,\mathcal{T}}$ and any used $\mathcal{A}_{f_i,\leq n_i}$.

6 Conclusion

In this thesis, a new aspect of description logic was investigated: the measurement of models. First, the idea of an initial part of a tree model as well as the notion of an indexed tree were established. Both approaches aimed to expand the possibilities of measurement. The initial part restricts the measurement to the redundancy-free part of a tree model so that conceptually equal elements are not taken into account more than once. For this purpose, the already defined automaton $\mathcal{A}_{C,\mathcal{T}}$ from [1], which tests the satisfiability of concepts, has been extended to $\mathcal{A}'_{C,\mathcal{T}}$, which can additionally recognize redundancies in trees. Indexed trees are trees where every child node has a unique index. Addressing child nodes by an index allows a finer control in the modeling of automata. Features like the degree of branching would otherwise not be measurable with the definitions for alternating tree automata and weighted alternating tree automata used in this thesis.

The presented features were divided into three categories. For each feature measuring automaton was shown how the underlying idea can be reused for different features of the same category. Finally, it has been demonstrated how these automata can be further utilized to constrain models to certain sizes. This can be realized by converting the feature measuring automaton using the cut-point construction [1]. In this context, the main focus was on constraining feature sizes to upper bounds. However, different approaches are possible too, e.g. ensuring a minimum size for concept extensions. As demonstrated by example in Section 5.2, the generic approach constraining tree models enables a straightforward combination of multiple automata, allowing multiple features to be ensured at once.

One potential application of this work is in explaining concept satisfiability. Recalling that a concept A is satisfiable w.r.t. a TBox $\mathcal{T}$ iff there exists an interpretation I, such that $A^I \neq \emptyset$ and for all $C \sqsubseteq D \in \mathcal{T} : C^I \subseteq D^I$. However, it is not always clear why a concept is satisfiable. How and why relations between concepts in $\mathcal{T}$ and A are correlated can be explored in an example model. In this regard restricting the depth, size, and possibly more features of such a model would help in keeping it concise and small. In this way, it would be much easier to determine the interrelationships.

The disadvantage of this approach is the triple exponential emptiness test of $\mathcal{A}'_{C,\mathcal{T}}$ (and thus all further automata). This results in a double exponential blowup compared to $\mathcal{A}_{C,\mathcal{T}}$. However, since the satisfiability check of $\mathcal{ALC}$-concepts is already exponential, the actual impact of this blowup in practice remains to be examined. In this regard, it might be possible to express

$\mathcal{A}'_{C,\mathcal{T}}$ as a weak alternating tree automaton for which the test $L(\mathcal{A}) \neq \emptyset$ is in polynomial time [6]. Finally, to what extent the approach could be extended to measure $\mathcal{ALC}$-ABoxes would be an interesting question for further research.

References

[1] Franz Baader and Andreas Ecke. "Reasoning with Prototypes in the Description Logic ALC Using Weighted Tree Automata". In: *Language and Automata Theory and Applications*. Springer International Publishing, 2016, pp. 63–75. DOI: 10.1007/978-3-319-30000-9_5. URL: https://doi.org/10.1007/978-3-319-30000-9_5.

[2] Franz Baader, Jan Hladik, and Rafael Peñaloza. "Automata can show PSpace results for description logics". In: *Information and Computation* 206.9 (2008). Special Issue: 1st International Conference on Language and Automata Theory and Applications (LATA 2007), pp. 1045–1056. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2008.03.006. URL: https://www.sciencedirect.com/science/article/pii/S0890540108000461.

[3] Franz Baader et al. *An Introduction to Description Logic*. Cambridge University Press, 2017. DOI: 10.1017/9781139025355. URL: https://doi.org/10.1017/9781139025355.

[4] H. Comon et al. *Tree Automata Techniques and Applications*. Available on: http://www.grappa.univ-lille3.fr/tata. release October, 12th 2007. 2007.

[5] Andreas Ecke. "Quantitative Methods for Similarity in Description Logics". PhD thesis. Dresden University of Technology, Germany, 2017. URL: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-223626.

[6] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. "Alternating automata, the weak monadic theory of the tree, and its complexity". In: *Automata, Languages and Programming*. Ed. by Laurent Kott. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 275–283. ISBN: 978-3-540-39859-2.

[7] Antonio A. Sánchez-Ruiz et al. "Measuring Similarity in Description Logics Using Refinement Operators". In: *Case-Based Reasoning Research and Development*. Springer Berlin Heidelberg, 2011, pp. 289–303. DOI: 10.1007/978-3-642-23291-6_22. URL: https://doi.org/10.1007/978-3-642-23291-6_22.

[8] Thomas Wilke. "Alternating tree automata, parity games, and modal µ-calculus". In: *Bull. Soc. Math. Belg* 8 (2001), p. 2001.