Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

An ExpTime Upper Bound for $\mathcal{ALC}$ with Integers

Nadia Labai , Magdalena Ortiz , Mantas Šimkus
Faculty of Informatics, TU Wien, Austria
{labai, simkus}@dbai.tuwien.ac.at, ortiz@kr.tuwien.ac.at

Abstract

Concrete domains, especially those that allow to compare features with numeric values, have long been recognized as a very desirable extension of description logics (DLs), and significant efforts have been invested into adding them to usual DLs while keeping the complexity of reasoning in check. For expressive DLs and in the presence of general TBoxes, for standard reasoning tasks like consistency, the most general decidability results are for the so-called ω-admissible domains, which are required to be dense. Supporting non-dense domains for features that range over integers or natural numbers remained largely open, despite often being singled out as a highly desirable extension. The decidability of some extensions of $\mathcal{ALC}$ with non-dense domains has been shown, but existing results rely on powerful machinery that does not allow to infer any elementary bounds on the complexity of the problem. In this paper, we study an extension of $\mathcal{ALC}$ with a rich integer domain that allows for comparisons (between features, and between features and constants coded in unary), and prove that consistency can be solved using automata-theoretic techniques in single exponential time, and thus has no higher worst-case complexity than standard $\mathcal{ALC}$. Our upper bounds apply to some extensions of DLs with concrete domains known from the literature, support general TBoxes, and allow for comparing values along paths of ordinary (not necessarily functional) roles.

1 Introduction

Concrete domains, especially those allowing to compare features with numeric values, are a very natural and useful extension of description logics. Their relevance was recognized since the early days of DLs (Baader and Hanschke 1991), and they arise in all kinds of application domains. Identifying extensions of DLs that keep the complexity of reasoning in check has been an ever present challenge for the DL community, and major research efforts have been devoted to that goal, see (Lutz 2002b) and its references. The best-known results so far are for the so-called ω-*admissible domains* which, among other requirements, must be dense. Decidability and tight complexity results have been established for several expressive DLs extended with ω-admissible domains based on the real or the rational numbers. However, non-dense numeric domains with the integer or natural numbers are not ω-admissible, and supporting them has been often singled out as an open challenge with significant practical implications (Lutz 2002a, 2002b).

To our knowledge, there are two decidability results for extensions of $\mathcal{ALC}$ with non-dense domains based on the integer numbers $\mathbb{Z}$. For some domains that support comparisons over the integers, decidability can be inferred from results on fragments of CTL* with constraints (Bozzelli and Gascon 2006). More recently, Carapelle and Turhan (2016) proved decidability for concrete domains that have the so-called *EHD-property* (for *existence of a homomorphism is definable*), which applies in particular to $\mathbb{Z}$ with comparison relations like '$=$' and '$<$'. However, neither of these works allow to infer any elementary bounds on the complexity of reasoning. The former result applies the theory of well-quasi-orders to some dedicated *graphose inequality systems*. The latter result reduces the satisfaction of the numeric constraints to satisfiability of a formula in a powerful extension of monadic second order logic with a *bounding quantifier*, which has been proved decidable over trees (Bojańczyk and Toruńczyk 2012). In both cases, the machinery stems from formalisms stronger than $\mathcal{ALC}$, and yields little insight on what is the additional cost of the concrete domain.

In this paper we propose an automata-theoretic algorithm tightly tailored for the DL $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, an extension of $\mathcal{ALCF}$ with a domain based on $\mathbb{Z}$ that follows the work of Carapelle and Turhan (2016). Not only do we obtain the first elementary complexity upper bounds, but in fact we obtain the best results that we could have hoped for: satisfiability is decidable in single exponential time, and thus not harder than for plain $\mathcal{ALC}$. The upper bound also applies to other approaches to concrete domains, and it extends to some domains over the real numbers that include unary predicates for asserting that some numbers must be integer or natural. Crucially, our setting accommodates general TBoxes, and allows to access the concrete domain along arbitrary paths of ordinary roles, and not only of functional ones. To our knowledge, this is the first decidability result with both of these features, even for ω-admissible domains.

Our upper bound is obtained using automata-theoretic techniques. Concretely, we rely on a suitable notion of the tree model property, and build a non-deterministic automaton on infinite trees that accepts representations of models

of the input. The key challenge in the presence of TBoxes comes from verifying whether an assignment of integer values along infinite paths exists. While an infinite path of ever increasing or ever decreasing values always exists, unsatisfiability of non-dense domains can arise from requiring an infinite number of integers that are larger than some integer and smaller than another. However, identifying that a given input enforces an infinite sequence of integers between two bounds may require us to identify, for example, if two infinite paths in the model meet at ever increasing distances. It is far from apparent how to detect this kind of very non-local behavior in standard automata, and we could not identify an automata-verifiable condition that precisely characterizes it. Instead, we use a condition similar to the one proposed for constraint LTL by Demri and D'Souza (2007), which is necessary on all trees, and sufficient on *regular* ones, and appeal to Rabin's theorem to obtain a sound and complete satisfiability test. The full proofs can be found in the extended version (Labai, Ortiz, and Šimkus 2020).

Related work The first DLs with concrete domains were introduced by Baader and Hanschke (1991), where concrete values are connected via paths of functional roles, often called *feature paths*. They showed that pure concept satisfiability is decidable for concrete domains $\mathcal{D}$ that are *admissible*, that is, satisfiability of conjunctions of predicates from $\mathcal{D}$ is decidable, and its predicates are closed under negation. Generalizations of this result and tight complexity bounds for specific settings were obtained in the following years. For example, concept satisfiability is PSPACE-complete under certain assumptions (Lutz 2002c). Adding acyclic TBoxes increases the complexity to NEXPTIME, and general TBoxes easily result in undecidability (Lutz 2004b). It remains decidable if the paths to concrete domains are restricted to single functional roles (Haarslev, Möller, and Wessel 2001). Lutz also studied specific concrete domains, for example for temporal reasoning (2004a), and summarized key results in a survey paper (2002b).

Later research relaxed the requirements on the concrete domain, and the most general results so far are for extensions of $\mathcal{ALC}(\mathcal{C})$ with *ω-admissible domains*, where concept satisfiability w.r.t. to general TBoxes remains decidable (Lutz and Milicic 2007). However, this and related results assume two key restrictions that we relax in our work: the concrete domain is dense, and only functional roles occur in the paths connecting to the concrete domains. Both restrictions are also present in $\mathbb{Q}$-$\mathcal{SHIQ}$, an extension of $\mathcal{SHIQ}$ with comparison predicates over the rational numbers, for which concept satisfiability w.r.t. general TBoxes is EXPTIME-complete. The logic we consider is closely related to $\mathbb{Q}$-$\mathcal{SHIQ}$. It includes the $\mathcal{ALCF}$ fragment of $\mathbb{Q}$-$\mathcal{SHIQ}$, but additionally allows us to replace the rational numbers by integers or naturals. Our EXPTIME upper bound also applies to the extension of $\mathbb{Q}$-$\mathcal{SHIQ}$ with an int or nat predicate to make only *some* values integer or natural, and, under certain restrictions, to its extension with arbitrary role paths.

Concerning the latter extension, already the seminal work of Baader and Hanschke (1991) points out the potential usefulness of allowing referral to the concrete domains also along paths of regular roles, but this easily results in undecidability. For example, such an extension of $\mathcal{ALC}$ known as $\mathcal{ALCFP}(\mathcal{D})$ is undecidable for any so-called *arithmetic domain* $\mathcal{D}$ (Lutz 2004b). However, $\mathcal{Z}_c$ and its analogue over the real numbers $\mathcal{R}_c$ are not arithmetic, and the corresponding DLs $\mathcal{ALCFP}(\mathcal{Z}_c)$ and $\mathcal{ALCFP}(\mathcal{R}_c)$ do not seem to have been studied before. By encoding these logics into $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ and $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_c)$, we prove that their satisfiability problem is decidable and obtain upper complexity bounds (which are tight under some restrictions).

Finally, we remark that the extensions of DLs with concrete domains that we consider here are closely related to constraint temporal logics. Our logic $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ subsumes constraint LTL as defined in (Demri and D'Souza 2007), whose satisfiability problem is PSPACE complete. It is in turn subsumed by constraint CTL*, and more specifically, by a fragment of it called CEF$^+$ in (Bozzelli and Gascon 2006), which unlike full CTL*, has a decidable satisfiability problem, but for which no tight complexity bounds are known. Although much of the work on concrete domains in the last decade has focused on lightweight DLs like DL-Lite (e.g. (Baader, Borgwardt, and Lippmann 2017; Poggi et al. 2008; Savkovic and Calvanese 2012; Artale, Ryzhikov, and Kontchakov 2012)), some advances in the area of constraint CTL (Carapelle, Kartzow, and Lohrey 2016) have inspired the study of expressive extensions that had long remained an open problem, like the ones considered here (Carapelle and Turhan 2016).

2 The $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ Description Logic

The DL $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$ was introduced by Carapelle and Turhan (2016) for arbitrary domains $\mathcal{D}$. Here we instantiate this DL with the concrete domain $\mathcal{Z}_c$ that is defined as $\mathbb{Z}$ equipped with the standard binary equality and comparison relations '$=$' and '$<$', as well as a family of unary relations for comparing with an integer constant.

Definition 1 (Syntax of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$)**.** *Let* Reg *be a countably infinite set of* registers *(also known as* concrete features*). A* register term *is an expression of the form $S^k x$, where $x \in \mathsf{Reg}$ and $k \geq 0$ is an integer. An* atomic constraint *is an expression of the form (i) $t = t'$, (ii) $t < t'$, or (iii) $t = c$, where t, t' are register terms, and $c \in \mathbb{Z}$. A* (complex) constraint Θ *is an expression built from atomic constraints using the Boolean connectives $\neg, \wedge$ and $\vee$. The depth of Θ (in symbols, $depth(\Theta)$) is the maximal d such that some register term $S^d x$ appears in Θ.*

Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be countably infinite sets of concept *and* role names*, respectively. We further assume an infinite set $\mathsf{N_F} \subseteq \mathsf{N_R}$ of* functional *role names. A* role path *P is any finite sequence $r_1 \cdots r_n$ of role names, with $n \geq 0$. We use $|P|$ to denote the length of P, i.e. $|P| = n$. Note that the empty sequence is also a role path, which we denote with ϵ.*

$\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts are defined as follows:

$$C := A \mid \neg C \mid (C \sqcap C) \mid \exists r.C \mid \exists P.[\![\Theta]\!]$$

where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, P is a role path, and Θ is a constraint with $depth(\Theta) \leq |P|$. We use $C \sqcup D$ as an abbreviation of $\neg(\neg C \sqcap \neg D)$, and $\forall r.C$ as an abbreviation of

$\neg\exists r.\neg C$. *Moreover, we use* $\forall P.[\![\Theta]\!]$ *instead of* $\neg\exists P.[\![\neg\Theta]\!]$. *Concepts of the form* $D = \exists P.[\![\Theta]\!]$ *and* $D = \forall P.[\![\Theta]\!]$ *are called* path constraints*, and we let* $depth(D) = |P|$.

A TBox $\mathcal{T}$ *is any finite set of* axioms*, where each axiom has the form* $C \sqsubseteq D$ *for some concepts* C *and* D.

(Plain) $\mathcal{ALCF}$ *concepts and TBoxes are defined as in* $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ *but do not allow path constraints.*

We can now define the semantics of the considered DL.

Definition 2 (Semantics)**.** *An* interpretation *is a tuple* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta)$*, consisting of a non-empty set* $\Delta^{\mathcal{I}}$ *(called* domain*), a* register function $\beta : \Delta^{\mathcal{I}} \times \mathsf{Reg} \to \mathbb{Z}$*, and a* (plain) interpretation function $\cdot^{\mathcal{I}}$ *that assigns* $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ *to every concept name* $C \in \mathsf{N_C}$*,* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *to every role name* $r \in \mathsf{N_R}$*. We further require that* $\{(v, v'), (v, v'')\} \subseteq r^{\mathcal{I}}$ *implies* $v' = v''$ *for all* $r \in \mathsf{N_F}$*. Role paths denote tuples of elements. For a role path* $r_1 \cdots r_n$*, we define* $(r_1 \cdots r_n)^{\mathcal{I}}$ *as the set of all tuples* $(v_0, \ldots, v_n) \in \Delta^{n+1}$ *such that* $(v_0, v_1) \in r_1^{\mathcal{I}}, \ldots, (v_{n-1}, v_n) \in r_n^{\mathcal{I}}$.

For an interpretation $\mathcal{I}$ *and a tuple* $\vec{v} = (v_0, \ldots, v_n)$ *of elements in* $\Delta^{\mathcal{I}}$*, we define the following, where* $\theta \in \{=, <\}$ *and* $c \in \mathbb{Z}$*:*

- $\mathcal{I}, \vec{v} \models \theta(S^i x, S^j y)$ *iff* $\beta(v_i, x)\theta\beta(v_j, y)$*;*
- $\mathcal{I}, \vec{v} \models S^i x = c$ *iff* $\beta(v_i, x) = c$*;*
- $\mathcal{I}, \vec{v} \models \Theta_1 \wedge \Theta_2$ *iff* $\mathcal{I}, \vec{v} \models \Theta_1$ *and* $\mathcal{I}, \vec{v} \models \Theta_2$*;*
- $\mathcal{I}, \vec{v} \models \Theta_1 \vee \Theta_2$ *iff* $\mathcal{I}, \vec{v} \models \Theta_1$ *or* $\mathcal{I}, \vec{v} \models \Theta_2$*;*
- $\mathcal{I}, \vec{v} \models \neg\Theta$ *iff* $\mathcal{I}, \vec{v} \not\models \Theta$.

Now the function $\cdot^{\mathcal{I}}$ *is extended to complex concepts as follows:*

- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ *and* $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$*,*
- $(\exists r.C)^{\mathcal{I}} = \{v \mid (v, v') \in r^{\mathcal{I}}, v' \in C^{\mathcal{I}}\}$*, and*
- $(\exists P.[\![\Theta]\!])^{\mathcal{I}} = \{u \mid (u, \vec{v}) \in P^{\mathcal{I}}$ *and* $\mathcal{I}, (u, \vec{v}) \models \Theta\}$.

An interpretation $\mathcal{I}$ *is a* model *of a TBox* $\mathcal{T}$*, if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *for all* $C \sqsubseteq D \in \mathcal{T}$*. We say that a concept* C *is* satisfiable w.r.t. $\mathcal{T}$ *if there is a model* $\mathcal{I}$ *of* $\mathcal{T}$ *with* $C^{\mathcal{I}} \neq \emptyset$.

Example 1. *The TBox with the axiom* $\top \sqsubseteq \exists r.[\![S^0 x < S^1 x]\!]$ *enforces an infinite chain of objects whose* x *registers store increasing integer values. This witnesses that* $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ *does not enjoy the finite model property.*

Example 2. *The next axiom identifies* active departments *whose employees lead projects started in the last two years:*

$$\exists employs\, leadsProject[\![S^2 year > 2018]\!] \sqsubseteq ActiveDept$$

In general, the roles 'employs' and 'leadsProject' need not be functional, but this is not a problem in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$.

Tree model property The automata-based techniques we employ in this paper rely on the *tree model property* of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. We recall the definition of tree-shaped models (cf. Carapelle and Turhan (2016)). For $n \geq 1$, let $[n] = \{1, \ldots, n\}$. We say $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta)$ is *an (n-)tree interpretation* if $\Delta^{\mathcal{I}} = [n]^\star$ for some $n \geq 1$, and for every $u, v \in \Delta^{\mathcal{I}}$, we have that $(u, v) \in r^{\mathcal{I}}$ for some $r \in \mathsf{N_R}$ iff $v = u\gamma$ for some $\gamma \in [n]$. Let $\gamma_1, \ldots, \gamma_k \in [n]$. If $u\gamma_k \in \Delta^{\mathcal{I}}$, we call u the *parent* of $u\gamma_k$, and if $v = u\gamma_1 \cdots \gamma_k \in \Delta^{\mathcal{I}}$, we call u the *k-th ancestor* of v.

The following theorem will allow us to focus on n-trees for our technical developments:

Theorem 3 ((Carapelle and Turhan 2016))**.** *Let* $\hat{\mathcal{C}}$*,* $\hat{\mathcal{T}}$ *be in negation normal form, where* d *is the maximal depth of an existential path constraint in* $\hat{\mathcal{C}}$ *or* $\hat{\mathcal{T}}$*, and* e *the number of existentially quantified subconcepts in* $\hat{\mathcal{C}}$ *or* $\hat{\mathcal{T}}$*. If* $\hat{\mathcal{C}}$ *is satisfiable w.r.t.* $\hat{\mathcal{T}}$*, then it has an* n*-tree model where* $n = d \cdot e$.

3 A Tight Upper Bound for Satisfiability

In this section we present our main result: an algorithm for deciding $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concept satisfiability w.r.t. to general TBoxes in *single exponential time*. The algorithm uses automata on infinite trees, and reduces the satisfiability test to the emptiness of a suitable automata. But first we bring $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts and TBoxes into a simpler shape that facilitates the later developments.

3.1 Atomic Normal Form

Here we go from a concept $\mathcal{C}'$ and TBox $\mathcal{T}'$ in general form to equisatisfiable $\hat{\mathcal{C}}$ and $\hat{\mathcal{T}}$ in *atomic normal form*, where the path constraint are of length 1 and the register constraints are atomic. This conversion relies on the tree model property of $\mathcal{ALCF}^{\mathcal{P}}$ and on $\mathcal{Z}_c$ being negation-closed. That is, the negation of an atomic relation can be expressed without negation via other relations; for $<$ and $=$ negation can be removed using only one disjunction, and for $= c$ negation can be removed using one conjunction with one disjunction and one fresh register name.

Definition 4 (Atomic normal form)**.** *An* $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$*-concept is in* atomic normal form (ANF) *if for every* $\exists P.[\![\Theta]\!]$ *and* $\forall P.[\![\Theta]\!]$ *that appears in it,* Θ *is an atomic constraint and* $|P| \leq 1$*. A TBox* $\mathcal{T}$ *is in ANF if the TBox-concept* $\bigsqcap_{C \sqsubseteq D \in \mathcal{T}}(\neg C \sqcup D)$ *is in ANF.*

Lemma 5. *Let* $\mathcal{C}'$ *and* $\mathcal{T}'$ *be a concept and a TBox in* $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$*. Then* $\mathcal{C}'$ *and* $\mathcal{T}'$ *can be transformed in polynomial time into* $\mathcal{C}$ *and* $\mathcal{T}$ *in ANF such that* $\mathcal{C}$ *is satisfiable w.r.t.* $\mathcal{T}$ *iff* $\mathcal{C}'$ *is satisfiable w.r.t.* $\mathcal{T}'$.

Proof sketch. We can convert $\mathcal{C}'$ and $\mathcal{T}'$ to negation normal form and then remove negation from atomic constraints using $\wedge$, $\vee$, and at most one fresh register name per constraint, all in linear time. Therefore we assume that $\mathcal{C}'$ and $\mathcal{T}'$ are negation free. Next, relying on the tree model property, we copy at each node u the registers of its ancestors that may occur in the same constraints as u's own registers. For this we propagate the register values of the ancestors one step at a time with axioms

$$\top \sqsubseteq \forall r.[\![S^1 x^k_{i,P} = S^0 x^{k-1}_{i,P}]\!]$$

We define a TBox $\mathcal{T}_{\mathsf{prop}}$ that contains such an axiom for each appropriate role name r and (fresh) register names associated with role paths P and depth k of path constraints used in $\mathcal{C}'$ and $\mathcal{T}'$. Note that along every path P, the TBox $\mathcal{T}_{\mathsf{prop}}$ propagates values into copy-registers associated with *all* paths appearing in $\mathcal{C}'$ or $\mathcal{T}'$, not just into the copy-registers associated with P. We will later restrict our attention to the relevant registers depending on context. The following claim is proved with a straightforward inductive construction:

Claim 6. *Every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$ contains a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$, and every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ can be expanded to a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$.*

For a role path P and an atomic constraint Θ, let $\mathsf{loc}(\Theta, P)$ denote the constraint obtained from Θ by replacing each occurrence of $S^j x_i^0$ with $S^0 x_{i,P}^{|P|-j}$. In the next step, we create some "test" concept names and axioms that will allow to check whether a given constraint is satisfied in a certain path in a tree model. For each (sub)constraint Θ and a role path P that appear in $\mathcal{C}'$ or $\mathcal{T}'$, take a fresh concept name $T_{P,\Theta}$ and add to a TBox $\mathcal{T}_{\mathsf{loc}}$ the following axioms (recall that $\mathcal{C}'$ and $\mathcal{T}'$ are negation free):

($\mathbf{A}_1$) $T_{P,\Theta} \equiv T_{P,\Theta_1} \sqcap T_{P,\Theta_2}$ if $\Theta = \Theta_1 \wedge \Theta_2$

($\mathbf{A}_2$) $T_{P,\Theta} \equiv T_{P,\Theta_1} \sqcup T_{P,\Theta_1}$ if $\Theta = \Theta_1 \vee \Theta_2$

($\mathbf{A}_3$) $T_{P,\Theta} \equiv \exists \epsilon [\![\mathsf{loc}(\Theta, P)]\!]$ if Θ is an atomic constraint.

We make two claims about combining $\mathcal{T}_{\mathsf{prop}}$ with $\mathcal{T}_{\mathsf{loc}}$. The first is that we can continue expanding the initial tree model:

Claim 7. *Every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$ can be expanded to a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, and every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ is a tree model of $\mathcal{C}'$ w.r.t $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$.*

The above claim follows by induction on Θ. Next, we claim that $\mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ indeed relates the satisfaction of path constraints to membership in the test concepts:

Claim 8. *Let $\mathcal{J}$ be a tree model of $\mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, and let P be a role path and Θ a constraint appearing in $\mathcal{C}'$ or $\mathcal{T}'$. Then it holds that*

1. *$\mathcal{J}$ contains a P-path $e_0, \ldots, e_{|P|}$ and if $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$, then the path $e_0, \ldots, e_{|P|}$ satisfies Θ in $\mathcal{J}$;*
2. *if $\mathcal{J}$ contains a P-path $e_0, \ldots, e_{|P|}$ that satisfies Θ, then $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$.*

Now we are ready to rewrite $\mathcal{C}'$ and $\mathcal{T}'$ into ANF using the locally available copy-registers and the test concept names; Given a concept D and a role path $P = r_1 \cdots r_n$, we write $\exists P.D$ as shorthand for $\exists r_1(\exists r_2(\cdots(\exists r_n.D)\cdots))$, and similarly for $\forall P.D$. Let $\mathcal{C}$ and $\mathcal{T}^*$ be obtained from $\mathcal{C}'$ and $\mathcal{T}'$, respectively, by replacing every concept $\exists P.[\![\Theta]\!]$ by $\exists P.T_{P,\Theta}$ and every $\forall P.[\![\Theta]\!]$ by $\forall P.T_{P,\Theta}$. Our desired normalization is $\mathcal{C}$ equipped with the TBox $\mathcal{T} = \mathcal{T}^* \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$.

Given a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$, by chaining Claim 6 and Claim 7, we get a tree model $\mathcal{J}$ of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, and by applying Claim 8 we get that

$$(\exists P.[\![\Theta]\!])^{\mathcal{J}} = (\exists P.T_{P,\Theta})^{\mathcal{J}}, \quad (\forall P.[\![\Theta]\!])^{\mathcal{J}} = (\forall P.T_{P,\Theta})^{\mathcal{J}}$$

Hence $\mathcal{J}$ is also a tree model of $\mathcal{C}$ w.r.t. $\mathcal{T}$.

Given a tree model $\mathcal{J}$ of $\mathcal{C}$ w.r.t. $\mathcal{T}$, again by applying Claim 8 we get that $\mathcal{J}$ is also a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ (and in particular w.r.t. $\mathcal{T}'$). □

3.2 Abstractions and Constraint Graphs

To check satisfiability of $\mathcal{C}$ w.r.t. $\mathcal{T}$, we follow the approach of (Carapelle and Turhan 2016) and split the task into two checks: a satisfiability check for an *abstracted* version of $\mathcal{T}, \mathcal{C}$, which is in plain $\mathcal{ALCF}$, and an *embeddability check* for so-called *constraint graphs*. We recall the definitions of abstracted $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts and constraint graphs from (Carapelle and Turhan 2016), adapted to our context.

Definition 9 (Abstraction)**.** *Consider a path constraint $E = \exists r.[\![\Theta]\!]$, where Θ is an atomic constraint. Let $B \in \mathbf{B}$ be a fresh concept name, which we call the* placeholder *of Θ. The* abstraction *of E is defined as*

$$E_a = \exists r.B$$

The abstraction of a universal path constraint $E' = \forall r.[\![\Theta']\!]$ is analogous. If $r = \epsilon$, then the abstraction is simply B.

The abstractions of concepts and TBoxes given in ANF are the (plain $\mathcal{ALCF}$) concepts and TBoxes obtained by replacing all path constraints with their abstracted versions.

Let $\mathcal{C}_a$, $\mathcal{T}_a$ be the abstractions of $\mathcal{C}$ and $\mathcal{T}$, respectively. Let $\mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ be the set of register names used in $\mathcal{C}$ and $\mathcal{T}$.

The constraint graph of a plain tree-shaped interpretation indicates how the values of its registers participate in relevant relations. (In)equalities between registers are represented as graph edges, and (in)equalities with constants are stored as node labels.

Definition 10 (Constraint graph)**.** *Let $\mathcal{I}_a = (\Delta^{\mathcal{I}_a}, \cdot^{\mathcal{I}_a})$ be a plain tree-shaped interpretation of $\mathcal{C}_a, \mathcal{T}_a$. The* constraint graph *of $\mathcal{I}_a$ is the directed partially labeled graph $\mathcal{G}_{\mathcal{I}_a} = (V, E, \lambda)$ where $V = \Delta^{\mathcal{I}_a} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$, $\lambda : V \to 2^{\mathbf{B}}$, and $E = E_< \cup E_=$ is such that, for every $(v, y), (u, x) \in V$,*

1. *$((v, y), (u, x)) \in E_<$ if and only if either*
   - *$u = v$, $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y < S^0 x$,*
   - *u is the parent of $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^1 y < S^0 x$, or*
   - *v is the parent of $u \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y < S^1 x$.*
2. *$((v, y), (u, x)) \in E_=$ if and only if*
   - *$u = v$, $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y = S^0 x$,*
   - *u is the parent of $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^1 y = S^0 x$, or*
   - *v is the parent of $u \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y = S^1 x$.*

In addition, for a placeholder B for $S^0 x = c$, we have that $B \in \lambda(u, x)$ if and only if $u \in B^{\mathcal{I}_a}$.

When the interpretation is clear from context, we write $\mathcal{G}$.

We say a constraint graph $\mathcal{G}$ is *embeddable* into $\mathcal{Z}_c$ if there is an integer assignment κ to the vertices V of $\mathcal{G}$ such that for every $(u, x), (v, y) \in V$, if $((v, y), (u, x)) \in E_<$ then $\kappa(u, w) < \kappa(v, y)$ (and similarly for $E_=$), and if $B \in \lambda(u, x)$ is a placeholder for $S^0 x = c$, then $\kappa(u, x) = c$.

Example 3. *The left hand side of Figure 1 shows a constraint graph for an interpretation where each element satisfies $\exists \epsilon.[\![S^0 x < S^0 y]\!]$, the root and its right child satisfy $\exists r.[\![S^0 y > S^1 x]\!]$, the root and its left child satisfy*

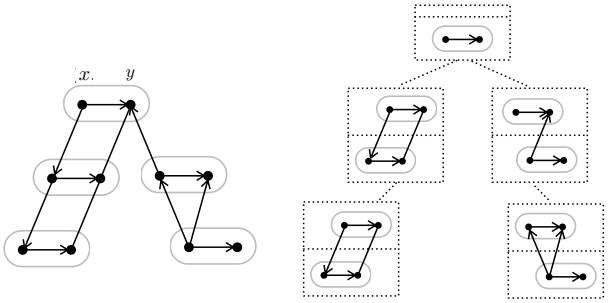

Figure 1: A constraint graph with registers x and y for each logical element (left) and its tree representation (right). The label λ is empty for all nodes and not shown.

$\exists r.[\![(S^0x < S^1x) \wedge (S^0y = S^1y)]\!]$, and the right child of the root additionally satisfies $\exists r.[\![(S^0x < S^1x) \wedge (S^0y < S^1x)]\!]$.

This constraint graph is embeddable into $\mathbb{Z}$. Consider, however, an infinite interpretation in which the leftmost branch of the constraint graph repeats infinitely. Then we would have paths from the x to the y register of the root involving any finite number of edges from $E_<$, which would imply that the integer values assigned to these registers must have infinitely many different integer values between them. Thus in this case, the graph would not be embeddable.

Abstractions and constraint graphs allow us to reduce $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ satisfiability to two separate checks:

Theorem 11 (Carapelle and Turhan, 2016)**.** *$\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$ if and only if there is a tree-shaped $\mathcal{I}_a \models_{\mathcal{T}_a} \mathcal{C}_a$ such that $\mathcal{G}_{\mathcal{I}_a}$ is embeddable into $\mathcal{Z}_c$.*

3.3 Embeddability Condition

Our first aim is to test embeddability using tree automata. For this, we represent constraint graphs and augmentations thereof as trees. In a nutshell, our *tree representations* are tree decompositions (cf. (Diestel 2012)) where each bag holds the subgraph induced by a logical element and its parent. Since our constraints have maximal depth 1, we can do this using an alphabet that stores information about two logical elements: a parent at the top (top) and a child at the bottom (bot).

This is illustrated in Figure 1, where the tree representation of the constraint graph is shown on the right hand side. Note that the bottom part of the label of each vertex induces the same graph at the top part of the label of each of its children, and that the label of the root vertex has no top row.

For these representations we use two copies x^{top} and x^{bot} of each $x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$, and call the respective sets $\mathsf{Reg}^{\mathrm{top}}_{\mathcal{C},\mathcal{T}}$ and $\mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$. The relevant information about (in)equalities with constants is stored as a partial labeling in these tree representations. Let c_0 be the smallest integer used in either $\mathcal{C}$ or $\mathcal{T}$ and let c_α be the largest. If no integers were used, set $c_0 = c_\alpha = 0$. Denote by $[c_0, c_\alpha]$ the range of integers between c_0 and c_α, inclusive. Let $\mathbf{U} = \{U_{<c_0}, U_{c_\alpha<}\} \cup \{U_c \mid c \in [c_0, c_\alpha]\}$ be fresh labels. Let Σ be the set of partially $\mathbf{U}$-labeled graphs where the vertex set is either exactly $V = \mathsf{Reg}^{\mathrm{top}}_{\mathcal{C},\mathcal{T}} \cup \mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$ or $V = \mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$, and $E = E_< \cup E_=$.

Definition 12 (Tree representation of constraint graph)**.** *Let $\mathcal{G}$ be the constraint graph of some plain interpretation $\mathcal{I}_a$. For $u \in \Delta^{\mathcal{I}}$ with parent v, define $X(u)$ as the subgraph of $\mathcal{G}$ induced by $\{(u,x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\} \cup \{(v,x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\}$. For $u = \varepsilon$, define $X(u)$ as the subgraph of $\mathcal{G}$ induced by $\{(u,x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\}$. Let $Y(u)$ be the following partially $\mathbf{U}$-labeled graph:*

- *The vertices of $Y(u)$ are obtained from $X(u)$ by renaming $(u,x) \mapsto x^{\mathrm{bot}}$ and $(v,x) \mapsto x^{\mathrm{top}}$ for every $x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$.*
- *The edges of $Y(u)$ are exactly those of $X(u)$ (under the renaming).*
- *We have $U_c(x^{\mathrm{bot}})$ if and only if (u,x) is labeled with a placeholder for equality with c, and similarly for x^{top}.*

The tree representation $\mathrm{Tr}(\mathcal{G})$ *of $\mathcal{G}$ is the tree over Σ where $\mathrm{Tr}(\mathcal{G})(u) = Y(u)$.*

Rather than considering tree representations where nodes are labeled with arbitrary graphs from Σ, it will be convenient to consider trees over a restricted alphabet that contains only graphs that have been enriched with either additional or implicit information in a maximal consistent way.

Definition 13 (Frame)**.** *A* frame *is a graph in Σ such that:*

1. *there is an edge between every pair of vertices*
2. *there are no strict cycles, i.e. no cycles that include an edge from $E_<$*
3. *if $(x,y) \in E_=$ then also $(y,x) \in E_=$*
4. *every vertex must have exactly one of the labels in $\mathbf{U}$*
5. *if $(x,y) \in E_=$ then x and y have the same label from $\mathbf{U}$, and if x and y have the same label from $\mathbf{U} \setminus \{U_{<c_0}, U_{c_\alpha<}\}$ then $(x,y) \in E_=$*
6. *if $(x,y) \in E_<$ then either (a) $U_{<c_0}(x)$, or (b) $U_{c_\alpha<}(y)$, or (c) $U_{c_i}(x)$ and $U_{c_j}(y)$ with $c_i, c_j \in [c_0, c_\alpha]$ and $c_i < c_j$.*

We denote the alphabet of frames by Σ_{fr}.

Definition 14 (Framified constraint graph)**.** *We say that an augmentation $\mathcal{G}_{\mathrm{fr}}$ of a constraint graph $\mathcal{G}$ is a* framified constraint graph *if its tree representation is over Σ_{fr}.*

Note that a constraint graph may have multiple framifications; e.g. if not all registers are compared to a constant. It may also have no framifications; e.g. if it contains a strict cycle. In fact, a framification may not introduce strict cycles.

Lemma 15. *Let $\mathcal{G}_{\mathrm{fr}}$ be a framified constraint graph. Then there are no strict cycles in $\mathcal{G}_{\mathrm{fr}}$.*

Proof Sketch. We show by induction that if a strict cycle spanning the registers of k logical elements exists, then due to the existence of an edge between every pair of vertices, there is also a strict cycle spanning the registers of $k-1$ logical elements. Repeating until the strict cycle spans at most 2 logical elements, we obtain a contradiction to Def. 13. □

An embeddable constraint graph can always be framified.

Observation 16. *Let $\mathcal{G}$ be an embeddable constraint graph. Then there exists a framification of $\mathcal{G}$.*

In the tree representation of (framified) constraint graphs, the bot part of a vertex coincides with the top of its children.

Definition 17 (Consistent frames)**.** *Let $\sigma_1, \sigma_2 \in \Sigma_{\mathrm{fr}}$. We call the pair (σ_1, σ_2)* consistent *if the following are equal:*
• *the subgraph induced by the $\mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$ vertices of σ_1, and*
• *the result of renaming each x^{top} to x^{bot} in the subgraph induced by the $\mathsf{Reg}^{\mathrm{top}}_{\mathcal{C},\mathcal{T}}$ vertices of σ_2.*

Not every tree T over Σ_{fr} corresponds to a framified constraint graph, but when all parent-child pairs are consistent, we can refer to the framified constraint graph T represents:

Definition 18. *Let T be a tree over Σ_{fr}. We call T* consistent *if the pair $(T(v), T(vh))$ is consistent for every $v \in [n]^\star$ and every $h \in [n]$. We denote by $\mathcal{G}^T_{\mathrm{fr}}$ the framified constraint graph with $\mathrm{Tr}(\mathcal{G}^T_{\mathrm{fr}}) = T$, and say that T represents $\mathcal{G}^T_{\mathrm{fr}}$.*

We use the following terminology for talking about paths.

Definition 19. *Let $\mathbf{w} = \gamma_1\gamma_2\cdots$ be a finite or infinite word over $[n]$ and let $u \in [n]^\star$.* A path along $\mathbf{w}$ from (u, x) *is a path of the form $p = (u, x) - (u\gamma_1, x_1) - (u\gamma_1\gamma_2, x_2) - \cdots$.*

An infinite path $p : \mathbb{N} \to \Delta \times \mathsf{Reg}$ is a forward path *if for every $n \in \mathbb{N}$, there is an edge from $p(i)$ to $p(i+1)$. It is a* backward path *if for every $n \in \mathbb{N}$, there is an edge from $p(i+1)$ to $p(i)$. The* strict length *of a finite path p is the number of strict edges in p. For an infinite path p, we say that p* is strict *if it has infinitely many strict edges.*

The following condition on framified constraint graphs will be crucial to deciding embeddability:

($\bigstar$) There are no $(u, x), (u, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ in $\mathcal{G}_{\mathrm{fr}}$ for which we have that: there exists an infinite $\mathbf{w} \in [n]^\omega$, and
1. an infinite forward path f from (u, x) along $\mathbf{w}$, and
2. an infinite backward path b from (u, y) along $\mathbf{w}$

such that f or b is strict, and such that for every $i \in \mathbb{N}$, there is a strict edge from $f(i)$ to $b(i)$.

Indeed, ($\bigstar$) is a necessary condition for embeddability:

Lemma 20. *If a constraint graph is embeddable, then it satisfies the condition ($\bigstar$).*

Proof Sketch. By contradiction. The existence of such a pair and paths would imply that the integers assigned to (u, x) and (u, y) have infinitely many different integers between them, since a path of strict length k from (u, x) to (u, y) implies there being a difference of at least k between their assigned values. □

Unfortunately, it is not sufficient in general.

Example 4 (From (Demri and D'Souza 2007))**.** *Figure 2 shows an example of a constraint graph; to avoid clutter, we omitted the edges augmented in its framification. It satisfies the condition ($\bigstar$) since there is no path with infinitely many strict edges. It is not embeddable into $\mathbb{Z}$: indeed, for any n, there is a path with at least n strict edges between x and z.*

Nonetheless, the condition will allow us to effectively test embeddability, since it is sufficient for *regular* framified constraint graphs.

Definition 21. *For an n-tree T over Σ, the* subtree rooted at *$w \in [n]^\star$ is the tree $T|_w(v) = T(wv)$ for all $v \in [n]^\star$.*

We say that an n-tree T over Σ is regular *if the set $\{T|_u \mid u \in [n]^\star\}$ of subtrees of T is finite. We say that a constraint graph is regular if its tree representation is regular.*

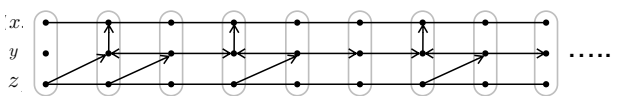

Figure 2: A constraint graph that satisfies ($\bigstar$) but is not embeddable into $\mathbb{Z}$.

The next key lemma is the most technical result of the paper.

Lemma 22. *Let $\mathcal{G}_{\mathrm{fr}}$ be a regular framified constraint graph. If $\mathcal{G}_{\mathrm{fr}}$ satisfies ($\bigstar$), then it is embeddable.*

Proof sketch. Let $\mathcal{G}_{\mathrm{fr}}$ be a regular framified constraint graph which is not embeddable. The heart of the proof is showing that there is a pair $(u, x), (u, y)$ and a finite path from (u, x) to (u, y) of a certain shape and positive strict length, which may be concatenated with itself indefinitely to obtain the desired f and b.

First, we show that there is a pair $(u, x), (u, y)$ such that for any $m \in \mathbb{N}$, there is a path from (u, x) to (u, y) of strict length at least m which only involves vertices whose logical element has the prefix u, that is, the path only involves vertices in the subtree rooted at u. However, the path may move down and up this subtree arbitrarily. We then use framification to extract from it a path p' of a specific shape, which first goes down along some w and then goes back up. The path p' may have reduced strict length, but we show a lower bound on the strict length of p' which is a function of m.

Next we use regularity to argue that for large enough m, the path p' becomes long enough that it essentially starts repeating itself, thus allowing us to extend it indefinitely (as well as the word w it runs long) to obtain the desired forward and backward paths; f is obtained by concatenating the downward portion of p' and b is obtained by concatenating the upward portion. The strict edges between f and b are given by the framification. □

3.4 A Rabin Tree Automaton for Embeddability

We still face two hurdles: verifying ($\bigstar$), and ensuring that satisfiable $\mathcal{C}$ w.r.t. $\mathcal{T}$ have a model with a *regular* constraint graph. We overcome both by using Rabin's tree automata.

Recall that the trees are over the alphabet of frames Σ_{fr} and are of degree n i.e. their nodes are over $[n]^\star$.

Definition 23 (Rabin tree automaton)**.** *A* Rabin tree automaton *over the alphabet Σ has the form $\mathcal{A} = (Q, q_0, \longrightarrow, \Omega)$ with a finite state set Q, initial state q_0, transition relation $\longrightarrow \subseteq Q \times \Sigma \times Q^n$, and $\Omega = \{(L_1, U_1), \ldots, (L_m, U_m)\}$ is a collection of "accepting pairs" of state sets $L_i, U_i \subseteq Q$. A* run *of $\mathcal{A}$ on a tree T is a map $\rho : [n]^\star \to Q$ with $\rho(\varepsilon) = q_0$ and $(\rho(w), T(w), \rho(w1), \ldots, \rho(wn)) \in \longrightarrow$ for $w \in [n]^\star$. For a path π in T and a run ρ denote by $\mathrm{In}(\rho \mid \pi)$ the set of states that appear infinitely often in the restriction of ρ to π. A run ρ of $\mathcal{A}$ is* successful *if*

$$\text{for all paths } \pi \text{ there exists an } i \in [m] \text{ with } \mathrm{In}(\rho \mid \pi) \cap L_i = \emptyset \text{ and } \mathrm{In}(\rho \mid \pi) \cap U_i \neq \emptyset.$$

A tree T is accepted *by the Rabin tree automaton if some run of $\mathcal{A}$ in T is successful.*

Theorem 24 (Rabin's Theorem (1972))**.** *Any non-empty Rabin recognizable set of trees contains a regular tree.*

Since condition (★) is necessary and sufficient for the embeddability of regular framified constraint graphs, we get:

Lemma 25. *Let $\mathcal{A}_{\mathrm{emb}}$ be a Rabin tree automaton that accepts exactly the consistent trees over Σ_{fr} satisfying (★). There is an embeddable constraint graph iff $L(\mathcal{A}_{\mathrm{emb}}) \neq \emptyset$.*

Proof. If there is an embeddable constraint graph $\mathcal{G}$, then it has some framification $\mathcal{G}_{\mathrm{fr}}$ (Observation 16), which satisfies the condition (★) (Lemma 20). Therefore the tree representation of $\mathcal{G}_{\mathrm{fr}}$ is accepted by $\mathcal{A}_{\mathrm{emb}}$ and $L(\mathcal{A}_{\mathrm{emb}}) \neq \emptyset$. For the other direction, assume $L(\mathcal{A}_{\mathrm{emb}}) \neq \emptyset$. Then by Rabin's Theorem, there is a regular tree $T \in L(\mathcal{A}_{\mathrm{emb}})$, which satisfies the condition (★). By Lemma 22, we have that the constraint graph represented by T is embeddable. □

Therefore it remains to show that the condition (★) is indeed verifiable by a Rabin tree automaton. We do this next.

Checking consistency of trees In our constructions of automata, it is useful to assume that they run on trees over Σ_{fr} that are consistent (in the sense of Definition 17), rather than complicating the constructions by incorporating the consistency check. Therefore we first describe an automaton $\mathcal{A}_{\mathrm{ct}}$ which accepts exactly the consistent trees, which we later intersect with the appropriate automata. $\mathcal{A}_{\mathrm{ct}}$ simply verifies the conditions of Definition 17 by only having transitions between consistent pairs of frames, and making sure the root vertex is labeled with a frame whose vertex set consists exactly of $\mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$. The comparison between subgraphs of pairs of frames requires $\mathcal{A}_{\mathrm{ct}}$ to remember the previous letter, and therefore its state set is exponential in $||\mathcal{C},\mathcal{T}||$.

Verifying (★) with a Rabin tree automaton We describe an automaton $\mathcal{B}$ which runs on consistent trees over Σ_{fr}, and finds a pair of registers which violates (★). The desired $\mathcal{A}_{\mathrm{emb}}$ is the complement of $\mathcal{B}$ intersected with $\mathcal{A}_{\mathrm{ct}}$.

We now define $\mathcal{B}$ and describe its behavior. We let

$$\mathcal{B} = (Q, q_0, \longrightarrow, (\emptyset, U)), \qquad \text{with:}$$

- $Q = \{q_0, q_1, q_2\} \cup Q_p$ where Q_p is the set of *path states*
$$Q_p = \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \{f, b\} \times \{0, 1\}.$$
- We describe $\longrightarrow$ next. The initial state q_0 and the state q_2 both represent that the problematic pair is (a) in the current subtree, (b) but not in the current node. From either of them, $\mathcal{B}$ picks one child for which (a) is also true, and possibly also (b). In the latter case, it moves to q_2 for that child, while the other children go into q_1. State q_1 means that the problematic pair is not in the subtree, and once $\mathcal{B}$ visits some node in q_1, it stays in q_1 for all its descendants. Denote by $\mathbf{q}^e_i$ the n-tuple containing q_2 for entry i and q_1 for every other entry.
  - For every $i \in [n]$ and $\sigma \in \Sigma_{\mathrm{fr}}$ we have
$$q_0 \xrightarrow{\sigma} \mathbf{q}^e_i \qquad \text{and} \qquad q_2 \xrightarrow{\sigma} \mathbf{q}^e_i$$
  - For every $\sigma \in \Sigma_{\mathrm{fr}}$, we have $q_1 \xrightarrow{\sigma} (q_1, \ldots, q_1)$

  At some point, $\mathcal{B}$ moves from a node where both (a) and (b) are true (that is, q_0 or q_2) to a node where (b) no longer holds, i.e., it guesses that the problematic pair is in that node u. At this point, it guesses the problematic pair x, y and whether it is the forward path f or the backward path b which will be strict. This will be stored in the flag f or b, which once chosen cannot change during the run.
  If the guessed pair x, y has a $\leq$ relation (needed for the strict edge from $f(0)$ to $b(0)$ required by (★)), $\mathcal{B}$ transitions accordingly to a path state $(x, y, f, 0)$ or $(x, y, b, 0)$; For every $i \in [n]$, $h \in \{f, b\}$, and $- \in \{0, 1\}$, denote by $(x, y, h, -)_i$ the n-tuple containing $(x, y, h, -)$ for entry i and q_1 for every other entry.
  - For every $i \in [n]$ and $h \in \{f, b\}$, if $e_<(x^{\mathrm{bot}}, y^{\mathrm{bot}}) \in \sigma$ we have $q_0 \xrightarrow{\sigma} (x, y, h, 0)_i$ and $q_2 \xrightarrow{\sigma} (x, y, h, 0)_i$

  Then $\mathcal{B}$ attempts to expand f and b by guessing a child v and a new pair z, w with a strict edge between z and w. It moves to the appropriate path state for the child v, and to q_1 for the remaining children. When doing so, is also uses another binary flag to indicate whether $\mathcal{B}$ just witnessed a strict edge relevant to f or b (1), or not (0).
  We describe the transitions for the case where the forward path is strict; there are similar transitions for backward paths. If the guess correctly extends f and b, that is,
$$e_<(z^{\mathrm{bot}}, w^{\mathrm{bot}}) \in \sigma \qquad \text{and} \qquad e_<(y^{\mathrm{top}}, w^{\mathrm{bot}}) \notin \sigma$$
  then, for every $i \in [n]$,
  - if the current edge on the forward path is strict, that is, $e_<(x^{\mathrm{top}}, z^{\mathrm{bot}}) \in \sigma$, we have
$$(x, y, f, -) \xrightarrow{\sigma} (z, w, f, 1)_i$$
  - and if the current edge is not strict, that is, $e_=(x^{\mathrm{top}}, z^{\mathrm{bot}}) \in \sigma$, then we have
$$(x, y, f, -) \xrightarrow{\sigma} (z, w, f, 0)_i$$
- $U = \{q_1\} \cup \{(x, y, h, 1) \mid h \in \{f, b\}\}$.
  Paths looping in q_1 are successful, and to guarantee that the guessed path is strict, it must contain infinitely many strict edges (marked with flag 1).

The number of states of $\mathcal{B}$ is polynomial in $||\mathcal{C},\mathcal{T}||$, and the alphabet is exponential. As mentioned before, the automaton $\mathcal{A}_{\mathrm{emb}}$ is the complement automaton of $\mathcal{B}$ intersected with $\mathcal{A}_{\mathrm{ct}}$. It has the same alphabet, but it may have exponentially many more states (Muller and Schupp 1995).

Proposition 26. *There is a Rabin tree automaton $\mathcal{A}_{\mathrm{emb}}$ that accepts exactly the consistent trees over Σ_{fr} that satisfy (★), whose number of states is bounded by a single exponential in $||\mathcal{C},\mathcal{T}||$ and whose Ω has a constant number of pairs.*

3.5 Deciding Satisfiability

The automaton $\mathcal{A}_{\mathrm{emb}}$ provides us an effective way to decide the embeddability of a constraint graph. With this central ingredient in place, we are ready to put together an algorithm for checking the satisfiability of $\mathcal{C}$ w.r.t. $\mathcal{T}$. We do so by building an automaton $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ whose language is not empty iff $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$. In a nutshell, we obtain it by intersecting $\mathcal{A}_{\mathrm{emb}}$ and an automaton for deciding satisfiability of the abstraction to $\mathcal{ALCF}$. For the latter, we may rely on existing constructions from the literature.

Satisfiability of the abstracted $\mathcal{ALCF}$ part A well known construction of a looping automaton which accepts exactly the tree models of an $\mathcal{ALC}$ concept w.r.t. a TBox can be found in (Baader 2009). Note that, for completeness, it is important that it accepts all tree models, as opposed to e.g. accepting some canonical model which may not necessarily have an embeddable constraint graph. That construction can be easily adapted to obtain a Rabin tree automaton $\mathcal{A}_{\text{alcf}}$, which also ensures functionality of the appropriate roles. The automaton $\mathcal{A}_{\text{alcf}}$ runs on trees over the alphabet Ξ, which consists of sets of the concept names in $\mathcal{C}, \mathcal{T}$ and a single role name from $\mathcal{C}, \mathcal{T}$. The role name in each letter indicates the role with which a logical element is connected to its parent. The states of $\mathcal{A}_{\text{alcf}}$ are maximal consistent sets of the subexpressions in $\mathcal{C}, \mathcal{T}$, also known as Hintikka sets. The number of states of $\mathcal{A}_{\text{alcf}}$ and the alphabet are exponential in $||\mathcal{C}, \mathcal{T}||$, and its Ω has a constant number of pairs.

Pairing the alphabet The final automaton $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ should accept only representations of models of the abstraction of $\mathcal{C}$ and $\mathcal{T}$ whose constraint graph is embeddable. Since one check is done by $\mathcal{A}_{\text{alcf}}$ and the other by $\mathcal{A}_{\text{emb}}$, we modify both automata to use the same alphabet. We let $\mathcal{A}'_{\text{emb}}$ and $\mathcal{A}'_{\text{alcf}}$ be the modification of $\mathcal{A}_{\text{emb}}$ and $\mathcal{A}_{\text{alcf}}$ to trees over the product alphabet $\Sigma_{\text{fr}} \times \Xi$, while ignoring the irrelevant part of each letter. Clearly, the state sets of $\mathcal{A}'_{\text{emb}}$ and $\mathcal{A}'_{\text{alcf}}$ are not affected and remain exponential in $||\mathcal{C}, \mathcal{T}||$, nor are their Ω sets, which still have a constant number of pairs.

Matching the alphabets It is not enough to verify if a tree over $\Sigma_{\text{fr}} \times \Xi$ is accepted by $\mathcal{A}'_{\text{emb}}$, which ignores Ξ, and by $\mathcal{A}'_{\text{alcf}}$, which ignores Σ_{fr}: such a tree could just pair a model of the abstraction with a totally unrelated constraint graph. We need to verify that the constraint graph matches the interpretation of the abstraction. For this, we take an automaton $\mathcal{A}_{\text{m}}$ that considers both parts of the product alphabet $\Sigma_{\text{fr}} \times \Xi$ and accepts the trees where the restriction of the input to Ξ induces the constraint graph corresponding to the restriction of the input to Σ_{fr}. This is done by verifying the conditions described in Definition 10 while applying the placeholders in the Ξ part of the letter to the bot vertices in the Σ_{fr} part of the letter. Such a test can be built into the transition relation, using a constant number of states.

Putting the automata together Finally, we build $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ as the intersection of $\mathcal{A}'_{\text{emb}}$, $\mathcal{A}'_{\text{alcf}}$, and $\mathcal{A}_{\text{m}}$. Each tree it accepts represents a model of the abstraction of $\mathcal{C}$ w.r.t. $\mathcal{T}$ whose constraint graph can be embedded into $\mathbb{Z}$, yielding the desired reduction of satisfiability to automata emptiness.

Proposition 27. *There is a Rabin tree automaton $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ whose state set is bounded by a single exponential in $||\mathcal{C}, \mathcal{T}||$ and the number of pairs in its Ω is bounded by a polynomial in $||\mathcal{C}, \mathcal{T}||$, such that $L(\mathcal{A}_{\mathcal{T},\mathcal{C}}) \neq \emptyset$ iff $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$.*

Since emptiness of Rabin tree automata is decidable in time polynomial in Q and exponential in the number of pairs in Ω (Emerson and Jutla 1988), our main result follows.

Theorem 28. *Satisfiability w.r.t. general TBoxes in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ is decidable in* ExpTime.

This bound is tight: satisfiability w.r.t. general TBoxes is ExpTime-hard already for plain $\mathcal{ALC}$ (Schild 1991).

4 Beyond $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$

In this section, we discuss some variants of our construction and how our results extend to other closely related settings.

Undefined register values Classical concrete domains often allow for the predicate $\uparrow x$ which is interpreted as the register x having undefined value. In order to support this in our setting, we expand $\mathcal{Z}_c$ to $\mathcal{Z}_{c,\mathsf{und}}$ by adding a fresh element to the integers to obtain $\mathbb{Z} \cup \{\mathsf{u}\}$, and adding a unary predicate und to the predicates of $\mathcal{Z}_c$. Our approach is adapted by redefining frames as follows. The set $\mathbf{U}$ of labels also includes U_{und}, and the first condition in Definition 13 is rephrased to be:

1. There is an edge between every pair of vertices which are not labeled U_{und}.

Note that due to condition 5, also every pair of vertices labeled U_{und} is connected (with an equality edge).

Adding int or nat predicates to dense domains When operating over a dense concrete domain such as the rationals or the reals, it can be useful to have a predicate which enforces that certain registers hold integer or natural number values, for example, when modeling the number of children a person has. We show that such predicates may be added to our setting while maintaining our complexity.

The predicate $\mathsf{int}(x)$ is interpreted as $\{u \in \Delta^{\mathcal{I}} \mid \beta(u, x) \in \mathbb{Z}\}$, and similarly for $\mathsf{nat}(x)$. Note that $\mathsf{nat}(x) \equiv \mathsf{int}(x) \sqcap \exists \epsilon.[\![0 \leq S^0 x]\!]$, so we limit our treatment to int.

We describe how to add the int predicate to $\mathcal{R}_c$, which is $\mathcal{Z}_c$ with the reals as the domain, while maintaining our complexity bounds. We need to check whether the subgraph induced by the registers satisfying the int predicate is embeddable, which boils down to making sure there is no pair of registers with infinitely many int registers between them that must have different values. Therefore we adapt the automaton $\mathcal{B}$ to look for a pair of registers violating the following updated condition:

($\bigstar_{\mathsf{int}}$) There are no $(u, x), (u, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ in $\mathcal{G}_{\text{fr}}$ for which we have that: there exists an infinite $\mathbf{w} \in [n]^{\omega}$ and

1. an infinite forward path f from (u, x) along $\mathbf{w}$
2. an infinite backward path b from (u, v) along $\mathbf{w}$

such that f or b is strict and has infinitely many int labels, and such that for every $i \in \mathbb{N}$, there is a strict edge from $f(i)$ to $b(i)$.

The automaton $\mathcal{B}$ is adapted so that it guesses whether f or b is strict and has infinitely many int registers, and we add to the acceptance condition the requirement that it witnesses infinitely many int registers on the path it guessed. In the proofs we also redefine the notion of strict length of paths, counting only int registers that occur between strict edges.

4.1 The Logic $\mathcal{ALCFP}(\mathcal{D})$

The DL $\mathcal{ALCFP}(\mathcal{D})$ was introduced for a general domain $\mathcal{D}$ in (Lutz 2001), and a related DL was studied already in (Baader and Hanschke 1992). While most extensions of DLs with concrete domains allow only functional roles on the paths participating in the concepts that refer to the concrete domain, $\mathcal{ALCFP}(\mathcal{Z}_c)$ allows arbitrary roles. It is similar to our logic, but it can compare values on different paths, while $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ can only compare values on the same path.

We briefly recall the definition of $\mathcal{ALCFP}(\mathcal{Z}_c)$, and refer to (Lutz 2001) for details. Since the syntax of $\mathcal{ALCFP}$ does not allow Boolean combinations of constraints, we enrich $\mathcal{Z}_c$ to explicitly include $\neq, \leq$, and $\geq$. This provides a closer comparison between the logics, and in the case of $\mathcal{ALCF}^{\mathcal{P}}$, it is equivalent to the simpler $\mathcal{Z}_c$ considered so far.

$\mathcal{ALCFP}(\mathcal{Z}_c)$ is the augmentation of $\mathcal{ALCF}$ with[1]

- $\exists Px. = c$ and $\forall Px. = c$ where $c \in \mathbb{Z}$ and P is a sequence of role names and x a register name, and similarly for $\neq c$. The formulas apply the constraint to the x register of the last element on a P path.
- $\exists P_1x_1, P_2x_2.\theta$ and $\forall P_1x_1, P_2x_2.\theta$ where $\theta \in \{\leq, <, =, \neq, >, \geq\}$ and each P_ix_i is a sequence of role names followed by a register name. The formulas apply the constraint to the x_1 register of the last element on a P_1 path and the x_2 register of the last element on a P_2 path, where both paths start at a common element.

We now translate $\mathcal{ALCFP}(\mathcal{Z}_c)$ to $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. In most cases we use additional registers.

- $\exists Px. = c$ translates to $\exists P.[\![S^{|P|}x = c]\!]$ and $\forall Px. = c$ translates to $\forall P.[\![S^{|P|}x = c]\!]$.
- For existential concepts $\exists P_1x_1, P_2x_2.\theta$, an easy translation is possible by using two fresh register names *copy*-g_1 and *copy*-g_2, which intuitively store the values at the end of P_1 and P_2. For example, $\exists P_1x_1, P_2x_2. <$ translates to

$$\begin{aligned} C := \exists P_1.[\![S^0\textit{copy-}x_1 = S^{|P_1|}x_1]\!] \\ \sqcap \exists P_2.[\![S^0\textit{copy-}x_2 = S^{|P_2|}x_2]\!] \\ \sqcap \exists \epsilon.[\![S^0\textit{copy-}x_1 < S^0\textit{copy-}x_2]\!] \end{aligned}$$

  The translations for $\theta \in \{\leq, <, =, \neq, >, \geq\}$ are similar.
- In the cases of $\forall P_1x_1, P_2x_2.\theta$, we treat differently the paths of only functional roles, and the case where arbitrary roles may occur.
  - If all roles occurring in P_1 and P_2 are functional, then an encoding similar to $\exists P_1x_1, P_2x_2.\theta$ can be used. For example, $\forall P_1x_1, P_2x_2.<$ translates to $\neg\exists P_1.\top \sqcup \neg\exists P_2.\top \sqcup C$, where C is as above.
  - If non-functional roles occur in P_1 and P_2, then we may need to compare numbers on several paths, and we may need more sophisticated tricks. For $\theta \in \{\leq, <, =, \neq, >, \geq\}$, this is still possible using just a few registers.

[1] $\mathcal{ALCFP}(\mathcal{Z}_c)$ supports $x\uparrow$, which we can simulate as above.

For example, we can translate $\forall P_1x_1, P_2x_2. <$ as

$$\begin{aligned} \neg\exists P_1.\top \sqcup \big(\exists P_1.[\![S^{|P_1|}x_1 = S^0\textit{copy-}x_1]\!] \\ \sqcap \forall P_1.[\![S^{|P_1|}x_1 \leq S^0\textit{copy-}x_1]\!] \\ \sqcap \forall P_2.[\![S^{|P_2|}x_2 > S^0\textit{copy-}x_1]\!]\big) \end{aligned}$$

We are essentially ensuring, via *copy*-x_1, that the largest value of x_1 seen with a P_1 path is smaller than every value of x_2 seen with a P_2 path.

If θ is $\neq$, our translation requires exponentially many new register names. Given $\mathcal{C}, \mathcal{T}$ we can ascertain a degree k of some tree model (if any model exists). In this model there would be at most $|P_1|^k$ different values to consider for the satisfaction of the constraint. Slightly abusing notation, we express that the x_1 registers at the end of P_1 paths contain values from a finite set which appears in the fresh register names of the common ancestor, and that this set does not intersect with the set of values of the x_2 registers at the end of P_2 paths:

$$\begin{aligned} \exists\epsilon.[\![S^0x_1 \neq \cdots \neq S^0x_{|P_1|^k}]\!] \\ \sqcap \forall P_1.[\![S^{|P_1|}x_1 = x_1 \vee \cdots \vee S^{|P_1|}x_1 = x_{|P_1|^k}]\!] \\ \sqcap \forall P_2.[\![S^{|P_2|}x_2 = x_1 \vee \cdots \vee S^{|P_2|}x_2 \neq x_{|P_1|^k}]\!] \end{aligned}$$

This translation allows us to give an upper bound on the complexity of reasoning w.r.t. general TBoxes in the DL $\mathcal{ALCFP}(\mathcal{Z}_c)$, which to the best of our knowledge, had never been provided before. Our upper bounds also apply if we replace the integers by the real numbers, with or without int and nat predicates in the concrete domain.

Theorem 29. *Satisfiability w.r.t. general TBoxes in $\mathcal{ALCFP}(\mathcal{Z}_c)$ is decidable in 2ExpTime, and it is ExpTime-complete if there is a constant bound on the length of any path P_1 that contains non-functional roles and occurs in a concept of the form $\forall P_1x_1, P_2x_2.\neq$.*

5 Conclusions

We have closed a long-standing open question in the literature of DLs with concrete domains: reasoning with general TBoxes in $\mathcal{ALC}$ extended with the non-dense domain $\mathcal{Z}_c$ is ExpTime-complete, and hence not harder than in plain $\mathcal{ALC}$, even if arbitrary paths of (not necessarily functional) roles are allowed to refer to the concrete domain. This positive result extends to other domains that have been advocated for in the literature, for example, comparisons over the real or rational numbers but with the int and nat predicates. Our technique builds on ideas used for constraint LTL in (Demri and D'Souza 2007), and our condition ($\bigstar$) is very similar to the condition used in that paper. Lifting the results from linear structures, as in LTL, to the tree-shaped ones needed in $\mathcal{ALC}$ is not trivial. It remains an open question whether our technical results can be transferred to fragments of constraint CTL* to obtain new complexity bounds. Natural next steps are exploring other DLs, for example $\mathcal{SHIQ}^{\mathcal{P}}(\mathcal{Z})$, and considering ABoxes and instance queries.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF) projects P30360, P30873, and W1255.

References

Artale, A.; Ryzhikov, V.; and Kontchakov, R. 2012. DL-Lite with attributes and datatypes. In *Proc. of ECAI 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 61–66. IOS Press.

Baader, F., and Hanschke, P. 1991. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI 1991*, 452–457. Morgan Kaufmann.

Baader, F., and Hanschke, P. 1992. Extensions of concept languages for a mechanical engineering application. In *GWAI*, volume 671 of *LNCS*, 132–143. Springer.

Baader, F.; Borgwardt, S.; and Lippmann, M. 2017. Query rewriting for DL-Lite with n-ary concrete domains. In *Proc. of IJCAI 2017*, 786–792. ijcai.org.

Baader, F. 2009. Description logics. In *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School*, volume 5689 of *LNCS*, 1–39. Springer.

Bojańczyk, M., and Toruńczyk, S. 2012. Weak MSO+U over infinite trees. In *Proc. of STACS 2012*, volume 14 of *LIPIcs*, 648–660. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bozzelli, L., and Gascon, R. 2006. Branching-time temporal logic extended with qualitative Presburger constraints. In *Proc. of LPAR 2006*, volume 4246 of *LNCS*, 197–211. Springer.

Carapelle, C., and Turhan, A. 2016. Description logics reasoning w.r.t. general TBoxes is decidable for concrete domains with the EHD-property. In *Proc. of ECAI 2016*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 1440–1448. IOS Press.

Carapelle, C.; Kartzow, A.; and Lohrey, M. 2016. Satisfiability of ECTL* with constraints. *J. Comput. Syst. Sci.* 82(5):826–855.

Demri, S., and D'Souza, D. 2007. An automata-theoretic approach to constraint LTL. *Inf. Comput.* 205(3):380–415.

Diestel, R. 2012. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer.

Emerson, E. A., and Jutla, C. S. 1988. The complexity of tree automata and logics of programs (extended abstract). In *Proc. of FOCS 1988*, 328–337. IEEE Computer Society.

Haarslev, V.; Möller, R.; and Wessel, M. 2001. The description logic alcnh$_{R+}$ extended with concrete domains: A practically motivated approach. In *Proc. of IJCAR 2001*, volume 2083 of *LNCS*, 29–44. Springer.

Labai, N.; Ortiz, M.; and Šimkus, M. 2020. An ExpTime upper bound for ALC with integers (extended version). *CoRR* abs/2006.02078.

Lutz, C., and Milicic, M. 2007. A tableau algorithm for description logics with concrete domains and general TBoxes. *J. Autom. Reasoning* 38(1-3):227–259.

Lutz, C. 2001. NEXPTIME-complete description logics with concrete domains. In *Proc. of IJCAR 2001*, volume 2083 of *LNCS*, 45–60. Springer.

Lutz, C. 2002a. Adding numbers to the SHIQ description logic: First results. In *Proc. of KR 2002*, 191–202.

Lutz, C. 2002b. Description logics with concrete domains-a survey. In *Proc. of Advances in Modal Logic 4*, 265–296. King's College Publications.

Lutz, C. 2002c. Pspace reasoning with the description logic ALCF(D). *Logic Journal of the IGPL* 10(5):535–568.

Lutz, C. 2004a. Combining interval-based temporal reasoning with general TBoxes. *Artif. Intell.* 152(2):235–274.

Lutz, C. 2004b. NEXPTIME-complete description logics with concrete domains. *ACM Trans. Comput. Logic* 5(4):669–705.

Muller, D. E., and Schupp, P. E. 1995. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.* 141(1&2):69–107.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *Journal on Data Semantics* 10:133–173.

Rabin, M. O. 1972. *Automata on Infinite Objects and Church's Problem*. Boston, MA, USA: American Mathematical Society.

Savkovic, O., and Calvanese, D. 2012. Introducing datatypes in DL-Lite. In *Proc. of ECAI 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 720–725. IOS Press.

Schild, K. 1991. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI 1991*, 466–471. Morgan Kaufmann.