# Unification in a Description Logic with Transitive Closure of Roles

Franz Baader[1] and Ralf Küsters[2]

[1] RWTH Aachen, Germany, baader@informatik.rwth-aachen.de
[2] CAU Kiel, Germany, kuesters@ti.informatik.uni-kiel.de

**Abstract.** Unification of concept descriptions was introduced by Baader and Narendran as a tool for detecting redundancies in knowledge bases. It was shown that unification in the small description logic $\mathcal{FL}_0$, which allows for conjunction, value restriction, and the top concept only, is already ExpTime-complete. The present paper shows that the complexity does not increase if one additionally allows for composition, union, and transitive closure of roles. It also shows that matching (which is polynomial in $\mathcal{FL}_0$) is PSpace-complete in the extended description logic. These results are proved via a reduction to linear equations over regular languages, which are then solved using automata. The obtained results are also of interest in formal language theory.

## 1 Introduction

Knowledge representation languages based on Description Logics (DL) can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way [10, 4]. With the help of these languages, the important notions of the domain can be described by *concept descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the DL language. Atomic concepts and concept descriptions represent sets of individuals, whereas roles and role descriptions represent binary relations between individuals.

Unification of concept descriptions was introduced by Baader and Narendran [8] as a new inference service for detecting and avoiding redundancies in DL knowledge bases. Unification considers concept patterns, i.e., concept descriptions with variables, and tries to make these descriptions equivalent by replacing the variables by appropriate concept descriptions. The technical results in [8] were concerned with unification in the small DL $\mathcal{FL}_0$, which allows for conjunction of concepts ($C \sqcap D$), value restriction ($\forall R.C$), and the top concept ($\top$). It is shown that unification of $\mathcal{FL}_0$-concept descriptions is equivalent to solving systems of linear equations over finite languages, and that this problem is ExpTime-complete.

In the present paper, we study unification in $\mathcal{FL}_{reg}$, the DL that extends $\mathcal{FL}_0$ by the role constructors identity role ($\varepsilon$), empty role ($\emptyset$), union ($R \cup S$), composition ($R \circ S$), and reflexive-transitive closure ($R^*$).[1] Unification of $\mathcal{FL}_{reg}$-concept descriptions is again equivalent to solving systems of linear language equations, but the finite languages are now replaced by regular languages. The first contribution of the present paper is to show that deciding the solvability of such equations is, as in the finite case, ExpTime-complete. At first sight one might think that it is sufficient to show that the problem is in ExpTime, since ExpTime-hardness already holds for the "simpler" case of unification in $\mathcal{FL}_0$. However, unification in $\mathcal{FL}_{reg}$ is not a priori at least as hard as unification in $\mathcal{FL}_0$ since the set of potential solutions increases. Thus, an $\mathcal{FL}_0$-unification problem (which can also be viewed as an $\mathcal{FL}_{reg}$-unification problem) may be solvable in $\mathcal{FL}_{reg}$, but not in $\mathcal{FL}_0$. (We will see such an example later on.)

Our complexity results are by reduction to/from decision problems for tree-automata. Whereas for equations over finite languages automata on finite trees could be used, we now consider automata working on infinite trees. As a by-product of the reduction to tree automata, we also show that, if a system of linear equations has some (possibly irregular) solution, then it also has a regular one. That is, restricting solutions to substitutions that map variables to regular languages does not make a difference in terms of the solvability of an equation.

Equations over regular languages have already been considered by Leiss [12, 11]. However, he does not provide any decidability or complexity results for the case we are interested in. Closely related to the problem of solving linear language equations is the problem of solving set constraints [1], i.e., relations between sets of terms. Set constraints are usually more general than the kind of equations we are dealing with here. The case we consider here corresponds most closely to positive set constraints for terms over unary and nullary function symbols where only union of sets is allowed. For solvability of positive set constraints over (at least two) unary and (at least one) nullary function symbols, ExpTime-completeness is shown in [1]. However, this result does not directly imply the corresponding result for our case. On the one hand, for set constraints one considers equations with finite languages as coefficients, whereas we allow for regular languages as coefficients. It is, however, easy to see that regular coefficients can be expressed using set constraints. On the other hand, for set constraints one allows for arbitrary (possibly) infinite solutions, whereas we restrict the attention to regular solutions. Using the (new) result that the restriction to regular sets does not change the solvability of an equation, our exponential upper bound also follows from the complexity result in [1]. The hardness result in [1] does not directly carry over since even positive set constraints allow for more complex types of equations than the linear ones considered here.

Matching is a special case of unification where only one of the patterns contains variables. In [8] it was shown that matching in $\mathcal{FL}_0$ is polynomial, and in [7] this result was extended to the more expressive DL $\mathcal{ALN}$. We will show that matching in $\mathcal{FL}_{reg}$ is PSpace-complete.

In case a unification/matching problem is solvable, one is usually interested in obtaining an actual solution. In the context of matching in description logics,

[1] Transitive closure then corresponds to the expression $R \circ R^*$.

| Syntax | Semantics | $\mathcal{FL}_0$ | $\mathcal{FL}_{reg}$ |
|---|---|---|---|
| $\top$ | $\Delta^I$ | x | x |
| $C \sqcap D$ | $C^I \cap D^I$ | x | x |
| $\forall R.C$ | $\{x \in \Delta^I \mid \forall y : (x,y) \in R^I \to y \in C^I\}$ | x | x |
| $\varepsilon$ | $\{(x,x) \mid x \in \Delta^I\}$ | | x |
| $\emptyset$ | $\emptyset$ | | x |
| $R \circ S$ | $\{(x,z) \mid \exists y : (x,y) \in R^I \wedge (y,z) \in S^I\}$ | | x |
| $R^*$ | $\bigcup_{n \geq 0} (R^I)^n$ | | x |

**Table 1.** Syntax and semantics of concept descriptions.

it has been argued [9, 5] that not all solutions of a matching problem are of interest to a user. Therefore, one must look for solutions with certain desired properties; for instance, least solutions where all variables are substituted by concept descriptions that are as specific as possible turned out to be appropriate in some contexts [9, 13]. For matching in $\mathcal{FL}_0$ and $\mathcal{FL}_{reg}$, solvable problems always have a least solution. For unification, we will show that this is only true for $\mathcal{FL}_{reg}$.

## 2 Unification in $\mathcal{FL}_{reg}$

Let us first introduce $\mathcal{FL}_0$- and $\mathcal{FL}_{reg}$-concept descriptions. Starting from the finite and disjoint sets $N_C$ of concept names and $N_R$ of role names, $\mathcal{FL}_0$-concept descriptions are built using the concept constructors conjunction ($C \sqcap D$), value restriction ($\forall r.C$), and the top concept ($\top$). $\mathcal{FL}_{reg}$ extends $\mathcal{FL}_0$ by additionally allowing for the role constructors identity role ($\varepsilon$), empty role ($\emptyset$), union ($R \cup S$), composition ($R \circ S$), and reflexive-transitive closure ($R^*$). As an example, consider the $\mathcal{FL}_{reg}$-concept description Woman $\sqcap\,\forall$child$^*$.Woman, which represents the set of all women with only female offspring.

Role names will be denoted by lower case letters ($r, s, \ldots \in N_R$), and complex roles by upper case letters ($R, S, T \ldots$). Note that a complex role can be viewed as a regular expression over $N_R$ where $\varepsilon$ is taken as the empty word, role names as elements of the alphabet, the empty role as the empty language, union as union of languages, composition as concatenation, and reflexive-transitive closure as Kleene star. Therefore, we sometimes view a complex role $R$ as a regular expression. In the following, we will abuse notation by identifying regular expressions with the languages they describe. In particular, if $R$ and $R'$ are regular expressions, then $R = R'$ will mean that the corresponding languages are equal.

As usual, the semantics of concept and role descriptions is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^I, \cdot^I)$. The domain $\Delta^I$ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^I$ maps each concept name $A \in N_C$ to a set $A^I \subseteq \Delta^I$ and each role name $r \in N_R$ to a binary relation $r^I \subseteq \Delta^I \times \Delta^I$. The extension of $\cdot^I$ to arbitrary concept and role descriptions is defined inductively, as shown in the second column of Table 1. The interested reader may note that $\mathcal{FL}_{reg}$-concept descriptions can also be viewed as concepts defined by cyclic $\mathcal{FL}_0$-TBoxes interpreted with the greatest fixed-point semantics [2]. The concept description $D$ *subsumes* the description $C$ ($C \sqsubseteq D$) iff $C^I \subseteq D^I$ for all interpretations $\mathcal{I}$. Two concept descriptions $C, D$ are *equivalent* ($C \equiv D$) iff they subsume each other.

In order to define unification of concept descriptions, we first have to introduce the notions concept patterns and substitutions operating on concept patters. To this purpose, we need a set of *concept variables* $N_X$ (disjoint from $N_C \cup N_R$). *$\mathcal{FL}_{reg}$-concept patterns* are $\mathcal{FL}_{reg}$-concept descriptions defined over the set $N_C \cup N_X$ of concept names and the set $N_R$ of role names. For example, given $A \in N_C$, $X \in N_X$, and $r \in N_R$, $\forall r.A \sqcap \forall r^*.X$ is an $\mathcal{FL}_{reg}$-concept pattern.

A *substitution* $\sigma$ is a mapping from $N_X$ into the set of all $\mathcal{FL}_{reg}$-concept descriptions. This mapping is extended from variables to concept patterns in the obvious way, i.e.,

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

**Definition 1.** *An $\mathcal{FL}_{reg}$-*unification problem *is of the form $C \equiv^? D$, where $C$, $D$ are $\mathcal{FL}_{reg}$-concept patterns. The substitution $\sigma$ is a* unifier *of this problem iff $\sigma(C) \equiv \sigma(D)$. In this case, the unification problem is* solvable*, and $C$ and $D$ are called* unifiable.

For example, the substitution $\sigma = \{X \mapsto \forall r \circ r^*.A,\ Y \mapsto \forall r.A\}$ is a unifier of the unification problem

$$\forall s.\forall r.A \sqcap \forall r.A \sqcap \forall r.X \equiv^? X \sqcap \forall s.Y. \tag{1}$$

Note that this problem can also be viewed as an $\mathcal{FL}_0$-unification problem. However, in this case it does not have a solution since there are no $\mathcal{FL}_0$-concept descriptions that, when substituted for $X$ and $Y$, make the two concept patterns equivalent.

For readers interested in unification theory, let us point out that (just as for $\mathcal{FL}_0$ [8]), unification in $\mathcal{FL}_{reg}$ can be viewed as unification modulo an appropriate equational theory, and that (like the theory corresponding to $\mathcal{FL}_0$) this theory is of unification type zero.

## 3 Reduction to regular language equations

We now show how unification in $\mathcal{FL}_{reg}$ can be reduced to solving linear equations over regular languages built using the alphabet $N_R$ of role names.

The equations we are interested in are built as follows. Let $\Sigma$ be a finite alphabet. For languages $L, M \subseteq \Sigma^*$, their concatenation is defined by $LM := \{vw \mid v \in L, w \in M\}$. Let $X_1, \ldots, X_n$ be variables. Given regular languages $S_0, S_1, \ldots, S_n, T_0, T_1, \ldots, T_n$[2] over $N_R$, a *linear equation over regular languages* is of the form

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

[2] We assume that these languages are given by regular expressions or nondeterministic finite automata.

A *(regular, finite) solution* $\theta$ of this equation is a substitution assigning to each variable a (regular, finite) language over $\Sigma$ such that the equation holds. We are particularly interested in *regular* solutions since these can be turned into $\mathcal{FL}_{reg}$-concept descriptions.

A *system of regular language equations* is a finite set of regular language equations. A substitution $\theta$ solves such a system if it solves every equation in it simultaneously. A system of equations can easily (in linear time) be turned into a single equation with the same set of solutions by concatenating all constant languages in an equation with a role $r$ (a new role for every equation), i.e., the languages $S_i$ and $T_i$ are replaced by $\{r\}S_i$ and $\{r\}T_i$. Then the different equations can be put together into a single equation without causing any interference (see [8] for details). Hence, for our complexity analysis we can focus on single equations.

To establish the reduction from unification in $\mathcal{FL}_{reg}$ to solvability of linear equations over regular languages, $\mathcal{FL}_{reg}$-concept patterns are written in the following normal form:

$$\bigsqcap_{A \in N_C} \forall R_A.A \sqcap \bigsqcap_{X \in N_X} \forall R_X.X,$$

where $R_A$ and $R_X$ are regular expressions over $N_R$. Every concept pattern can (in polynomial time) be turned into such a normal form by exhaustively applying the following equivalence preserving rule: $\forall R.C \sqcap \forall R'.C \longrightarrow \forall(R \cup R').C$, where $R, R'$ are regular expressions over $N_R$ and $C$ is some $\mathcal{FL}_{reg}$-concept pattern. Correctness of our reduction from unification to solvability of linear equations depends on the following (easily provable [2,3]) characterization of equivalence:

**Lemma 1.** *Let $C, D$ be $\mathcal{FL}_{reg}$-concept descriptions such that*

$$C \equiv \bigsqcap_{A \in N_C} \forall S_A.A \quad \textit{and} \quad D \equiv \bigsqcap_{A \in N_C} \forall T_A.A.$$

*Then $C \equiv D$ iff $S_A = T_A$ for all $A \in N_C$.*

As an easy consequence, we obtain the following theorem, which shows that unification in $\mathcal{FL}_{reg}$ is equivalent via linear time reductions to solving regular language equations.

**Theorem 1.** *Let $C, D$ be $\mathcal{FL}_{reg}$-concept patterns such that*

$$C \equiv \bigsqcap_{A \in N_C} \forall S_A.A \sqcap \bigsqcap_{X \in N_X} \forall S_X.X \quad \textit{and} \quad D \equiv \bigsqcap_{A \in N_C} \forall T_A.A \sqcap \bigsqcap_{X \in N_X} \forall T_X.X.$$

*Then $C, D$ are unifiable iff, for all $A \in N_C$, the regular language equation $E_{C,D}(A)$ below has a solution:*

$$S_A \cup \bigcup_{X \in N_X} S_X X_A = T_A \cup \bigcup_{X \in N_X} T_X X_A$$

Note that the language equations in this system do not share variables, and thus they can be solved separately. In the equation $E_{C,D}(A)$, the variable $X_A$ is a new copy of $X \in N_X$. Different equations have different copies.

Continuing our example, from the unification problem (1) we obtain the following language equation (assuming $N_C = \{A\}$):

$$\{r, sr\} \cup \{r\}X_A = \{\varepsilon\}X_A \cup \{s\}Y_A$$

A solution of this equation is $X_A = rr^*$ and $Y_A = r$, which corresponds to the solution $\sigma$ of (1).

## 4 The decision problem

The first theorem of this paper gives the exact complexity of solving systems of linear equations over regular languages.

**Theorem 2.** *Deciding (regular) solvability of (systems of) equations of the form (2) is an ExpTime-complete problem.*

As an immediate consequence, unification in $\mathcal{FL}_{reg}$ is ExpTime-complete as well.

**The upper complexity bound.** To prove that the problem can be solved in ExpTime, it suffices to concentrate on a single equation. Moreover, instead of (2) we consider equations where the variables occur in front of the coefficients. Such an equation can easily be obtained from (2) by considering the mirror images (or reverse) of the coefficient languages. That is, we go from a language $L \subseteq N_R^*$ to its mirror image $L^{mi} := \{r_m \cdots r_1 \mid r_1 \cdots r_m \in L\}$. The mirror equation of (2) is of the form

$$S_0^{mi} \cup X_1 S_1^{mi} \cup \cdots \cup X_n S_n^{mi} = T_0^{mi} \cup X_1 T_1^{mi} \cup \cdots \cup X_n T_n^{mi}. \tag{3}$$

Obviously, the mirror images of solutions of (3) are exactly the solutions of (2).

To test (3) for solvability, we build a looping tree-automaton $\mathcal{B}$, i.e., a Büchi tree-automaton where all states are final. Let us briefly introduce infinite trees and looping tree-automata (see [15] for details). Let $\Sigma$ be a finite alphabet and, w.l.o.g., $N_R = \{1, \ldots, k\}$. A *$\Sigma$-labeled $k$-ary infinite tree $t$* is a mapping from $N_R^*$ into $\Sigma$. (In particular, the nodes of $t$ can be viewed as words over $N_R$.) In case $\Sigma$ is a singleton, $t$ is called *unlabeled.* A *looping tree-automaton $\mathcal{A}$* is a tuple $(Q, \Sigma, I, \Delta)$ where $Q$ is the finite set of states of $\mathcal{A}$, $\Sigma$ is a finite alphabet, $I \subseteq Q$ is the set of initial states, and $\Delta \subseteq Q \times \Sigma \times Q^k$ is the transition relation. (Note that we do not define final states. Also, we will omit $\Sigma$ in case it is a singleton.) A *run $r$* of $\mathcal{A}$ on the tree $t$ is a $Q$-labeled $k$-ary tree such that $(r(u), t(u), r(u1), \ldots, r(uk)) \in \Delta$. It is called successful if $r(\varepsilon) \in I$. The tree language accepted by $\mathcal{A}$ is $L(\mathcal{A}) := \{t \mid \text{there exists a successful run of } \mathcal{A} \text{ on } t\}$.

Our looping tree-automaton $\mathcal{B}$ will work on the (unique) unlabeled $k$-ary infinite tree $t$ (thus $L(\mathcal{B})$ will be the empty set or $\{t\}$). The idea underlying the construction is as follows. A $Q$-labeled $k$-ary infinite tree $r$ can be used to describe sets of words by taking those words $u$ for which the label $r(u)$ satisfies a certain property. In principle, a run of $\mathcal{B}$ on $t$ represents i) a set of words over $N_R$ obtained by instantiating the equation with one of its solutions (called ***solution sets*** in the following), and ii) the solution itself, i.e., the languages substituted for the variables. To achieve this, while working its way down $t$, in every step $\mathcal{B}$ guesses whether the current node (or more precisely the word it represents) a) belongs to the solution set, and b) to the language substituted for $X_i$ $(i = 1, \ldots, n)$. In addition, $\mathcal{B}$ checks whether the guesses made actually yield a solution.

Formally, $\mathcal{B} = (Q, I, \Delta)$ is defined as follows. (We provide a more detailed explanation after the definition.) Let $\mathcal{A}_{S,i} = (Q_{S,i}, N_R, q_{S,i}, \Delta_{S,i}, F_{S,i})$ and $\mathcal{A}_{T,i} = (Q_{T,i}, N_R, q_{T,i}, \Delta_{T,i}, F_{T,i})$ be (nondeterministic) finite automata accepting the languages $S_i^{mi}$ and $T_i^{mi}$ $(i = 0, \ldots, n)$, respectively. We assume (w.l.o.g.) that the set of states of these automata are pairwise disjoint. Let $N := \{0, 1, \ldots, n\}$, $Q_S$ $(Q_T)$ be the union of the sets $Q_{S,i}$ $(Q_{T,i})$, $i = 0, \ldots, n$, and $F_S$ $(F_T)$ be the union of the sets $F_{S,i}$ $(F_{T,i})$.

1. $Q := 2^N \times 2^{Q_S} \times 2^{Q_T}$;
2. $I := \{(G, L, R) \mid G \subseteq N,\ L = \{q_{S,0}\} \cup \{q_{S,i} \mid i \in G\}$, and $R := \{q_{T,0}\} \cup \{q_{T,i} \mid i \in G\}$;
3. $\Delta$ consists of all tuples $((G_0, L_0, R_0), (G_1, L_1, R_1), \ldots, (G_k, L_k, R_k)) \in Q \times Q^k$ such that
   (a) $0 \in G_0$ iff $L_0 \cap F_S \neq \emptyset$ iff $R_0 \cap F_T \neq \emptyset$;
   (b) for all $i = 1, \ldots, k$,
   $L_i := \text{suc}(L_0, i) \cup \{q_{S,j} \mid j \in G_i\}$ and $R_i := \text{suc}(R_0, i) \cup \{q_{T,j} \mid j \in G_i\}$, where $\text{suc}(L_0, i) := \{q \mid$ there exists $q'$ and $j$ with $q' \in L_0 \cap Q_{S,j}$ and $(q', i, q) \in \Delta_{S,j}\}$ and $\text{suc}(R_0, i)$ is defined analogously.

Intuitively, $\mathcal{B}$ uses the first component of its states to guess whether a node (the word it represents) belongs to the solution set and/or to one of the variables $X_i$. That is, given a state $(G, L, R)$, $0 \in G$ means that the current node belongs to the solution set and $i \in G$ means that the node belongs to $X_i$ (more accurately, to the language substituted for $X_i$). The other two components are used to do the book-keeping necessary to check whether the guesses actually yield a solution. To understand their rôle, assume that $r$ is a run of $\mathcal{B}$ on $t$. W.l.o.g. we consider the second component. If $r(u) = (G, L, R)$ and $j \in G$, for some $j \neq 0$, then $u$ belongs to $X_j$, and thus $uv$ belongs to $X_j S_j^{mi}$ for all $v \in S_j^{mi}$. Consequently, if $r(uv) = (G', L', R')$, then we must have $0 \in G'$. To enforce this, $q_{S,j}$ (the initial state of the automaton $\mathcal{A}_{S,j}$ accepting $S_j^{mi}$) is added to $L$. The transitions of $\mathcal{B}$ then simulate the transitions of $\mathcal{A}_{S,j}$ in the second component. Thus, in $r(uv)$ the set $L'$ contains a final state of $\mathcal{A}_{S,j}$, and now (3a) implies that $0 \in G'$ must hold. Conversely, if $0 \in G'$, then $L'$ must contain a final state of some of the automata $\mathcal{A}_{S,i}$ $(i = 0, \ldots, n)$.

Given a successful run $r$ of $\mathcal{B}$, it is now easy to prove that the substitution $\theta_r$:

$$\theta_r(X_i) := \{u \mid r(u) = (G, L, R) \text{ and } i \in G\}$$

is a solution of (3). Conversely, it is not hard to show that a given solution of (3) induces a successful run of $\mathcal{B}$.

**Lemma 2.** *There is a one-to-one correspondence between solutions of (3) and successful runs of $\mathcal{B}$.*

The lemma implies that equation (3) has a solution iff $\mathcal{B}$ has a successful run (i.e., $L(\mathcal{B}) \neq \emptyset$). The size of the set of states of $\mathcal{B}$ is exponential in the size of equation (3), where the size of the regular sets $S_i^{mi}$ and $T_i^{mi}$ are measured by the size of nondeterministic finite automata accepting these sets. Since the emptiness problem for Büchi tree-automata (and thus looping tree-automata) can be solved in polynomial time in the size of the automaton [15] (and actually in linear time for looping automata), this yields the desired exponential time algorithm for deciding the solvability of equation (3). However, the existence of a solution does not a priori imply that there is also a regular one. Thus, we must still show that regular solvability can also be decided in ExpTime.

It is well-known [15] that a Büchi-automaton has a successful run iff it has a regular (or rational) run. It is easy to show that the solution corresponding to a regular run is a regular solution.

**Proposition 1.** *If (3) has a solution, then it also has a regular one.*

This proposition also follows from our results in Section 5.

**The lower complexity bound.** The hardness result can be shown similarly to the proof by Baader and Narendran [8] for systems of equations over finite languages. In their proof, the intersection emptiness problem for deterministic root-to-frontier automata on finite trees, which has been shown to be ExpTime-complete by Seidl [14], is reduced to the solvability of systems of equations over finite languages. The *intersection emptiness problem* is defined as follows: given a sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of deterministic root-to-frontier automata over the same ranked alphabet $\Sigma$, decide whether there exists a tree $t$ accepted by $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Instead of deterministic root-to-frontier automata we will here use deterministic looping tree-automata: a looping tree-automaton is *deterministic* if it has one initial state and, for every state $q$ and symbol $f$, there exists at most one transition of the form $(q, f, \ldots)$. We will show that Seidl's result easily carries over to these automata. However, we need to consider looping tree-automata over infinite trees labeled by elements of a *ranked* alphabet. That is, the number of successors of a node varies depending on the arity of the label attached to the node. Modifying the definition of looping tree-automata to work on these trees is straightforward.

**Proposition 2.** *The intersection emptiness problem for looping tree-automata over a ranked alphabet is ExpTime-hard.*

This can be shown by reducing the intersection emptiness problem for root-to-frontier automata to the intersection emptiness problem for looping tree-automata. The main idea is to turn every finite tree $t$ into an infinite tree $\widehat{t}$ by adding a new symbol # (say of rank 1) to the alphabet, and extending the finite tree at every leaf by attaching the infinite tree labeled by # only. A given root-to-frontier automaton $\mathcal{A}$ can then easily be modified to a looping tree-automaton

$\mathcal{B}$ such that every successful run of $\mathcal{A}$ on $t$ corresponds to a successful run of $\mathcal{B}$ on $\widehat{t}$ and vice versa.

It remains to show how the intersection emptiness problem for looping tree-automata can be reduced to the solvability of systems of linear equations over regular languages. In the following, let $\Sigma$ be a ranked alphabet. Seidl's result implies that it suffices to restrict the attention to symbols of rank 1 and 2.

We represent an infinite tree $t$ over the ranked alphabet $\Sigma$ by an infinite set $S(t)$ of words over $\Sigma \cup \{1,2\}$. This set contains one element for every node of the tree. Given a node $u$, the corresponding word describes the path from this node to the root of the tree by listing the labels of the nodes $v$ on this path together with the information whether $v$ is the first or second successor of its parent node. To be more precise, if $t = f(t_1, t_2)$ is the tree whose root is labeled with $f$ and has the two successor trees $t_1$ and $t_2$, then $S(t) := \{\varepsilon\} \cup \{u1f \mid u \in S(t_1)\} \cup \{u2f \mid u \in S(t_2)\}$. Accordingly, if $t = g(t')$, then $S(t) := \{\varepsilon\} \cup \{u1g \mid u \in S(t')\}$. For example, if $f$ is binary, $g$ is unary, and $t$ is the infinite tree labeled with $g$ only, then $S(f(t,t)) = \{\varepsilon\} \cup (1g)^*1f \cup (1g)^*2f$. Given a node $u$ in $t$ we denote the word representing $u$ in $S(t)$ by $w^t(u)$. In the example, $w^{f(t,t)}(211) = 1g1g2f$.

Now, let $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ be a deterministic looping tree-automaton over the ranked alphabet $\Sigma$. We construct the following linear equation, where the variables $X_{(q,g)}$ range over (possibly infinite) sets of words over $\Sigma' := \Sigma \cup Q \cup \{1,2\}$:

$$\bigcup_{(q,g)\in Suc} \{q\}X_{(q,g)} = \{q_0\} \cup \bigcup_{(q,g,q_1,\ldots,q_k)\in\Delta} \{q_1 1g, \ldots, q_k kg\}X_{(q,g)}, \tag{4}$$

where $Suc$ denotes the set of tuples $(q,g)$ for which there exist $q_1, \ldots, q_k$ with $(q, g, q_1, \ldots, q_k) \in \Delta$, and $k$ denotes the rank of $g$.

We want to show that solutions of (4) induce accepting runs of $\mathcal{A}$ and vice versa. Assuming that (4) has the solution $\theta$, let us try to construct a tree $t$ and a successful run of $\mathcal{A}$ on $t$. Since $q_0$ occurs on the right-hand side of (4), it must also occur on the left-hand side. Thus, there must exist a symbol $g$ such that $(q_0, g) \in Suc$ and $\varepsilon \in \theta(X_{(q_0,g)})$. Intuitively, this corresponds to setting $t(\varepsilon) := g$ and $r(\varepsilon) := q_0$. Now, since $\varepsilon \in \theta(X_{(q_0,g)})$, additional words occur on the right-hand side of (4). Indeed, since $(q_0, g) \in Suc$, there exist $q_1, \ldots, q_k$ with $(q_0, g, q_1, \ldots, q_k) \in \Delta$. Thus, the words $q_1 1g, \ldots, q_k kg$ occur on the right-hand side. This corresponds to setting $r(1) := q_1, \ldots, r(k) := q_k$. Let us look at $q_1 1g$. This word must also occur on the left-hand side of (4). Thus, there must exist a symbol $f$ with $(q_1, f) \in Suc$ and $1g \in \theta(X_{(q_1,f)})$. This corresponds to setting $t(1) := f$. Now, since $1g \in \theta(X_{(q_1,f)})$, additional words occur on the right-hand side of (4), and one continues just as in the case $\varepsilon \in \theta(X_{(q_0,g)})$. This illustrates that, if (4) is solvable, then one can construct a tree $t$ and an accepting run $r$ of $\mathcal{A}$ on $t$. Moreover, it follows that $S(t) \subseteq V_\theta := \bigcup_{(q,g)\in Suc} \theta(X_{(q,g)})$.

Conversely, if $t \in L(\mathcal{A})$ and $r$ is the (unique) accepting run of $\mathcal{A}$ on $t$, then we can use $r$ to construct a solution $\theta$ of (4) such that $S(t) = V_\theta$:

$$\theta(X_{(q,g)}) := \{w^t(u) \mid t(u) = g \wedge r(u) = q\}.$$

**Lemma 3.** *If $\theta$ solves (4), then there exists $t \in L(\mathcal{A})$ with $S(t) \subseteq V_\theta$. Conversely, if $t \in L(\mathcal{A})$, then there exists a solution $\theta$ of (4) with $S(t) = V_\theta$.*

The inclusion in the first part of the lemma may be strict. In fact, by the second part, every tree in $L(\mathcal{A})$ yields a solution of (4). Since the solutions of such linear equations are closed under (argument-wise) union, there are solutions $\theta$ representing more than one accepted tree. Because of this fact, our reduction will depend on the following lemma.

**Lemma 4.** *Let $\theta$ be a solution of (4) and $t$ a tree. If $S(t) \subseteq V_\theta$, then $t \in L(\mathcal{A})$.*

In contrast to the previous lemma, Lemma 4 holds only because the automaton $\mathcal{A}$ is assumed to be deterministic (see [6] for a proof).

We are now ready to reduce the intersection emptiness problem to solving a system of linear equations. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be deterministic looping-tree automata with pairwise disjoint sets of states. For every $\mathcal{A}_i$, we consider a system of equations $E_i$ that consists of the equation of the form (4) induced by $\mathcal{A}_i$ together with the equation

$$X = \bigcup_{(q,g) \in Suc} X_{(q,g)}. \tag{5}$$

Now, let $E$ be the union of the systems $E_i$ $(i = 1, \ldots, n)$. Note that we use the same variable $X$ for every equation $E_i$. Otherwise, the equations $E_i$ do not share variables since the set of states of the automata $\mathcal{A}_i$ were assumed to be pairwise disjoint.

We need to show that $E$ has a solution iff $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_n) \neq \emptyset$. If there exists $t \in L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_n)$, then, according to Lemma 3, for every $i$ there exists a solution $\theta_i$ of the equation corresponding to $\mathcal{A}_i$ satisfying $S(t) = V_{\theta_i}$. Let $\theta$ be the substitution defined by $\theta(X_{(q,g)}) := \theta_i(X_{(q,g)})$ if $q$ is a state of $\mathcal{A}_i$, and $\theta(X) := S(t)$. Then $\theta$ solves the system $E$. Conversely, if $\theta$ is a solution of $E$, then it solves equation (4) for every automaton $\mathcal{A}_i$. In particular, by Lemma 3, there exists a tree $t_1 \in L(\mathcal{A}_1)$ such that $S(t_1) \subseteq V_\theta$. Since $\theta$ solves the equation corresponding to $\mathcal{A}_i$, Lemma 4 thus yields $t_1 \in L(\mathcal{A}_i)$ for every $i$. Thus, $t_1 \in L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_n)$. This completes the proof of the lower complexity bound stated in Theorem 2.

## 5 Least unifiers and greatest solutions

In case a unification problem is solvable, one is usually interested in obtaining an actual solution. Since a given unification problem may have infinitely many unifiers, one must decide which ones to prefer.[3] As mentioned in the introduction,

[3] From the viewpoint of unification theory, we consider ground unifiers (i.e., substitutions whose images do not contain variables). Thus, it does not make sense to employ the usual instantiation pre-order on unifiers. Anyway, the equational theory corresponding to $\mathcal{FL}_{reg}$ is of unification type zero, and thus most general unifiers or even finite complete sets of unifiers need not exist.

least unifiers are of interest in some applications. The unifier $\sigma$ is a *least unifier* of an $\mathcal{FL}_{reg}/\mathcal{FL}_0$ unification problem if it satisfies $\sigma(X) \sqsubseteq \sigma'(X)$ for all unifiers $\sigma'$ and variables $X$ occurring in the problem.

For $\mathcal{FL}_0$, least unifiers need not exist. For example, assume that $N_C = \{A\}$ and $N_R = \{r\}$. Then the (trivially solvable) unification problem $X \equiv^? X$ does not have a least unifier in $\mathcal{FL}_0$; however, $\sigma$ with $\sigma(X) = \forall r^*.A$ is the least unifier of this problem in $\mathcal{FL}_{reg}$.

It is easy to see that the least unifier of a given $\mathcal{FL}_{reg}$ unification problem corresponds to the greatest regular solution of the corresponding formal language equations. The solution $\theta$ is a *greatest solution* of an equation of the form (2) (or (3)) iff it satisfies $\theta'(X) \subseteq \theta(X)$ for all solutions $\theta'$ and variables $X$ occurring in the equation. Thus, we are interested in the existence and computability of greatest regular solutions of linear equations over regular languages.

The existence of a greatest solution of a solvable equation is obvious since the set of solutions is closed under union. In fact, if $\theta_j$, $j \in J$, are solutions of (3), then so is $\theta$ with $\theta(X) := \bigcup_{j\in J} \theta_j(X)$ for all variables $X$ occurring in the equation. Thus, the greatest solution can be obtained as the union over all solutions. However, this greatest solution can only be translated into a least unifier if it is regular. We will show that this is indeed always the case.

**Theorem 3.** *Every solvable equation of the form (3) has a greatest solution, and this solution is regular. This solution may grow exponentially in the size of (3), and it can be computed in exponential time.*

Assume that $\theta$ is the greatest solution of a solvable equation of the form (3). We first show that this solution is regular. Lemma 2 implies that there exists a corresponding run $r_\theta$ of the automaton $\mathcal{B}$ obtained from the equation (cf. Section 4). We proceed in three steps.

1. We restrict $\mathcal{B} = (Q, I, \Delta)$ to contain only so-called active states. The resulting automaton is called $\mathcal{B}' = (Q', I', \Delta')$.
2. Using $\mathcal{B}'$, we show that $r_\theta$ is regular, i.e., for every $q \in Q'$, the set $\{u \in N_R^* \mid r_\theta(u) = q\}$ is regular.
3. From $r_\theta$, finite automata accepting $\theta(X_i)$ are derived.

A state $q$ of $\mathcal{B}$ is called *active*, if $L(Q, \{q\}, \Delta) \neq \emptyset$, i.e., starting from $q$ there exists a successful run of $\mathcal{B}$. Otherwise, $q$ is called *passive*. The active states can be computed as follows. One first eliminates all states $q$ for which there exist no transitions of the form $(q, \ldots)$. One also eliminates all transitions containing these states. This process is iterated until no more states are eliminated. It is easy to see that the remaining states are exactly the active ones. Obviously, this procedure needs time polynomial in the size of $\mathcal{B}$. (There even exists a linear time algorithm for this task.) Let $\mathcal{B}' = (Q', I', \Delta')$ denote the automaton obtained from $\mathcal{B}$ by eliminating all passive states. (Note that $L(\mathcal{B}') = \emptyset$ iff $I' = \emptyset$.)

To show that $r_\theta$ is regular, we need the following partial ordering $\preceq$ on transitions of a state $q$. Let $\eta = (q, q_1, \ldots, q_k), \eta' = (q, q_1', \ldots, q_k') \in \Delta'$, $q_i = (G_i, L_i, R_i)$, and $q_i' = (G_i', L_i', R_i')$. Then, $\eta \preceq \eta'$ iff $G_i \setminus \{0\} \subseteq G_i' \setminus \{0\}$ for all

$i = 1, \ldots, k$. Note that $\preceq$ is in fact antisymmetric: If $\eta \preceq \eta'$ and $\eta \preceq \eta'$, then $G_i \setminus \{0\} = G'_i \setminus \{0\}$ for all $i = 1, \ldots, k$. Since the sets $L_i, R_i$ ($L'_i, R'_i$) are uniquely determined by $G_i$ ($G'_i$) and $0 \in G_i$ ($0 \in G'_i$) is determined by $L_i, R_i$ ($L'_i, R'_i$), this yields $\eta = \eta'$.

Now, let $u \in N_R^*$. We claim that the transition $\eta = (r_\theta(u), q_1, \ldots, q_k) \in \Delta'$, where $q_i = r_\theta(ui) =: (G_i, L_i, R_i)$, is the greatest transition among the transitions of $r_\theta(u)$ in $\mathcal{B}'$. Otherwise, there exists a transition $\eta' = (r_\theta(u), q'_1, \ldots, q'_k) \in \Delta'$, where $q'_i = (G'_i, L'_i, R'_i)$, and $i \in \{1, \ldots, k\}$ such that $G'_i \setminus \{0\} \not\subseteq G_i \setminus \{0\}$, i.e., there exists $0 \neq j \in G'_i \setminus G_i$. We can construct a new run $r'$ of $\mathcal{B}'$ that uses $\eta'$ at node $u$ instead of $\eta$. Since, by definition of $\mathcal{B}'$, the states $q'_i$ in $\eta'$ are all active, starting from these states there exist runs in $\mathcal{B}'$. Thus, a successful run $r'$ using this transition at $u$ really exists. This run corresponds to a solution of (3). However, in this solution $ui$ belongs to $X_j$ whereas this is not the case for the greatest solution, a contradiction. Thus, $\eta$ must be the greatest transition.

As a consequence, if $\mathcal{B}'$ is in the same state at different nodes, then the same transition (namely, the greatest) is used by the run $r_\theta$. From this, it easily follows that $r_\theta$ is regular: given $q \in Q'$, the following (deterministic) finite automaton $\mathcal{A}_q = (Q'', \{1, \ldots, k\}, q_I, \Delta'', \{q\})$ accepts the set $\{u \mid r_\theta(u) = q\}$:

- $Q'' := Q'$;
- $q_I := r_\theta(\varepsilon)$;
- $\Delta'' := \{(q, i, q_i) \mid (q, q_1, \ldots, q_k)$ is the greatest transition of $q$ in $\Delta'$ and $i = 1, \ldots, k\}$.

If in $\mathcal{A}_q$ the set of final states is $\{(G, L, R) \in Q' \mid i \in G\}$ instead of $\{q\}$, then this automaton accepts the language substituted for $X_i$ in the greatest solution. Thus, the greatest solution of (3) is regular. Finally, since $\mathcal{B}'$ and $\mathcal{A}_q$ can be computed in time exponential in the size of (3), the upper complexity bound for computing the greatest solution follows as well.

It remains to show that the size of the greatest solution may indeed grow exponentially. To this purpose, consider the equation

$$L_1\{1\} \cup \cdots \cup L_k\{k\} = L_1\{1\} \cup \cdots L_k\{k\} \cup X\{1, \ldots, k\}, \qquad (6)$$

where the $L_i$s are regular languages over $N_R$. Obviously, the greatest solution is the one that replaces $X$ by $L_1 \cap \cdots \cap L_k$. From results shown in [16] it follows that the size of automata accepting this intersection may grow exponentially in the size of automata accepting $L_1, \ldots, L_k$.[4]

## 6 Matching in $\mathcal{FL}_{reg}$

Matching is the special case of unification where the pattern $D$ on the right-hand side of the equation $C \equiv^? D$ does not contain variables. As an easy consequence

[4] Although these results have been shown for deterministic finite automata, they easily carry over to the nondeterministic case.

of Theorem 1, matching in $\mathcal{FL}_{reg}$ can be reduced (in linear time) to solving linear equations over regular languages of the following form:

$$S_0 \cup S_1X_1 \cup \cdots \cup S_nX_n = T_0. \tag{7}$$

For $\mathcal{FL}_0$, one obtains the same kind of equations, but there $S_0, \ldots, S_n, T_0$ are finite languages, and one is interested in finite solvability. In [8] it was shown that matching in $\mathcal{FL}_0$ is polynomial, and in [7] this result was extended to the DL $\mathcal{ALN}$.

For $\mathcal{FL}_{reg}$, matching is at least PSpace-hard since equality of regular languages is a PSpace-complete problem if one assumes that the languages are given by regular expressions or nondeterministic finite automata. Thus, the equivalence problem in $\mathcal{FL}_{reg}$ is already PSpace-complete (this corresponds to the case $n = 0$ in equation (7)). We can show that matching is not harder than testing for equivalence.

**Theorem 4.** *Matching in $\mathcal{FL}_{reg}$ is a PSpace-complete problem.*

It remains to be shown that solvability of equations of the form (7) can be decided within polynomial space. Again, we consider the mirror equation

$$S_0^{mi} \cup X_1S_1^{mi} \cup \cdots \cup X_nS_n^{mi} = T_0^{mi} \tag{8}$$

in place of the original equation (7). The main idea underlying the proof of Theorem 4 is that such an equation has a solution iff a certain candidate solution solves the equation.

**Lemma 5.** *Let $L_i := \{w \mid \{w\}S_i^{mi} \subseteq T_0^{mi}\}$. Then equation (8) has a solution iff*

$$S_0^{mi} \cup L_1S_1^{mi} \cup \cdots \cup L_nS_n^{mi} = T_0^{mi}. \tag{9}$$

*In this case, the $L_i$s yield a greatest solution of (8).*

The proof of this lemma is similar to the one for the case of finite languages given in [8]. It remains to be shown that the validity of identity (9) can be tested within polynomial space (in the size of nondeterministic finite automata for the languages $S_0^{mi}, \ldots, S_n^{mi}, T_0^{mi}$). By definition of the sets $L_i$, the inclusion from left-to-right holds iff $S_0^{mi} \subseteq T_0^{mi}$. Obviously, this can be tested in PSpace.

How to derive a PSpace-test for the inclusion in the other direction is not that obvious. Here, we sketch how the inclusion $T_0^{mi} \subseteq L_1S_1^{mi}$ can be tested (the extension to the union in identity (9) is then simple). First, we define an *exponentially large* automaton for $L_1S_1^{mi}$. However, the representation of each state of this automaton requires only polynomial space, and navigation in this automaton (i.e., determining initial states, final states, and state transitions) can also be realized within polynomial space. Thus, if we construct the automaton on-the-fly, we stay within PSpace.

An automaton $\mathcal{B}$ for $L_1 = \{w \mid \{w\}S_1^{mi} \subseteq T_0^{mi}\}$ can be obtained as follows. We construct the usual deterministic powerset automaton from the given nondeterministic automaton $\mathcal{A}$ for $T_0^{mi}$. The only difference is the definition of the final states. A state $P$ of $\mathcal{B}$ (i.e., a subset of the set of states of $\mathcal{A}$) is a final state iff $S_1^{mi} \subseteq L_{\mathcal{A}}(P)$, where $L_{\mathcal{A}}(P)$ is the language accepted by $\mathcal{A}$ if $P$ is taken as its set of initial states. It is easy to see that the automaton $\mathcal{B}$ obtained this way indeed accepts $L_1$, and that we can navigate in this automaton within PSpace. In particular, note that testing whether a state $P$ of this automaton is a final state is a PSpace-complete problem.

The automaton $\mathcal{C}$ for $L_1 S_1^{mi}$ has as states tuples, where the first component is a state of $\mathcal{B}$ and the second component is a set of states of $\mathcal{A}_1$, the nondeterministic automaton for $S_1^{mi}$. Transitions in the first component are those of $\mathcal{B}$. In the second component, they are in principle the transitions of the powerset automaton corresponding to $\mathcal{A}_1$, with the following difference: if, on input $r$, the automaton $\mathcal{B}$ reaches a final state, then in the second component we extend the set reached with $r$ in the powerset automaton of $\mathcal{A}_1$ by the initial states of $\mathcal{A}_1$. Final states of $\mathcal{C}$ are those whose second component contains a final state of $\mathcal{A}_1$. The initial state is $(I, J)$, where $I$ is the initial state of $\mathcal{B}$ and $J$ is the set of initial states of $\mathcal{A}_1$ or empty, depending on whether $I$ is a final state of $\mathcal{B}$ or not. Again, it is easy to see that navigation in $\mathcal{C}$ is possible within PSpace.

To decide whether $T_0^{mi} \subseteq L_1 S_1^{mi}$, we try to "guess" a counterexample (recall that PSpace = NPSpace). This is a word that is in $T_0^{mi}$, but not in $L_1 S_1^{mi}$. The length of a minimal such word can be bounded by the product of the size of $\mathcal{A}$ (the nondeterministic automaton for $T_0^{mi}$) and the size of $\mathcal{C}$ (the deterministic automaton for $L_1 S_1^{mi}$). We traverse $\mathcal{A}$ and $\mathcal{C}$ simultaneously, and have a counterexample if $\mathcal{A}$ is in a final state and $\mathcal{C}$ is not. The next letter and the successor state in $\mathcal{A}$ is guessed, and the successor state in $\mathcal{C}$ can be computed in PSpace. In addition, we use an exponential counter (requiring only polynomial space) that terminates the search if the (exponential) bound on the length of a minimal counterexample is reached.

## 7 Conclusion

We have shown that unification in $\mathcal{FL}_{reg}$ is equivalent via linear time reductions to solvability of linear equations over regular languages, and that these problems are ExpTime-complete. If we restrict the attention to matching problems (equations where one side does not contain variables), then the problem is PSpace-complete. In both cases, solvable problems (equations) have least (greatest) solutions, which may be exponential in the size of the problem (equation),[5] and which can be computed in exponential time. In addition to the application for description logics, we think that the results on solving linear equations over regular languages are also of interest in their own right (e.g., in formal language theory).

From the description logic point of view, one is of course also interested in unification in more expressive DLs, but this appears to be a hard problem. Recently, we have extended the decidability results to the DL obtained from

[5] Note that equation (6) actually corresponds to a matching problem.

$\mathcal{FL}_{reg}$ by adding inconsistency ($\bot$). Surprisingly, it is not clear how to handle the corresponding extension of $\mathcal{FL}_0$.

## References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The Complexity of Set Constraints. In *Proc. 1993 Conf. Computer Science Logic (CSL'93)*, volume 832 of *LNCS*, pages 1–17. European Association Computer Science Logic, Springer, 1993.
2. F. Baader. Augmenting Concept Languages by Transitive Closure of Rules: An Alternativ to Terminological Cycles. In *Proc. of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 446–451, 1991. Morgan Kaufmann Publishers.
3. F. Baader. Using Automata Theory for Characterizing the Semantics of Terminological Cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.
4. F. Baader and B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proc. of the First International Workshop on Processing Declarative Knowledge*, volume 572 of *LNCS*, pages 67–85, 1991. Springer–Verlag.
5. F. Baader and R. Küsters. Matching in Description Logics with Existential Restrictions. In *Proc. of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 261–272, 2000. Morgan Kaufmann Publishers.
6. F. Baader and R. Küsters. Unification in a Description Logic with Transitive Closure of Roles. LTCS-Report 01-05, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2001. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.
7. F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in Description Logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.
8. F. Baader and P. Narendran. Unification of Concept Terms in Description Logics. In *Proc. of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 331–335, 1998. John Wiley & Sons Ltd. An extended version has appeared in J. Symbolic Computation 31:277–305, 2001.
9. A. Borgida and D. L. McGuinness. Asking Queries about Frames. In *Proc. of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 340–349, 1996. Morgan Kaufmann Publishers.
10. R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
11. E. Leiss. Implicit language equations: Existence and uniqueness of solutions. *Theoretical Computer Science A*, 145:71–93, 1995.
12. E. Leiss. *Language Equations.* Springer-Verlag, 1999.
13. D.L. McGuinness. *Explaining Reasoning in Description Logics.* PhD thesis, Department of Computer Science, Rutgers University, October, 1996.
14. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2), 1994.
15. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, Amsterdam, 1990.
16. S. Yu and Q. Zhuang. On the State Complexity of Intersection of Regular Languages. *ACM SIGACT News*, 22(3):52–54, 1991.