Unification in a Description Logic with Transitive Closure of Roles

Franz Baader
Theoretische Informatik, RWTH Aachen, Germany
e-mail: baader@informatik.rwth-aachen.de

Ralf Küsters
Institut für Informatik und Praktische Mathematik
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 $\mathcal{F}L_0$, 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 $\mathcal{F}L_0$) is PSpace-complete in the extended description logic.

1 Introduction

Unification of concept descriptions has been introduced by Baader and Narendran [5] as a new inference service for detecting and avoiding redundancies in description logic (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 [5] 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 (ε), 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 main 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. This is, however, only an interesting observation from the linear equation point of view. From the point of view of unification in $\mathcal{FL}_{reg}$, irregular solutions do not make sense since they don't correspond to $\mathcal{FL}_{reg}$-concept descriptions.

Equations over regular languages have already been considered by Leiss [7, 6]. 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 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

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

equations than the linear ones considered here.

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 (ε), 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 $\mathrm{Woman} \sqcap \forall \mathrm{child}^*.\mathrm{Woman}$, which represents the set of all women with only female offspring.

Role names will henceforth 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 ε 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 language.

The semantics of concept descriptions built from these constructors is defined in the usual way (see, e.g., [2]). $\mathcal{FL}_{reg}$-concept descriptions can also be viewed as concepts defined by cyclic $\mathcal{FL}_0$-TBoxes interpreted with the greatest fixed-point semantics [2]. Two concept descriptions C, D are *equivalent* ($C \equiv D$) iff they denote the same concept in every interpretation (i.e., $C^I = D^I$ for all interpretations I).

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* σ 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.,

- $\sigma(A) := A$ for all $A \in N_C$,
- $\sigma(\top) := \top$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$, 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* σ *is a* unifier *of this problem iff* $\sigma(C) \equiv \sigma(D)$*. In this case, the unification problem is called 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

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

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.

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 Σ 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 $S_0, S_1, \ldots, S_n, T_0, T_1, \ldots, T_n$ over N_R, a *linear equation over regular languages* is of the form

$$S_0 \cup S_1X_1 \cup \cdots \cup S_nX_n = T_0 \cup T_1X_1 \cup \cdots \cup T_nX_n \tag{2}$$

A *(regular) solution* θ of this equation is a substitution assigning to each variable a *(regular)* language over Σ such that the equation holds. We are particularly interested in *regular* solutions since only 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 θ 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 [5] 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 languages over N_R.[2] Every concept pattern can (in polynomial time) be turned into such a normal form by exhaustively applying the following equivalence preserving normalization rule

$$\forall R.C \sqcap \forall R'.C \longrightarrow \forall (R \cup R').C,$$

where R, R' are regular languages 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 characterization of equivalence between $\mathcal{FL}_{reg}$-concept descriptions:

Lemma 2 *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 3 *Let C, D be $\mathcal{FL}_{reg}$-concept patterns such that*

$$\begin{aligned} C &\equiv \bigsqcap_{A \in N_C} \forall S_A.A \sqcap \bigsqcap_{X \in N_X} \forall S_X.X, \\ D &\equiv \bigsqcap_{A \in N_C} \forall T_A.A \sqcap \bigsqcap_{X \in N_X} \forall T_X.X. \end{aligned}$$

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

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

has a solution.

Note that the language equations in this system do not share variables, and thus they can be solved separately.[3]

[2] Strictly speaking, R_A and R_X are regular expressions describing regular languages. 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.

[3] 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 σ of (1).

4 Solving regular language equations

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

Theorem 4 *Deciding the solvability of (systems of) equations of the form (2) is an ExpTime-complete.*

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

To prove the theorem, 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).

In principle, to solve (3), we build a Büchi tree-automaton that accepts the trees representing i) sets of words obtained by instantiating the equation with its solutions (called *solution sets* in the following), and ii) the solution itself, i.e., the languages substituted for the variables.

The trees the Büchi-automaton is working on are total mappings from the set N_R^* into $\{0,1\}^{n+1}$. That is, a tree t is a $\{0,1\}^{n+1}$-labeled $|N_R|$-ary infinite tree. For $v \in N_R^*$, $t(v)_i$, $i = 0, \ldots, n$, denotes the ith component of $t(v)$.

Every path from the root of t to some node $v \in N_R^*$ corresponds to a (finite) word over N_R, namely v. The label $t(v) = b_0 b_1 \cdots b_n \in \{0,1\}^{n+1}$ of v should be read as follows: v belongs to the solution set of (3) iff $b_0 = 1$; and (the language substituted for) X_i, $i = 1, \ldots, n$, contains v iff $b_i = 1$. Thus, such a tree completely determines the solution set as well as the substitutions for the variables, provided that the substitution really solves the equation.

For a given tree t, we need to check whether the encoded solution set really corresponds to the result of inserting the encoded substitution into the right- and the left-hand side of equation (3). This can be done using a Büchi-automaton that checks whether the following conditions are equivalent for every $v \in N_R^*$:

1. $t(v)_0 = 1$.
2. $v \in S_0^{mi}$, or there exists $i = 1, \ldots, n$ and w, w' such that $v = ww'$, $t(w)_i = 1$, and $w' \in S_i^{mi}$.
3. $v \in T_0^{mi}$, or there exists $i = 1, \ldots, n$ and w, w' such that $v = ww'$, $t(w)_i = 1$, and $w' \in T_i^{mi}$.

It is fairly easy to construct a Büchi tree-automaton accepting exactly those trees satisfying the above equivalence. The size of the set of states of this automaton turns out to be exponential in the size of the equation, where the size of the regular sets S_i^{mi} and T_i^{mi} are measured by the size of non-deterministic finite automata accepting these sets. Moreover, all states of the Büchi-automaton are final. Thus, we are using a restricted form of Büchi-automata (sometimes called *looping tree-automata* in the literature). Since the emptiness problem for Büchi tree-automata can be solved in polynomial time in the size of the automaton [9] (and actually in linear time for looping automata), this yields an exponential time algorithm deciding whether an equation of the form (3) has a solution. However, the existence of a solution does not a priori imply that there is also a regular one.

It is well known [9] that the set of trees accepted by a Büchi-automaton contains a regular (or rational) tree t. As also stated in [9], a tree t is regular iff for every label ℓ the set $\{v \mid t(v) = \ell\}$ is regular. In our setting this means that the language $\{v \mid t(v) = b_0 \cdots b_n\}$ for some fixed label $b_0 \cdots b_n \in \{0,1\}^{n+1}$ is regular. In particular, the finite union

$$\bigcup_{b_0 b_1 \cdots b_n \in \{0,1\}^{n+1}, b_i = 1} \{v \mid t(v) = b_0 \cdots b_n\},$$

which describes the language substituted for X_i, is regular. Consequently, we have shown the following proposition.

Proposition 5 *If (3) has a solution, then it also has a regular one.*

As a direct consequence of this proposition we obtain the exponential upper bound claimed in Theorem 4.

The hardness result can be shown similarly to the proof by Baader and Narendran [5] for systems of equations over finite languages. In their proof, the intersection problem of deterministic root-to-frontier automata on finite trees, which has been shown to be ExpTime-complete by Seidl [8], is reduced to the solvability of systems of equations over finite languages. One can use Seidl's result to show that the intersection problem for deterministic looping tree-automata is ExpTime-complete as well. A reduction of this problem to solvability of systems of linear equations over regular languages then establishes the exponential lower bound.

5 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 of Theorem 3, matching in $\mathcal{FL}_{reg}$ can be reduced (in linear time) to solving linear equations over regular languages of the following form:

$$S_0 \cup S_1 X_1 \cup \cdots \cup S_n X_n = T_0. \tag{4}$$

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 [5] it was shown that matching in $\mathcal{FL}_0$ is polynomial, and in [4] 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 non-deterministic finite automata. Thus, the equivalence problem in $\mathcal{FL}_{reg}$ is already PSpace-complete (this corresponds to the case $n = 0$ in equation (4)). We can show that matching is not harder than testing for equivalence.

Theorem 6 *Matching in $\mathcal{FL}_{reg}$ is a PSpace-complete problem.*

It remains to be shown that solvability of equations of the form (4) can be decided within polynomial space. Again, we consider the mirror equation

$$S_0^{mi} \cup X_1 S_1^{mi} \cup \cdots \cup X_n S_n^{mi} = T_0^{mi} \tag{5}$$

in place of the original equation (4). The main idea underlying the proof of Theorem 6 is that such an equation has a solution iff a certain candidate solution solves the equation.

Lemma 7 *Let $L_i := \{w \mid \{w\} S_i^{mi} \subseteq T_0^{mi}\}$. Then equation (5) has a solution iff*

$$S_0^{mi} \cup L_1 S_1^{mi} \cup \cdots \cup L_n S_n^{mi} = T_0^{mi}. \tag{6}$$

The proof of this lemma is similar to the one for the case of finite languages given in [5]. It remains to be shown that the validity of identity (6) can be tested within polynomial space (in the size of non-deterministic 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 $T_0^{mi} \subseteq L_1 S_1^{mi}$ can be tested (the extension to the union in identity (6) is then simple). First, we define an *exponentially large* automaton for $L_1 S_1^{mi}$. 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 non-deterministic 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 non-deterministic 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 non-deterministic 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.

6 Future work

In case a unification problem is solvable, one is usually interested in an actual solution. It is easy to see that solutions can be derived from accepting runs of our Büchi-automata. In the context of matching in description logics [4, 3], it has been argued that not all solutions of a matching problem are of interest for a user. Therefore, one must look for solutions with desired properties; for instance, least solutions where all variables are substituted by concept descriptions that are as specific as possible. For matching in $\mathcal{FL}_{reg}$, the candidate solutions used to decide solvability of matching problems yield such least solutions. For unification in $\mathcal{FL}_{reg}$, we can show that solvable problems always have least solutions. What is not clear yet is how hard it is to compute them.

Finally, one is of course also interested in unification in more expressive DLs, for example, those that allow for number restrictions, existential restrictions, disjunction, or negation.

References

[1] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The Complexity of Set Constraints. In *Proceedings 1993 Conf. Computer Science Logic (CSL'93)*, volume 832 of *Lecture Notes in Computer Science*, pages 1–17, 1993

[2] F. Baader. Augmenting Concept Languages by Transitive Closure of Rules: An Alternativ to Terminological Cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 446–451, 1991.

[3] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 261–272, 2000.

[4] F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in Description Logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.

[5] F. Baader and P. Narendran. Unification of concept terms in description logics. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 331–335, Brighton, UK, 1998. An extended version has appeared in J. Symbolic Computation 31:277–305, 2001.

[6] E. Leiss. Implicit language equations: Existence and uniqueness of solutions. *Theoretical Computer Science A*, 145:71–93, 1995.

[7] E. Leiss. *Language Equations.* Springer-Verlag, 1999.

[8] H. Seidl. Haskell overloading is DExpTime-complete. *Information Processing Letters*, 52(2), 1994.

[9] 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.