Solving Linear Equations over Regular Languages

Franz Baader[1] and Ralf Küsters[2]

[1] LuFG Theoretische Informatik, RWTH Aachen, Germany,
baader@informatik.rwth-aachen.de
[2] Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany,
kuesters@ti.informatik.uni-kiel.de

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 knowledge bases. The technical results in [5] are concerned with unification in the small Description Logic $\mathcal{FL}_0$, which allows for conjunction of concepts, value restriction, and the top concept. It is shown that unification of $\mathcal{FL}_0$-concept descriptions is equivalent to unification modulo the equational theory ACUIh, which is concerned with a binary associative-commutative-idempotent function symbol with a unit and finitely many unary function symbols that behave like endomorphisms for the binary symbol and the unit. Since ACUIh is a commutative/monoidal theory [6], unification modulo this theory can in turn be reduced to solving linear equations in a certain semiring. For the theory ACUIh, this semiring consists of finite sets of words over a finite alphabet Σ, with union as addition and concatenation as multiplication. Thus, the linear equations to be solved are of the following form:

$$S_0 \cup X_1 S_1 \cup \cdots \cup X_n S_n = T_0 \cup X_1 T_1 \cup \cdots \cup X_n T_n, \tag{1}$$

where the coefficients S_i, T_i are finite sets of words over Σ and the X_i are variables that stand for finite sets of words. In [5] it was shown that the problem of determining whether such an equation has a solution or not is ExpTime-complete.

Closely related to the problem of solving such linear language equations is the problem of solving positive set constraints [1] 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]. The main difference between this result and the one in [5] is that solutions of set constraints may be infinite, i.e., in the setting of linear language equations, the variables X_i may be replaced by (possibly) *infinite* sets of words.

In the present paper, we study the problem of solving equations of the form (1) over regular languages, i.e., both the coefficients S_i, T_i and the languages substituted for the variables X_i may be regular languages over Σ. Equations over regular languages have already been considered by Leiss [7,8]. However, he does not provide any decidability or complexity results for the case we are interested in. From the Description Logic point of view, the problem of solving such equations corresponds to unification in the Description Logic $\mathcal{FL}_{reg}$ [2], which extends $\mathcal{FL}_0$ by composition, union, and transitive closure of roles.

Our main contribution is that the problem is also ExpTime-complete. This result is obtained 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. Using this result, our exponential upper bound also follows from the complexity result in [1], if one takes into account that regular coefficients can be expressed using set constraints. The hardness result in [1] does not directly carry over since set constraints allow for more complex types of equations than the linear ones considered here.

2 Solving regular language equations

Let Σ be a finite alphabet. A *linear equation over regular languages* is an equation of the form (1) such that the coefficients S_i, T_i are regular languages over Σ. A *solution* θ of this equation is a substitution assigning to each variable X_i a language $L_i \subseteq \Sigma^*$ such that the equation holds, i.e., $S_0 \cup L_1 S_1 \cup \cdots \cup L_n S_n = T_0 \cup L_1 T_1 \cup \cdots \cup L_n T_n$ where for languages L, S the expression LS denotes their concatenation $\{vw \mid v \in L, w \in S\}$. This solution is called *regular* (*finite*) iff the languages L_i are regular (finite).

For example, let $\Sigma := \{a\}$ and consider the linear equation

$$\{a\} \cup X\{a\} = X.$$

The substitution that replaces X by $\{a\}\{a\}^*$ is a regular solution of this equation. Note that the coefficients are all finite, but that the equation does not have a finite solution.

Instead of only single equations, one can also look at *systems of regular language equations*, i.e., finite sets of equations. A substitution θ solves a system of regular language equations if it solves every equation in it simultaneously. However, 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 symbol a (a new symbol for every equation), i.e., the languages S_i and T_i are replaced by $S_i\{a\}$ and $T_i\{a\}$. 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.

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

Theorem 1. *Deciding whether a linear equation over regular languages has a (regular) solution is an ExpTime-complete problem.*

As an immediate consequence of the result for regular solvability, unification in $\mathcal{FL}_{reg}$ is ExpTime-complete as well.

In principle, to solve (1), 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 Σ^* into $\{0,1\}^{n+1}$. That is, a tree t is a $\{0,1\}^{n+1}$-labeled $|\Sigma|$-ary infinite tree. For $v \in \Sigma^*$, $t(v)_i$, $i = 0, \ldots, n$, denotes the ith component of $t(v)$.

Every path from the root of t to some node $v \in \Sigma^*$ corresponds to a (finite) word over Σ, 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 (1) 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 (1). This can be done using a Büchi-automaton that checks whether the following conditions are equivalent for every $v \in \Sigma^*$:

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

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 and T_i 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 [11]). Since the emptiness problem for Büchi tree-automata can be solved in polynomial time in the size of the automaton [10] (and actually in linear time for looping automata), this yields an exponential time algorithm deciding whether an equation of the form (1) has a solution. However, this does not a priori mean that there is also a regular one.

It is well known [10] that the set of trees accepted by a Büchi-automaton contains a regular (or rational) tree t if it is nonempty. In our setting this implies 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

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

is regular. This set describes the language substituted for X_i. This shows the following proposition.

Proposition 1. *If a linear equation over regular languages 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 1 also for the regular case.

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 [9], 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.

3 Conclusion and open problems

We have shown that deciding the solvability of (systems of) linear equations over regular languages, a problem equivalent (via linear-time reduction) to unification in the Description Logic $\mathcal{FL}_{reg}$, is ExpTime-complete.

In case an equation/unification problem is solvable, one is usually interested in obtaining an actual solution. It is easy to see that solutions can be derived from accepting runs of our Büchi-automata. In addition, if there is such a run, it is also possible to compute enough information about it to read off the solution it describes. However, since an equation may have several (even infinitely many) solutions, one must ask which solutions should be computed.

In the context of matching in Description Logics [3,4], it has been argued that least solutions, i.e., solutions where all variables are substituted by concept descriptions that are as specific as possible, are of interest. For our language equations this would mean that variables are replaced by the greatest (w.r.t. set inclusion) (regular) languages. Thus, the following questions arise. Do solvable linear equations over regular languages always have such greatest solutions? Are these solutions regular? (Only in this case can they be translated into $\mathcal{FL}_{reg}$-concept descriptions.) How hard is it to compute greatest solutions?

Another open problem is the complexity of matching in $\mathcal{FL}_{reg}$. This corresponds to solving linear equations over regular languages where the right-hand side does not contain variables (i.e., $T_1 = \ldots = T_n = \emptyset$). For linear equations over finite languages matching is polynomial. In the regular case, it is easy to see that the problem is at least PSpace-hard, but it is not clear whether it can be solved in PSpace.

Finally, for Description Logics one is also interested in unification in more expressive logics. On the language equation side, this could lead to other (probably more involved) kinds of equations, if the language-based approach is applicable at all. For matching, the language-based approach can also be applied to more expressive Description Logics (allowing for atomic negation and number restrictions) [4], but for some Description Logics (e.g., ones allowing for existential restrictions) other approaches are more appropriate [3]. For unification, nothing is known for these more expressive languages.

References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The Complexity of Set Constraints. In *Proc.CSL'93*, LNCS 832, pages 1–17, 1993.
2. F. Baader. Augmenting concept languages by transitive closure of roles: an alternative to terminological cycles. In *Proc. IJCAI-91*, pages 446–451, 1991.
3. F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proc. KR2000*, pages 261–272, Breckenridge, CO, 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 *Proc. ECAI-98*, pages 331–335, 1998. An extended version has appeared in J. Symbolic Computation 31:277–305, 2001.
6. F. Baader and W. Nutt. Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning. *J. Applicable Algebra in Engineering, Communication and Computing*, 7(4):309–337, 1996.
7. E. Leiss. Implicit language equations: Existence and uniqueness of solutions. *Theoretical Computer Science A*, 145:71–93, 1995.
8. E. Leiss. *Language Equations.* Springer-Verlag, 1999.
9. H. Seidl. Haskell overloading is DExpTime-complete. *Information Processing Letters*, 52(2), 1994.
10. 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.
11. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.